Міністерство освіти і науки України

Державний університет "Житомирська політехніка"

Кваліфікаційна наукова
праця на правах рукопису

САВІЦЬКИЙ РОМАН СВЯТОСЛАВОВИЧ

УДК 004.4:[004.91+004.514]

ДИСЕРТАЦІЯ

МЕТОДИ І АЛГОРИТМИ ЗАБЕЗПЕЧЕННЯ ВЕБДОСТУПНОСТІ ТЕКСТУ ДЛЯ ЛЮДЕЙ З КОГНІТИВНИМИ ТА ІНТЕЛЕКТУАЛЬНИМИ ПОРУШЕННЯМИ

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело



Р. С. Савіцький

Науковий керівник **Вакалюк Тетяна Анатоліївна**, доктор педагогічних наук, професор

Житомир 2026

АНОТАЦІЯ

Савіцький Р.С. Методи і алгоритми забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 121 "Інженерія програмного забезпечення". – Державний університет "Житомирська політехніка". – Житомир, 2026.

Зміст анотації

Дисертаційну роботу присвячено розробці методів і алгоритмів забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями. Когнітивні та інтелектуальні порушення охоплюють широкий спектр станів – від синдрому дефіциту уваги та специфічних порушень навчання до розладів аутистичного спектра й набутих когнітивних дефіцитів, – що суттєво обмежують сприйняття й опрацювання інформації. За даними Всесвітньої організації охорони здоров'я, понад мільярд людей мають ту чи іншу форму інвалідності, і складність текстового контенту залишається для них системним бар'єром в освіті та суспільному житті.

Міжнародні стандарти WCAG 2.2 та напрацювання COGA W3C закріплюють загальні принципи когнітивної доступності, проте не надають обчислюваних метрик для автоматизованої оцінки складності тексту. Класичні індекси читабельності (Flesch–Kincaid, Gunning Fog, SMOG) розроблені для англомовних текстів і для морфологічно складних мов, зокрема української, дають ненадійні результати. Наявні системи спрощення реалізують статичний підхід без урахування стану конкретного читача.

Для України проблема набуває гостроти в контексті гармонізації законодавства з Директивою ЄС 2016/2102 та переходу освіти до дистанційних форм, тоді як більшість освітніх вебресурсів не має адаптованих версій. Поєднання автоматизованого аналізу тексту з неінвазивним моніторингом емоційного стану відкриває можливість адаптивного спрощення, орієнтованого на реальний стан користувача.

Поєднання автоматизованого аналізу тексту засобами обробки природної мови з неінвазивним моніторингом емоційного стану користувача через комп'ютерний зір відкриває можливість адаптивного спрощення, орієнтованого на реальний стан конкретного читача, що формує основу для побудови гібридного методу забезпечення когнітивної доступності вебтексту.

Метою дослідження є розробка методів та алгоритмів забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями.

У першому розділі «Аналіз проблеми дослідження вебдоступності для людей з когнітивними та інтелектуальними порушеннями» встановлено, що когнітивна вебдоступність є окремою науково-практичною проблемою, яка не зводиться до вимог сенсорної або моторної доступності. Проаналізовано міжнародні стандарти WCAG 2.2, ATAG, UAAG та напрацювання COGA W3C і встановлено, що чинна нормативна рамка лише частково охоплює когнітивну доступність, визначаючи загальні принципи без обчислюваних метрик. Досліджено національне законодавство України, зокрема ДСТУ 8845:2019 та Постанову КМУ № 757 від 21.07.2023, що закріплюють вимоги до доступності вебресурсів. Проведено систематичний огляд наявних методів оцінювання та спрощення текстового контенту, що охоплює класичні індекси читабельності, засновані на правилах системи, підходи машинного навчання, нейромережеві та генеративні моделі. Аналіз показав, що жоден з методів окремо не забезпечує одночасно пояснюваного оцінювання, точної локалізації лексичних і синтаксичних бар'єрів, контрольованого ітераційного спрощення та відтворюваності результатів, що обґрунтовує доцільність гібридного підходу з поєднанням детермінованих словникових механізмів та генеративних моделей.

У другому розділі «Методи забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями» запропоновано індекс когнітивної доступності тексту (ІКД) як інтегральний нормований показник у діапазоні [0; 1], що поєднує три компоненти: лексичний (мультиплікативна модель за сімома категоріями бар'єрів: рідковживані слова, терміни, складна

лексика, архаїзми, абревіатури, сленг, ненормативна лексика), синтаксичний (зважена сума чотирьох факторів: структура речень, підрядна складність, глибина дерева залежностей, частка речень зі структурою «підмет-присудок-додаток») та компонент читабельності з перекалібруванням класичних формул для української мови методом OLS-регресії. Вагові коефіцієнти компонентів визначено за результатами експертного оцінювання із застосуванням конкордаційного аналізу Кендалла (W = 0,929). Розроблено метод визначення стабільної емоції розуміння тексту на основі розпізнавання виразу обличчя через відеокамеру в реальному часі із застосуванням бібліотеки DeepFace з бекендом VGG-Face, матрицею відображення семи базових емоцій у три когнітивні стани (спокійний, дезорієнтований, перевантажений), чотирма незалежними агрегаторами (короткостроковий та довгостроковий для емоцій і станів) та правилом негативного консенсусу. Сформовано спеціалізовані словники синонімічних замін семи категорій проблемної лексики загальним обсягом 54 631 лема та 467 872 словоформи з шестиетапною процедурою формування, контролю якості та експертної валідації. Побудовано узагальнений метод забезпечення когнітивної доступності як керований ітераційний процес із функціоналом корисності, трьома режимами деталізації (документний, сторінковий, курсорний), формальними критеріями зупинки та гібридною архітектурою замін із детермінованим словниковим рівнем та резервним генеративним рівнем на основі великих мовних моделей.

У третьому розділі «Програмна реалізація забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями» здійснено алгоритмічну реалізацію узагальненого методу як браузерного програмного комплексу з трирівневою архітектурою: клієнтський рівень (React), серверний модуль аналізу та спрощення (Flask, NLP-конвеєр Stanza, морфологічний аналізатор pymorphy2) та сервіс емоційного контексту (DeepFace, OpenCV). Реалізовано алгоритм ітераційного спрощення з п’ятьма функціональними етапами, детерміновану ієрархію пошуку замін з чотирьох рівнів охоплення, гібридну інтеграцію великих мовних моделей та NER-модель виявлення термінів

на основі архітектури RoBERTa. Комплекс доповнено підсистемами безпрокруткової навігації, підсвітки бар'єрів, синтезу мовлення та інтернаціоналізації. Серія з восьми обчислювальних експериментів підтвердила конвергенцію алгоритму за 1–3 ітерації, перевагу ІКД над класичними індексами у виявленні семантичних бар'єрів (10/10 проти 0/10 коректних ідентифікацій), збереження семантики тексту (косинусна подібність 0,92) та тональності (відхилення $\leq 0{,}09$). Результати експертного оцінювання програмного комплексу підтвердили його функціональну придатність (середня оцінка 4,25/5, коефіцієнт конкордації $W = 0{,}661$, коефіцієнт кореляції Спірмена $\rho = 0{,}661$).

Наукова новизна та теоретичне значення отриманих результатів полягають у тому, що *вперше:*

- запропоновано індекс когнітивної доступності тексту як інтегральний нормований показник, що поєднує лексичний, синтаксичний і компонент читабельності в єдиній шкалі, адаптованій для української мови та цільової групи користувачів, на відміну від наявних метрик читабельності, які орієнтовані на англомовні тексти і не враховують специфічних когнітивних бар'єрів сприйняття;

- розроблено метод визначення стабільної емоції на основі розпізнавання виразу обличчя через відеокамеру пристрою в реальному часі, що дозволяє використовувати емоційний стан як індикатор когнітивного навантаження в задачах забезпечення спрощення тексту;

- розроблено узагальнений метод та алгоритм забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями, що поєднує статичний аналіз ІКД з динамічним управлінням на основі стабілізованої емоції та спрощенні тексту.

Практичне значення отриманих результатів полягає в тому, що:

– розроблено програмний комплекс забезпечення когнітивної доступності україномовного вебтексту із застосуванням трирівневої архітектури, що реалізує повний цикл від аналізу когнітивної складності до ітераційного спрощення та

може бути впроваджений як інструмент підтримки читання для осіб з когнітивними та інтелектуальними порушеннями;

– сформовано спеціалізовані словники синонімічних замін семи категорій проблемної лексики з детермінованою ієрархією підстановок та морфологічним узгодженням, що становлять лінгвістичний ресурс, придатний для повторного використання в інших системах обробки природної мови;

– розроблено модуль розпізнавання емоційного стану користувача з двоканальною стабілізацією та програмною калібрацією параметрів на базі телеметричного корпусу, що може використовуватися як самостійний компонент неінвазивного моніторингу когнітивного навантаження;

– проведено серію з восьми обчислювальних експериментів, результати яких підтвердили конвергенцію алгоритму за 1–3 ітерації, збереження семантики та масштабованість методу.

Матеріали дослідження можуть бути використані освітніми, соціальними та адміністративними установами для забезпечення когнітивної доступності текстового вебконтенту для людей з когнітивними та інтелектуальними порушеннями.

Ключові слова: методи та алгоритми, вебдоступність, когнітивні та інтелектуальні порушення, індекс когнітивної доступності, когнітивне моделювання, когнітивна продуктивність, методологія, великі мовні моделі, стабільна емоція, розпізнавання образів, машинний зір, нейронні мережі, штучний інтелект, машинне навчання, глибоке навчання.

ABSTRACT

Savitskyi R.S. Methods and algorithms for web text accessibility analysis for people with cognitive and intellectual disabilities. – Qualifying scientific paper of manuscript copyright.

The dissertation for a Doctor of Philosophy degree, speciality 121 – “Software Engineering”. – Zhytomyr Polytechnic State University. – Zhytomyr, 2026.

Abstract content

The dissertation is devoted to the development of methods and algorithms to ensure web accessibility of text for people with cognitive and intellectual impairments. Cognitive and intellectual impairments encompass a broad spectrum of conditions—ranging from attention deficit disorder and specific learning disabilities to autism spectrum disorders and acquired cognitive deficits—that substantially limit the perception and processing of information. According to the World Health Organisation, more than one billion people have some form of disability, and the complexity of textual content remains a systemic barrier to their education and participation in social life.

The international WCAG 2.2 standards and the work of the W3C COGA Task Force establish general principles of cognitive accessibility, yet they do not provide computable metrics for the automated assessment of text complexity. Classical readability indices (Flesch–Kincaid, Gunning Fog, SMOG) were designed for English-language texts and yield unreliable results for morphologically complex languages, in particular Ukrainian. Existing simplification systems use a static approach that does not account for the reader's state.

For Ukraine, the problem becomes especially acute in the context of harmonising national legislation with EU Directive 2016/2102 and the transition of education to distance learning formats, while the majority of educational web resources lack adapted versions. Combining automated text analysis with non-invasive monitoring of emotional state opens the possibility of adaptive simplification tailored to the user's actual state.

Combining automated text analysis via natural language processing with non-invasive computer vision-based monitoring of the user's emotional state opens the possibility of adaptive simplification tailored to the individual reader's current state, forming the basis for a hybrid method to ensure the cognitive accessibility of web text.

The research aims to develop methods and algorithms to ensure web accessibility of text for people with cognitive and intellectual impairments.

In the first chapter, "Analysis of the Research Problem of Web Accessibility for People with Cognitive and Intellectual Impairments", it is established that cognitive web accessibility constitutes a distinct scientific and practical problem that cannot be reduced to the requirements of sensory or motor accessibility. The international WCAG 2.2, ATAG, and UAAG standards and the work of the W3C COGA Task Force are analysed, and it is established that the current regulatory framework only partially covers cognitive accessibility, defining general principles without computable metrics. The national legislation of Ukraine is examined, in particular DSTU 8845:2019 and Resolution of the Cabinet of Ministers of Ukraine No. 757 of 21.07.2023, which establish requirements for the accessibility of web resources. A systematic review of existing methods for assessing and simplifying textual content is conducted, covering classical readability indices, rule-based systems, machine learning approaches, and neural-network and generative models. The analysis showed that no method individually provides, at the same time, explainable assessment, precise localisation of lexical and syntactic barriers, controlled iterative simplification, and reproducibility of results, which justifies the expediency of a hybrid approach combining deterministic dictionary-based mechanisms with generative models.

In the second chapter, "Methods for Ensuring the Web Accessibility of Text for People with Cognitive and Intellectual Impairments", a Text Cognitive Accessibility Index (TCAI) is proposed as an integral normalised indicator in the range [0; 1] that combines three components: a lexical component (a multiplicative model across seven categories of barriers: rare words, terms, complex vocabulary, archaisms, abbreviations, slang, and profanity), a syntactic component (a weighted sum of four factors: sentence structure, subordinate-clause complexity, dependency-tree depth, and

the proportion of sentences with a subject–verb–object structure), and a readability component obtained by recalibrating classical formulas for the Ukrainian language using OLS regression. The component weighting coefficients were determined based on expert evaluation results using Kendall's concordance analysis (W = 0.929). A method for determining the stable emotion of text comprehension is developed, based on real-time facial-expression recognition via a video camera using the DeepFace library with the VGG-Face backend, a mapping matrix of seven basic emotions onto three cognitive states (calm, disoriented, overloaded), four independent aggregators (short-term and long-term for both emotions and states), and a negative-consensus rule. Specialised synonym dictionaries for seven categories of problematic vocabulary are compiled, totalling 54,631 lemmas and 467,872 word forms, using a six-stage construction, quality control, and expert validation process. A generalised method for ensuring cognitive accessibility is constructed as a controlled, iterative process with a utility function, three levels of detail (document-level, page-level, and cursor-level), formal stopping criteria, and a hybrid substitution architecture with a deterministic dictionary layer and a fallback generative layer based on large language models.

In the third chapter, "Software Implementation of Web Accessibility of Text for People with Cognitive and Intellectual Impairments", an algorithmic implementation of the generalised method is carried out as a browser-based software system with a three-tier architecture: a client tier (React), a server-side analysis and simplification module (Flask, the Stanza NLP pipeline, and the pymorphy2 morphological analyser), and an emotional-context service (DeepFace, OpenCV). An iterative simplification algorithm with five functional stages, a deterministic substitution-search hierarchy with four coverage levels, hybrid integration of large language models, and a RoBERTa-based NER model for term detection are implemented. The system is augmented with subsystems for scroll-free navigation, barrier highlighting, speech synthesis, and internationalisation. A series of eight computational experiments confirmed the convergence of the algorithm within 1–3 iterations, the superiority of the TCAI over classical indices in detecting semantic barriers (10/10 versus 0/10 correct identifications), the preservation of text semantics (cosine similarity of 0.92),

and the preservation of sentiment (deviation ≤ 0.09). The results of the expert evaluation of the software system confirmed its functional suitability (mean score of 4.25/5, concordance coefficient W = 0.661, Spearman's correlation coefficient ρ = 0.661).

The scientific novelty and theoretical significance of the results lie in the fact that:

for the first time, a cognitive accessibility index of text has been proposed as an integral normalised indicator that combines lexical, syntactic, and readability components within a unified scale adapted for the Ukrainian language and the target user group, unlike existing readability metrics oriented toward English-language texts that do not account for the specific cognitive barriers of perception; method for determining stable emotion has been developed based on real-time facial expression recognition via the device camera, enabling the use of emotional state as an indicator of cognitive load in text simplification tasks; a generalised method and algorithm for ensuring web accessibility of text for people with cognitive and intellectual disabilities has been developed, combining static cognitive accessibility index analysis with dynamic management based on stabilised emotion and text simplification.

The practical significance of the results obtained lies in the fact that:

– a software suite has been developed to ensure the cognitive accessibility of Ukrainian-language web text using a three-level architecture that implements the full cycle from cognitive complexity analysis to iterative simplification and can be implemented as a reading support tool for people with cognitive and intellectual impairments;

– specialised dictionaries of synonymic substitutions for seven categories of problematic vocabulary have been compiled, featuring a deterministic substitution hierarchy and morphological alignment, constituting a linguistic resource suitable for reuse in other natural language processing systems;

– a module for recognising the user’s emotional state has been developed, featuring dual-channel stabilisation and software calibration of parameters based on a

telemetric corpus, which can be used as a standalone component for non-invasive monitoring of cognitive load;

– a series of eight computational experiments was conducted, the results of which confirmed the convergence of the algorithm within 1–3 iterations, the preservation of semantics and the scalability of the method.

Educational, social and administrative institutions can use the research materials to ensure the cognitive accessibility of text-based web content for people with cognitive and intellectual impairments.

Keywords: methods and algorithms, web accessibility, cognitive and intellectual impairments, cognitive accessibility index, cognitive modelling, cognitive performance, methodology, large language models, stable emotion, image recognition, computer vision, neural networks, artificial intelligence, machine learning, deep learning.

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Наукові праці в фахових виданнях України, в яких опубліковані основні наукові результати дисертації

1. Савіцький Р. С. Безбар’єрність вебпорталів освітніх навчальних закладів України. *Технічна інженерія.* 2023. № 1 (91). С. 172–177. DOI: 10.26642/ten-2023-1(91)-172-177
2. Савіцький Р. С. Гібридна система виявлення термінів в українськомовних текстах для забезпечення когнітивної доступності. *Computer Science and Applied Mathematics.* 2025. № 2. С. 47–54. DOI: 10.26661/2786-6254-2025-2-06.
3. Савіцький Р. С., М’яновська А. Ю. Інтеграція бібліотек NLP та попередньо навчених моделей для автоматичної оцінки складності тексту. *Вісник Херсонського національного технічного університету.* 2025. № 2 (93). С. 319–326. DOI: 10.35546/kntu2078-4481.2025.2.2.39.
4. Савіцький Р., Вакалюк Т. Гібридна система генерації лексичних замін україномовного тексту для підвищення когнітивної доступності. *Вісник Хмельницького національного університету. Серія: Технічні науки.* 2026. № 2 (363). С. 182–188. DOI: 10.31891/2307-5732-2026-363-24.

Наукові праці, що засвідчують апробацію матеріалів дисертації

5. Савіцький Р. С. Стандарти доступності у питанні вивчення когнітивних та інтелектуальних порушень. *Інформаційно-комп’ютерні технології – 2024*: *тези XIV Міжнар. наук.-техн. конф. (Житомир, 28–29 берез. 2024 р.).* Житомир, 2024. С. 164.
6. Савіцький Р. С. Синхронізація українського законодавства з ЄС для покращення вебдоступності в освіті. *Автоматизація та комп’ютерно-інтегровані технології у виробництві та освіті: стан, досягнення, перспективи розвитку: тези всеукр. наук.-практ. інтернет-конф. (Черкаси, 11–17 берез. 2024 р.).* Черкаси: Черкас. нац. ун-т ім. Б. Хмельницького, 2024. С. 268.

7. Бабицький Б. О., Савіцький Р. С., Желізко В. В. Аналіз методів розпізнавання облич. *Комп'ютерні інтелектуальні системи та мережі: тези доп. XVIII Всеукр. наук.-практ. WEB-конф. аспірантів, студентів та молодих вчених (Кривий Ріг, 25–27 берез. 2025 р.).* Кривий Ріг: КНУ, 2025. С. 175–178.
8. Фоменко В. А., Савіцький Р. С. Використання Hugging Hub як сервісу бекенду для розгортання мовних моделей. *Development of Education, Science and Business: Results 2025: proceedings of the International scientific and practical Internet conference (Dnipro, 18–19 Dec. 2025).* Dnipro, 2026. 538 p.
9. Савіцький Р. С., Єфремов Ю. М. Огляд сучасного стану веб-доступності в Україні. *Міжнародна наукова інтернет-конференція (Тернопіль, Україна – Переворськ, Польща, 3–4 квіт. 2023 р.).* 2023. С. 39.
10. Савіцький Р. С. Використання random forest алгоритму для пошуку термінів в українській мові. *Практичні та теоретичні питання розвитку науки та освіти: матеріали XV Міжнар. наук.-практ. конф. (Львів, 29–30 серп. 2025 р.).* Львів: Львівський науковий форум, 2025. С. 109–110.
11. Савіцький Р. С., Бабицький Б. О. Аналіз архітектур нейронних мереж для розпізнавання облич. *Інформаційні технології і автоматизація – 2025: тези доп. XVIII Міжнар. наук.-практ. конф. (Одеса, 30–31 жовт. 2025 р.).* Одеса: ОНТУ, 2025. С. 1032.

Наукові праці, які додатково відображають наукові результати дисертації:

12. Карапет Л. В., Савіцький Р. С. Платформа для навчання з обмеженими можливостями. *Комп'ютерні технології: інновації, проблеми, рішення: тези доп. VI Всеукр. наук.-техн. конф. (Житомир, 29–30 листоп. 2023 р.).* Житомир: Житомирська політехніка, 2023. С. 213–214.
13. Волков О. М., Савіцький Р. С. Вплив Universal Sentence Encoder на розвиток обробки природної мови. *Інформаційні технології – 2025: тези*

доп. XII Всеукр. наук.-практ. конф. молодих учених (Київ, 15 трав. 2025 р.). Київ: Київ. столич. ун-т ім. Б. Грінченка, 2025. С. 18.

14. Фоменко В. А., Савіцький Р. С. Автоматизований пошук помилок в українському тексті з використанням моделей штучного інтелекту. *Комп'ютерні технології: інновації, проблеми, рішення: тези VIII Всеукр. наук.-практ. конф. (Житомир, 2–3 груд. 2025 р.).* Житомир: Житомирська політехніка, 2025. С. 197.
15. Chupryna V. A., Savitskyi R. S. The use of eye-tracking technologies in educational systems. *Актуальні питання розвитку інформаційних технологій: матеріали VII Міжнар. конф.* (Дніпро, 19 листопада 2025). Дніпро: Приазовський державний технічний університет, 2025. С. 152–154.

Зміст

Перелік умовних позначень 17

Вступ 19

РОЗДІЛ 1. Аналіз проблеми дослідження вебдоступності для людей з когнітивними та інтелектуальними порушеннями 26

1.1. Проблеми та особливості вебдоступності для людей з когнітивними та інтелектуальними порушеннями 26

1.2. Нормативно-правові засади та стандарти вебдоступності 38

1.3. Огляд сучасних досліджень у галузі вебдоступності для людей з когнітивними та інтелектуальними порушеннями 45

1.4. Наявні методи аналізу тексту для людей з когнітивними обмеженнями 53

Висновки до розділу 1 71

РОЗДІЛ 2. Методи забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями 73

2.1. Визначення індексу когнітивної доступності тексту для людей з когнітивними та інтелектуальними порушеннями 73

2.2. Метод визначення стабільної емоції користувача 98

2.3. Спеціалізовані словники синонімічних замін української мови ... 112

2.4. Узагальнений метод забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями 127

Висновки до розділу 2 138

РОЗДІЛ 3. Програмна реалізація забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями 140

3.1. Алгоритми та програмна реалізація забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями 140

3.2. Реалізація адаптивного інтерфейсу та допоміжних підсистем програмного комплексу 166

3.3. Перевірка ефективності програмної реалізації авторського методу 174

Висновки до розділу 3 189
Висновки 192
Список використаних джерел 195
Додатки 213
Додаток А 213
Додаток Б 217
Додаток В 220
Додаток Г 223
Додаток Д 227
Додаток Е 532

Перелік умовних позначень

Скорочення, термін, позначення	Пояснення
AASPIRE	Партнерство з досліджень і освіти у сфері аутизму серед дорослих (Academic Autism Spectrum Partnership in Research and Education)
ACL	Асоціація комп'ютерної лінгвістики (Association for Computational Linguistics)
AffectNet	Набір даних для моделювання та розпізнавання емоцій
ARI	Автоматизований індекс читабельності
ATAG	Настанови з доступності інструментів авторства
BERT	Двонапрямлена трансформерна модель кодування тексту
BiLSTM	Двонапрямлена мережа з довгою короткостроковою пам'яттю (Bidirectional LSTM)
BoW	Мішок слів (Bag-of-Words)
ІКД/CAI	Когнітивний індекс доступності (Cognitive Accessibility Index)
CNN	Згорткова нейронна мережа
COGA	Робоча група з когнітивної доступності W3C
CV	Комп'ютерний зір
Doc2Vec	Векторні представлення документів
DOM	Об'єктна модель документа
FastText	Швидкий текст (бібліотека векторних представлень)
FER	Розпізнавання виразу обличчя
FKGL	Індекс Флеша-Кінкейда (Flesch-Kincaid Grade Level)
FRES	Формула Флеша (Flesch Reading Ease Score)
GB	Градієнтний бустинг (Gradient Boosting)
GloVe	Глобальні вектори для представлення слів (Global Vectors for Word Representation)
GPT	Генеративна попередньо навчена трансформерна модель
GRU	Рекурентна мережева одиниця з вентилями
k-NN	Метод k-найближчих сусідів (k-Nearest Neighbors)
LightGBM	Полегшена реалізація градієнтного бустингу (Light Gradient Boosting Machine)
LLM	Велика мовна модель

Скорочення, термін, позначення	Пояснення
LR	Логістична регресія (Logistic Regression)
LSTM	Довга короткострокова пам’ять
ML	Машинне навчання
NB	Наївний баєсівський класифікатор
NDJSON	Newline Delimited JSON
NER	Розпізнавання іменованих сутностей
NLP	Обробка природної мови
NLTK	Інструментарій обробки природної мови
NLU	Розуміння природної мови
PLM	Попередньо навчена мовна модель (Pretrained Language Model)
POUR	Perceivable-Operable-Understandable-Robust
POS	Частина мови (Part of Speech)
RAF-DB	Набір даних реальних облич (Real-world Affective Faces Database)
ResNet	Залишкова нейронна мережа (Residual Network)
RF	Випадковий ліс (Random Forest)
RNN	Рекурентна нейронна мережа
RoBERTa	Оптимізована версія моделі BERT
SE	Стиснення та збудження (Squeeze-and-Excitation)
SMOG	Індекс читабельності на основі багатоскладових слів
SVM	Метод опорних векторів
TF-IDF	Частота терміна / обернена частота документа
TSAR	Семінар з автоматизованого спрощення тексту (Text Simplification, Accessibility and Readability)
TTS	Синтез мовлення з тексту
UAAG	Настанови з доступності агентів користувача
XGBoost	Екстремальний градієнтний бустинг (Extreme Gradient Boosting)
W3C	Консорціум Всесвітньої павутини
WAI	Ініціатива з вебдоступності W3C
WCAG	Настанови з доступності вебконтенту
Word2Vec	Векторні представлення слів (алгоритм)
СДУГ	Синдром дефіциту уваги та гіперактивності

Вступ

Актуальність дослідження. Когнітивні та інтелектуальні порушення охоплюють широкий спектр станів, що впливають на сприйняття, опрацювання й утримання інформації: від синдрому дефіциту уваги та гіперактивності і специфічних порушень навчання до інтелектуальної недостатності, синдрому Дауна, розладів аутистичного спектра й набутих когнітивних дефіцитів унаслідок черепно-мозкових травм або нейродегенеративних захворювань. За даними Всесвітньої організації охорони здоров'я, понад мільярд людей у світі мають ту чи іншу форму інвалідності, а когнітивні порушення входять до числа найпоширеніших. Для цієї категорії осіб складність текстового контенту є системним бар'єром, що обмежує можливості в освіті, зайнятості та повноцінній участі в суспільному житті.

Міжнародна нормативна база у сфері вебдоступності, стандарти WCAG 2.2 [138] і напрацювання робочої групи COGA W3C [136], закріплює загальні принципи доступності для людей з когнітивними порушеннями. Проте ці стандарти переважно формулюють вимоги на рівні рекомендацій, не надаючи конкретних обчислюваних метрик для автоматизованої оцінки когнітивної складності тексту. Класичні індекси читабельності, зокрема Флеша–Кінкейда [56], Ганнінга Фога [62] і SMOG [92], розроблені для англомовних текстів і спираються на поверхневі морфологічні ознаки, що недостатньо відображає реальні когнітивні бар'єри сприйняття. Для морфологічно складних мов, зокрема української, ці підходи дають ненадійні результати через відмінності у словотворчих і синтаксичних структурах.

Для України проблема набуває особливої гостроти в контексті гармонізації законодавства з вимогами Директиви ЄС 2016/2102 [54] про доступність вебсайтів і мобільних застосунків органів публічного сектора, а також у зв'язку з переходом освіти до дистанційних форм навчання. Більшість освітніх вебресурсів не має адаптованих версій для осіб з когнітивними порушеннями, а наявні інструменти аналізу вебдоступності орієнтовані на технічні аспекти,

зокрема контрастність, альтернативний текст для зображень і навігацію з клавіатури, та не охоплюють когнітивну складність текстового контенту як самостійний вимір доступності.

Наявні системи спрощення тексту переважно реалізують статичний підхід: текст адаптується до заздалегідь визначеного рівня складності незалежно від когнітивного стану конкретного читача в момент взаємодії. Такий підхід ігнорує індивідуальну варіабельність когнітивного навантаження, яка є особливо вираженою у людей з синдромом дефіциту уваги та гіперактивності (СДУГ), тривожними розладами і набутими когнітивними дефіцитами. Натомість поєднання автоматизованого аналізу тексту з неінвазивним моніторингом емоційного стану через вбудовану відеокамеру відкриває можливість реалізувати адаптивне спрощення, що орієнтується на реальний стан конкретного користувача, а не на усереднений профіль цільової аудиторії.

Розробка методів і алгоритмів автоматизованого аналізу та адаптивного спрощення україномовного вебтексту з урахуванням когнітивних потреб цільової аудиторії й емоційного стану користувача є актуальним завданням на перетині програмної інженерії, комп’ютерної лінгвістики і спеціальної педагогіки. Вирішення цього завдання сприятиме реалізації права на рівний доступ до інформації для людей з когнітивними та інтелектуальними порушеннями в умовах цифровізації освіти і суспільства загалом.

Зв'язок роботи з науковими програмами, планами, темами. Дисертаційне дослідження виконувалось відповідно до НДР кафедри інженерії програмного забезпечення Державного університету “Житомирська політехніка” “Аналіз вебдоступності в освітньому процесі” (0124U004618, 2024–2027 рр.), одним із виконавців якої є дисертант.

Мета дослідження – розробка методів та алгоритмів забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями.

Визначена мета досягається шляхом вирішення таких завдань:

1. Проаналізувати проблему вебдоступності для людей з когнітивними та інтелектуальними порушеннями, систематизувати наявні підходи до оцінювання та адаптації текстового контенту і визначити обмеження наявних рішень для україномовного середовища.

2. Розробити метод визначення індексу когнітивної доступності тексту (ІКД), що враховує лексичні, синтаксичні та структурні характеристики україномовних текстів.

3. Розробити метод визначення стабільної емоції розуміння тексту користувачем.

4. Розробити узагальнений метод забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями.

5. Розробити алгоритм та програмну реалізацію забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями, перевірити ефективність.

Об’єктом дослідження є процес забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями.

Предмет дослідження – є методи і алгоритми забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями.

Методи дослідження. Для вирішення визначених завдань використовувались такі методи: аналізу, узагальнення, систематизації для опрацювання наявних джерел; системного аналізу і синтезу для побудови узагальненої моделі когнітивної доступності тексту та архітектури програмного комплексу; обробки природної мови та лінгвістичного аналізу для визначення лексичних, синтаксичних і структурних компонентів індексу ІКД; машинного навчання для розпізнавання емоційного стану користувача у реальному часі; статистичні методи, зокрема конкордаційний аналіз Кендалла і кореляцію Спірмена для верифікації відповідності алгоритмічних оцінок людському сприйняттю складності тексту; об’єктно-орієнтованого проєктування та

тестування програмного забезпечення для реалізації і верифікації програмного комплексу.

Використання технологій штучного інтелекту.

У процесі виконання дисертаційного дослідження технології штучного інтелекту використовувались як допоміжний інструмент візуалізації для кращого представлення окремих розроблених схем за власним описом.

Формування наукових положень, постановка мети і завдань дослідження, проведення експериментів, обробка результатів та формулювання висновків виконані автором самостійно. Технології штучного інтелекту не використовувались для генерування тексту дисертації чи формулювання наукових результатів.

Наукова новизна та теоретичне значення отриманих результатів полягають у тому, що *вперше:*

- запропоновано індекс когнітивної доступності тексту як інтегральний нормований показник, що поєднує лексичний, синтаксичний і компонент читабельності в єдиній шкалі, адаптованій для української мови та цільової групи користувачів, на відміну від наявних метрик читабельності, які орієнтовані на англомовні тексти і не враховують специфічних когнітивних бар'єрів сприйняття;

- розроблено метод визначення стабільної емоції на основі розпізнавання виразу обличчя через відеокамеру пристрою в реальному часі, що дозволяє використовувати емоційний стан як індикатор когнітивного навантаження в задачах забезпечення спрощення тексту;

- розроблено узагальнений метод та алгоритм забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями, що поєднує статичний аналіз ІКД з динамічним управлінням на основі стабілізованої емоції та спрощенні тексту.

Практичне значення отриманих результатів полягає в тому, що:

– розроблено програмний комплекс забезпечення когнітивної доступності україномовного вебтексту із застосуванням трирівневої архітектури, що реалізує

повний цикл від аналізу когнітивної складності до ітераційного спрощення та може бути впроваджений як інструмент підтримки читання для осіб з когнітивними та інтелектуальними порушеннями;

– сформовано спеціалізовані словники синонімічних замін семи категорій проблемної лексики з детермінованою ієрархією підстановок та морфологічним узгодженням, що становлять лінгвістичний ресурс, придатний для повторного використання в інших системах обробки природної мови;

– розроблено модуль розпізнавання емоційного стану користувача з двоканальною стабілізацією та програмною калібрацією параметрів на базі телеметричного корпусу, що може використовуватися як самостійний компонент неінвазивного моніторингу когнітивного навантаження;

– проведено серію з восьми обчислювальних експериментів, результати яких підтвердили конвергенцію алгоритму за 1–3 ітерації, збереження семантики та масштабованість методу.

Матеріали дослідження можуть бути використані освітніми, соціальними та адміністративними установами для забезпечення когнітивної доступності текстового вебконтенту для людей з когнітивними та інтелектуальними порушеннями.

Впровадження результатів дослідження в практику підтверджується довідками ТОВ «ТІЄТОЕВРІ КРІЕЙТ УКРАЇНА» та "ВМК ГРУП ЮА" (довідки додаються).

Особистий внесок здобувача. Усі основні результати дисертаційного дослідження, які представлені до захисту, одержані автором особисто (звіт подібності сформований за допомогою програми StrikePlagiarism додається).

У працях, опублікованих у співавторстві, автору належать: експеримент та опис бібліотек та інструментів NLP для навчання моделей та роботи з ними [9]; розробка, опис та апробація системи [6]; аналіз та опис платформи для навчання осіб з обмеженими можливостями [5]; прототип системи розпізнавання облич [1]; розбір енкодера в задачах NLP [2]; розгортання, створення моделі та її публікація на Hugging Face [15]; створення моделі пошуку помилок в

українському тексті [16]; огляд стану вебдоступності в Україні та аналіз вебсайтів університетів України [12]; аналіз та порівняння архітектур, заснованих на DeepFace [14]; аналіз використання трекінгу очей у навчальних системах в Україні [37].

Апробація результатів дисертації. Основні результати наукових досліджень неодноразово доповідалися на ряді міжнародних та Всеукраїнських наукових конференціях та семінарах, зокрема: Міжнародна наукова інтернет-конференція (Тернопіль, Україна – Переворськ, Польща, 2023); VI Всеукраїнська науково-технічна конференція «Комп'ютерні технології: інновації, проблеми, рішення» (Житомир, 2023); XIV Міжнародна науково-технічна конференція «Інформаційно-комп'ютерні технології – 2024» (Житомир, 2024); Всеукраїнська науково-практична інтернет-конференція «Автоматизація та комп'ютерно-інтегровані технології у виробництві та освіті: стан, досягнення, перспективи розвитку» (Черкаси, 2024); XVIII Всеукраїнська науково-практична WEB-конференція аспірантів, студентів та молодих вчених «Комп'ютерні інтелектуальні системи та мережі» (Кривий Ріг, 2025); XII Всеукраїнська науково-практична конференція молодих учених «Інформаційні технології – 2025» (Київ, 2025); XV Міжнародна науково-практична конференція «Практичні та теоретичні питання розвитку науки та освіти» (Львів, 2025); XVIII Міжнародна науково-практична конференція «Інформаційні технології і автоматизація – 2025» (Одеса, 2025); VII Міжнародна конференція «Актуальні питання розвитку інформаційних технологій» (Дніпро, 2025); VIII Всеукраїнська науково-практична конференція «Комп'ютерні технології: інновації, проблеми, рішення» (Житомир, 2025); Міжнародна науково-практична інтернет-конференція «Development of Education, Science and Business: Results 2025» (Дніпро, 2025).

Публікації. Результати дослідження викладено в 15 опублікованих працях, із них – 4 статті у наукових фахових виданнях України і 11 тез в інших наукових виданнях.

Структура та обсяг дисертації. Дисертаційна робота складається з анотації, вступу, трьох розділів, висновків до кожного розділу, висновків, списку використаних джерел (151 найменування, з них 135 іноземною мовою), 6 додатків. Загальний обсяг дисертації – 535 сторінок, з них 176 сторінок основного тексту, у якому міститься 36 таблиць та 26 рисунків.

РОЗДІЛ 1. Аналіз проблеми дослідження вебдоступності для людей з когнітивними та інтелектуальними порушеннями

1.1. Проблеми та особливості вебдоступності для людей з когнітивними та інтелектуальними порушеннями

У межах сучасного цифрового середовища забезпечення вебдоступності постає як обов'язкова передумова ефективної та повноцінної взаємодії користувача з вебресурсами й цифровими сервісами, оскільки саме вона визначає можливість їх використання незалежно від індивідуальних особливостей людини чи умов доступу. Значення вебдоступності виходить за межі технічної сумісності з допоміжними засобами, оскільки йдеться про можливість сприймати, розуміти, використовувати й контролювати цифровий контент без системних бар'єрів. За даними Всесвітньої організації охорони здоров'я, приблизно 1,3 млрд людей, або близько 16 % населення світу, живуть з інвалідністю [69, 142], тому проблема цифрової доступності має масовий, а не локальний характер.

Незважаючи на розвиток нормативних підходів і практичних інструментів забезпечення доступності, фактичний стан вебсередовища не відповідає задекларованим вимогам. Це засвідчують результати масштабного дослідження WebAIM Million, відповідно до яких більшість проаналізованих домашніх сторінок містить порушення, що суперечать базовим критеріям вебдоступності [128]. Водночас більш суттєвим є те, що навіть формальна відповідність окремим критеріям доступності не забезпечує реальної зрозумілості ресурсу для користувачів із когнітивними та інтелектуальними порушеннями. За таких умов вебдоступність доцільно трактувати не як суто технічну характеристику системи, а як комплексну властивість, що виникає у процесі взаємодії людини з контентом і цифровим інтерфейсом.

У дослідженні прийнято визначення ініціативи W3C Web Accessibility Initiative (WAI). Вебдоступність – це властивість вебсайтів, інструментів та технологій з можливістю сприйняття, розуміння, навігації та взаємодії для людей з інвалідністю [71]. Визначення фіксує чотири базові виміри доступності

(сприйняття, розуміння, навігацію та взаємодію) та слугує відправною точкою для подальшого виокремлення когнітивного аспекту.

У науковій літературі вебдоступність визначається як багатокомпонентне поняття. У роботі H. Petrie, A. Savva та C. Power підкреслено, що вона не зводиться до одного універсального визначення, оскільки включає технічний, функціональний, користувацький і соціальний виміри [72, 105]. Такий підхід є особливо важливим для дослідження когнітивної доступності, оскільки труднощі цієї групи користувачів виникають не лише через недоступність інтерфейсних елементів, а й через сам спосіб подання інформації. Якщо для частини користувачів критичним є, наприклад, альтернативний текст до зображення або можливість навігації з клавіатури, то для людей з когнітивними та інтелектуальними порушеннями ключовим бар'єром часто стає надмірна складність тексту, заплутана послідовність дій, велика кількість взаємопов'язаних інструкцій або високе навантаження на увагу і пам'ять.

У межах такого підходу доцільним є розмежування понять загальної та когнітивної вебдоступності. Перше охоплює сукупність вимог до структури, подання й технічної реалізації цифрового контенту та інтерфейсів. Друге ж зосереджується на характеристиках сприйняття й використання ресурсу, зокрема на тому, наскільки він є зрозумілим, передбачуваним і придатним до керування для користувачів, які відчувають труднощі з читанням, концентрацією уваги, робочою пам'яттю, інтерпретацією інструкцій або послідовним виконанням дій [59]. Відповідно, когнітивна доступність пов'язана не лише з формальною відповідністю стандарту [138], а з фактичною можливістю користувача побудувати правильну ментальну модель інтерфейсу й контенту.

Важливо також відмежувати когнітивну доступність від простої читабельності тексту. Читабельність зазвичай описує ступінь формальної складності тексту, наприклад довжину речень, довжину слів або структурну простоту фраз. Натомість когнітивна доступність є ширшою характеристикою. Вона включає лексичну зрозумілість, синтаксичну прозорість, логічну послідовність викладу, передбачуваність структури, однозначність інструкцій,

контрольоване інформаційне навантаження і зменшення кількості елементів, що можуть відволікати або перевантажувати користувача [59]. Саме тому текст, який формально є читабельним, не обов’язково є когнітивно доступним.

Когнітивна вебдоступність (англ. cognitive web accessibility) – це властивість вебконтенту, вебзастосунків та цифрових інтерфейсів із забезпеченням сприйняття, розуміння, навігації та взаємодії для людей з когнітивними та інтелектуальними порушеннями [39, 97]. Адаптація подання інформації, структури, мови та механізмів взаємодії здійснюється з урахуванням особливостей когнітивних процесів – сприйняття, пам'яті, уваги, розуміння мови та розв'язання задач [58].

Цю позицію підтримують і сучасні рекомендації W3C, зокрема документ Making Content Usable for People with Cognitive and Learning Disabilities [136], у якому підкреслюється необхідність спеціальних підходів до проєктування контенту для користувачів із когнітивними та навчальними труднощами. Рекомендації роблять акцент на ясності мови, передбачуваності структури, спрощенні інструкцій, зменшенні когнітивного перевантаження та підтримці покрокової взаємодії. Це означає, що когнітивна доступність не може бути досягнута лише за рахунок технічної валідності HTML-структури чи автоматизованого проходження частини WCAG-критеріїв. Вона потребує окремого аналітичного й методичного опрацювання.

Таким чином, вебдоступність для осіб із когнітивними та інтелектуальними порушеннями доцільно інтерпретувати як окрему науково-практичну проблему, у межах якої перетинаються аспекти вебдоступності, мовної зрозумілості, регуляції когнітивного навантаження, чіткості навігаційних структур і безпечності взаємодії з контентом. Для цієї групи користувачів саме текстовий компонент і способи його організації нерідко виступають визначальним джерелом бар’єрів у процесі користування ресурсом [121, 140].

Когнітивні та інтелектуальні порушення у контексті вебвзаємодії доцільно розглядати не як суто медичну характеристику користувача, а як сукупність функціональних обмежень, що впливають на сприйняття, розуміння,

запам’ятовування і послідовне виконання цифрових дій. Саме такий підхід використовується в дослідженнях когнітивної доступності, де увага зосереджується на тому, які саме труднощі виникають у користувача під час роботи з вебконтентом, а не лише на клінічній назві стану [59]. З позиції цифрової взаємодії найбільш значущими є труднощі, пов’язані з увагою, робочою пам’яттю, виконавчими функціями, обробкою мовної інформації, швидкістю орієнтації в інтерфейсі та побудовою ментальної моделі послідовності дій [125].

На відміну від сенсорних порушень, для яких веббар’єри часто мають відносно чітке технічне вираження, когнітивні бар’єри зазвичай виникають унаслідок поєднання кількох чинників. До них належать складна лексика, багаторівневі інструкції, перевантажені сторінки, нестабільна навігація, надмірна кількість варіантів вибору, неоднозначні формулювання та потреба одночасно утримувати кілька елементів контексту [59]. Такі бар’єри є особливо критичними у вебсередовищі, оскільки цифровий інтерфейс вимагає від користувача не лише прочитати текст, а й інтерпретувати його, співвіднести з інтерфейсними елементами, обрати правильну дію та проаналізувати його.

Сучасні рекомендації щодо когнітивної доступності виходять із того, що значна частина труднощів користувачів пов’язана з базовими когнітивними навичками: увагою, пам’яттю, обробкою мовлення й тексту, розумінням вибору, часовим менеджментом та здатністю підтримувати послідовність дій [136]. Саме тому для людей з когнітивними та інтелектуальними порушеннями критичними стають не лише окремі слова чи речення, а й загальна структура сторінки, спосіб подачі інструкцій, передбачуваність навігації, обсяг фрагмента, який потрібно осмислити одночасно, а також кількість відволікальних елементів. У прикладному вимірі це означає, що одна й та сама сторінка може бути технічно доступною, але залишатися фактично важкою для розуміння і використання.

Особливої уваги потребує зв’язок між когнітивним навантаженням і текстовим контентом. Текст у вебсередовищі виконує не лише інформативну функцію, а й функцію навігації, інструктажу, підтвердження дії, пояснення

помилки та підтримки прийняття рішень. Через це лексична складність, синтаксична щільність, абревіатури, терміни, неявні припущення та великі за обсягом фрагменти тексту можуть перетворюватися на системний бар’єр навіть тоді, коли візуальна частина інтерфейсу побудована коректно [93]. У задачі забезпечення когнітивної доступності це робить текстовий контент центральним об’єктом аналізу.

Таким чином, для людей з когнітивними та інтелектуальними порушеннями ключовими є не лише бар’єри доступу до інтерфейсу як такого, а й бар’єри когнітивного опрацювання змісту. Це означає, що вебдоступність для цієї групи користувачів має оцінюватися через здатність ресурсу підтримувати розуміння, передбачуваність, послідовність дій і зниження надлишкового когнітивного навантаження. Саме в такій постановці аналіз окремих груп користувачів, зокрема людей з дислексією, СДУГ, аутизмом та інтелектуальними порушеннями, стає методично обґрунтованим [139].

Поряд із суб'єктивними характеристиками контенту, когнітивне навантаження може бути оцінено через об'єктивні ознаки стану користувача. Зокрема, дослідження у сфері обчислення емоцій підтверджують зв'язок між рівнем когнітивного напруження та мімікою людини: вираз обличчя є одним із надійних непрямих індикаторів когнітивного навантаження, доступних для автоматизованого зчитування через відеокамеру [107]. Це відкриває можливість формування зворотного зв'язку в реальному часі: система може виявляти ознаки ускладненого сприйняття тексту ще до того, як користувач свідомо звернеться за допомогою, і адаптивно коригувати рівень спрощення.

Однією з найбільш досліджених груп у контексті когнітивної доступності тексту є люди з дислексією. Для цієї групи характерними є труднощі з декодуванням письмового мовлення, стабільністю читання, швидкістю розпізнавання слів, утриманням фонологічних і візуальних відповідностей, а також збереженням темпу розуміння під час роботи з великими фрагментами тексту [91, 111]. У цифровому середовищі ці труднощі проявляються не ізольовано, а в поєднанні з інтерфейсними чинниками, через що вебконтент,

який формально є доступним, може залишатися важким для реального використання.

Огляд наукових праць з вебдоступності для людей з дислексією показує, що найбільший вплив на якість взаємодії мають організація контенту, типографіка, контрастність, можливість персоналізації відображення тексту та підтримка зосередженого читання [27, 52]. Це означає, що для користувачів з дислексією бар'єр виникає не лише на рівні окремого слова, а й на рівні всієї конфігурації текстового представлення: щільності сторінки, кількості візуальних відволікань, довжини абзаців, структури заголовків і візуального ритму читання.

Важливу роль у поясненні цих труднощів відіграють дослідження візуального перевантаження і позиційного кодування літер. Роботи в цій галузі показують, що для частини читачів з дислексією критичними є інтерференція сусідніх літер, підвищена чутливість до щільного розміщення символів та труднощі з точним розрізненням позиції літер у межах слова [41, 141]. Саме тому навіть незначні зміни в інтервалах, щільності набору або просторовій організації тексту можуть відчутно впливати на якість читання.

Цей висновок підтверджують і дослідження типографічних параметрів. Зокрема, збільшення міжлітерного інтервалу в окремих випадках істотно покращує швидкість і точність читання у людей з дислексією, що безпосередньо пов'язують із зменшенням ефекту crowding [143, 151]. Для вебконтенту це має прикладне значення: труднощі користувача залежать не лише від змісту тексту, а й від його візуального втілення на екрані. Відповідно, когнітивна доступність для цієї групи користувачів не може оцінюватися лише через мовну складність без урахування способу подання тексту.

Окремо слід підкреслити, що для користувачів з дислексією критичними є не тільки типографічні параметри, а й лінгвістична структура контенту. Довгі речення, рідковживана лексика, перевантажені термінами інструкції, абревіатури, неоднозначні формулювання і великі блоки суцільного тексту підвищують бар'єр сприйняття навіть тоді, коли інтерфейс сторінки загалом є

технічно коректним [47, 52]. Саме це робить дислексію однією з ключових груп для обґрунтування методів автоматизованого аналізу і спрощення тексту.

Таким чином, у випадку дислексії вебдоступність значною мірою визначається тим, наскільки текст є структурованим, візуально керованим, лексично зрозумілим і синтаксично прозорим. Це дозволяє розглядати людей з дислексією як одну з основних цільових груп при розробленні методів забезпечення когнітивної доступності текстового контенту [66, 73].

Іншою важливою цільовою групою є люди із синдромом дефіциту уваги і гіперактивності. У контексті взаємодії ключовими для них є труднощі підтримання уваги, контролю відволікання, моніторингу послідовності дій, утримання контексту та виконання завдань, що потребують багатоетапного опрацювання інформації [150]. Такі особливості безпосередньо впливають на сприйняття вебконтенту, оскільки цифрове середовище зазвичай передбачає постійну зміну фокусу, а також необхідність одночасно обробляти текст.

Дослідження академічного і соціального функціонування людей із СДУГ показують, що для них труднощі не обмежуються лише поведінковою імпульсивністю. Вони також проявляються у підтриманні уваги на завданні, читанні, письмовому опрацюванні інформації та тривалому виконанні структурованих дій [50, 51]. Тому надлишкова щільність інформації та складні текстові блоки здатні істотно знижувати ефективність взаємодії.

Окремі емпіричні дослідження підтверджують, що у підлітків і дорослих із СДУГ можуть спостерігатися нижчі показники точності, швидкості та розуміння під час читання текстів, а також істотний зв’язок між якістю читання й виконавчими функціями [38, 96]. Вище перелічені показники принципово важливі для задачі когнітивної доступності, оскільки демонструють, що бар’єр для користувача формується не лише на рівні мови як такої, а й на рівні здатності підтримувати послідовне смислове опрацювання тексту.

Труднощі у людей із СДУГ виникають не лише з утриманням уваги, а й з кодуванням, відтворенням та відсіюванням сторонньої інформації під час роботи з вербальним матеріалом [20, 122]. У вебсередовищі це проявляється, наприклад,

коли користувач стикається з багатокроковим сценарієм і змушений одночасно утримувати в пам'яті попередні дії та розбиратися з прихованими залежностями між елементами інтерфейсу.

Короткі завершені фрагменти тексту, чітка ієрархія заголовків і прозора навігація допомагають людям із СДУГ зосереджуватися на конкретному завданні без зайвого відволікання [25, 78]. Саме тому в межах дослідження, особи із СДУГ розглядаються як одна з груп користувачів, для яких автоматизоване виявлення та локальне спрощення текстових бар'єрів може мати практичну цінність. Вебдоступність для людей із СДУГ визначається тим, наскільки текстове середовище ресурсу знижує зовнішнє когнітивне навантаження і дозволяє послідовно опрацьовувати матеріал без втрати контексту [80, 82].

До окремої групи, що потребує спеціального врахування в задачі вебдоступності, належать користувачі з розладами аутистичного спектра. Для них критичними можуть бути сенсорне перевантаження, труднощі з обробкою неоднозначних інструкцій, висока чутливість до нестабільної структури сторінки, необхідність додаткового прагматичного контексту та потреба у чітких, передбачуваних сценаріях взаємодії [110]. У цифровому середовищі це проявляється в підвищеній вимозі до структурної стабільності, прямоти формулювань, повторюваності інтерфейсної логіки та відсутності декоративного шуму. Важливим орієнтиром є Web Accessibility Guidelines for Autistic Web Users, розроблені AASPIRE за участю самих аутичних користувачів і перевірені емпірично [110]. Рекомендації охоплюють простоту інтерфейсу, однозначне маркування функцій, явне відображення навігаційних елементів і пояснення нестандартних дій. Цей підхід підкреслює, що когнітивна вебдоступність не зводиться до технічної коректності, а передбачає також прогнозованість і інтелектуальну прийнятність інтерфейсу для користувача.

Для людей з інтелектуальними порушеннями перешкоди у взаємодії з вебсередовищем виникають на рівні розуміння контенту, інтерпретації інструкцій, співвідношенні тексту з дією та підтриманні послідовності

взаємодії [29, 45]. Значення має не лише мова інтерфейсу, а й те, наскільки інформація подана в простому, покроковому, семантично прозорому та візуально стабільному форматі.

Емпіричні дослідження доступних вебсайтів для користувачів з інтелектуальними порушеннями показують, що адаптація контенту відповідно до принципів «easy-to-read» істотно покращує ефективність використання ресурсу і не створює негативного ефекту для користувачів без таких порушень [70, 77]. Це має принципове значення для дослідження, оскільки вказує на можливість формувати доступніші тексти без зниження загальної придатності ресурсу для ширшої аудиторії.

Ширший цифровий контекст доступу до інтернету для людей з інтелектуальними порушеннями розкриває бар'єри, що виходять за межі інфраструктури і охоплюють мовну складність, залежність від сторонньої підтримки, непрозорі сценарії взаємодії та низьку адаптованість цифрового контенту до реальних когнітивних можливостей користувача [34, 89]. Проблема полягає не лише у доступі до інтернету як такого, а у можливості самостійно та безпомилково користуватися інформаційним середовищем.

Таким чином, у випадку аутизму та інтелектуальних порушень ключового значення набувають передбачуваність, простота, структурна стабільність, уникнення неоднозначності та орієнтація на покрокове розуміння контенту. Це ще раз підтверджує, що когнітивна доступність повинна аналізуватися як окремий напрямок вебдоступності, безпосередньо пов’язаний із якістю тексту, логікою інструкцій і способом організації взаємодії.

Текстовий контент є центральним носієм когнітивних бар'єрів у вебсередовищі, оскільки він одночасно передає основну інформацію, задає навігаційну логіку, пояснює сценарії взаємодії, формулює повідомлення про стан системи, описує помилки та підтримує прийняття рішень користувачем. Лексична складність, синтаксична щільність, непрозорість інструкцій або надмірний обсяг тексту впливають не лише на читання як таке, а й на здатність людини орієнтуватися в інтерфейсі та безпомилково виконувати дії.

Попри відмінності між дислексією, СДУГ, розладами аутистичного спектра та інтелектуальними порушеннями, у задачі вебдоступності для цих груп виявляється спільне ядро проблем, а саме підвищена чутливість до мовної складності, труднощі утримання контексту, потреба в передбачуваній структурі, зростання помилок при багатокрокових сценаріях та негативний вплив перевантаження інформацією. Саме ця міжгрупова спільність і дає підстави не обмежуватися набором приватних рекомендацій для окремих станів, а розглядати можливість побудови узагальненого методу забезпечення когнітивної доступності вебконтенту [81].

Аналіз типових труднощів (див. табл. 1.1) показує, що хоча конкретні прояви труднощів відрізняються між окремими групами користувачів, більшість релевантних бар'єрів концентрується навколо кількох спільних властивостей вебконтенту, а саме надмірної мовної складності, синтаксичної щільності, непрозорої структури, нестабільної навігації, багатокрокових сценаріїв взаємодії та перевантаження інформацією. Це дає підстави розглядати когнітивну доступність не як набір ізольованих приватних випадків, а як окремий аналітичний вимір вебдоступності [88].

Таблиця 1.1

Типові труднощі користувачів з когнітивними та інтелектуальними порушеннями та відповідні бар’єри вебдоступності

Група користувачів	**Типові когнітивні труднощі**	**Прояв у роботі з вебконтентом**	**Релевантні бар’єри вебдоступності**
Люди з дислексією	Декодування письмового тексту, позиційне кодування літер, зниження темпу читання	Складність читання довгих і щільних текстів, підвищена чутливість до типографіки	Рідковживана лексика, довгі речення, суцільні текстові блоки, щільний набір, абревіатури
Люди із СДУГ	Нестійка увага, труднощі підтримки контексту	Швидка втрата фокусу, труднощі з довгими інструкціями і багатокроковими сценаріями	Перевантажені сторінки, надлишковий текст, нестабільна структура, велика кількість одночасних опцій

продовж. табл. 1.1

Люди з РАС	Потреба у передбачуваності, труднощі з неоднозначністю, сенсорне перевантаження	Чутливість до нестабільної навігації, неявних правил, шумного інтерфейсу	Неочевидні інструкції, непослідовний інтерфейс, декоративний шум, неоднозначні формулювання
Люди з інтелектуальними порушеннями	Труднощі базового розуміння, послідовного виконання дій, інтерпретації інструкцій	Потреба у простому, покроковому, семантично прозорому контенті	Складні інструкції, термінологічне перевантаження, багатокрокові сценарії, непрозорі дії

Під когнітивними бар'єрами вебдоступностіварто розуміти перешкоди у взаємодії з вебресурсами (сайтами, додатками, онлайн-сервісами), які виникають у людей із когнітивними порушеннями через невідповідність дизайну, структури чи контенту їхнім особливостям обробки інформації. Також когнітивними бар’єрами вебдоступності є чинники та процеси дизайну вебконтенту, які перешкоджають доступу до онлайн-інформації та сервісів людям із когнітивними порушеннями [59].

До типових бар'єрів вебдоступності для людей з когнітивними та інтелектуальними порушеннями належать лексичні, синтаксичні, структурні, навігаційні та інструктивні бар'єри (рис. 1.1).

Лексичні бар'єри пов'язані з використанням рідковживаної лексики, термінів без пояснення, абревіатур, іншомовних вставок та семантично перевантажених конструкцій.Синтаксичні бар'єри виникають у разі довгих речень, складнопідрядних структур, великої кількості вставних елементів і неочевидних зв'язків між частинами висловлювання. Структурні бар'єри охоплюють надмірну щільність сторінки, слабку візуальну ієрархію, перевантажені блоки та відсутність чітких меж між фрагментами інформації [71].

Не менш важливими є навігаційні та інструктивні бар’єри. Для користувачів із когнітивними та інтелектуальними порушеннями складність

часто створює не лише сам текст, а й спосіб, у який він вбудований у сценарій взаємодії.

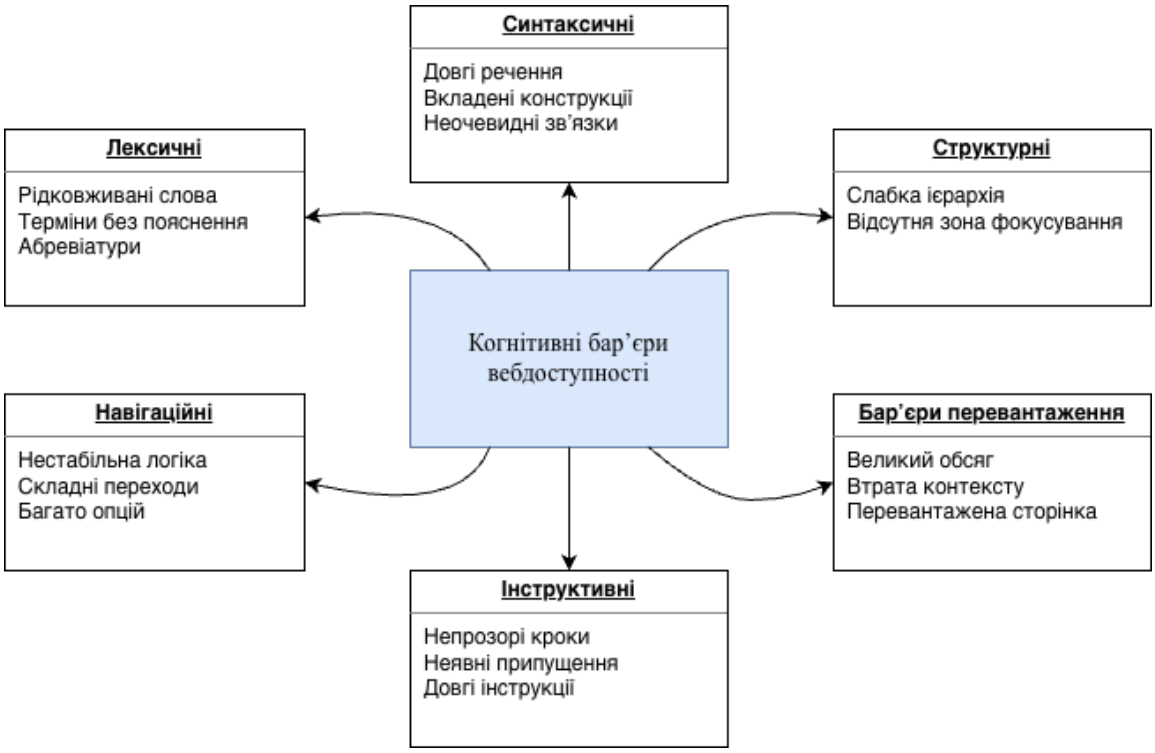


Рис. 1.1. Основні когнітивні бар'єри вебдоступності.

Якщо користувачеві потрібно одночасно читати довгу інструкцію, пам'ятати попередній крок, зіставляти текст із елементами інтерфейсу та приймати рішення між кількома альтернативами, то когнітивне навантаження швидко зростає [136].

Наслідком недостатньої когнітивної доступності є не лише ускладнення роботи з вебресурсом, а й системне зниження якості взаємодії. Для користувача це може проявлятися у збільшенні часу на розуміння контенту, зростанні кількості помилок під час виконання дій, втраті контексту, відмові від завершення сценарію, залежності від сторонньої допомоги та зниженні автономності під час роботи з цифровим середовищем [34, 100]. У ширшому сенсі це означає, що недостатня когнітивна доступність обмежує не лише комфорт користування, а й реальну можливість повноцінної участі людини в цифровому житті.

Отже, люди з когнітивними та інтелектуальними порушеннями стикаються у вебсередовищі зі специфічними бар'єрами, значна частина яких пов'язана саме

з текстовим контентом, логікою інструкцій і структурою взаємодії. Ці бар’єри лише частково покриваються загальними підходами до вебдоступності, що обґрунтовує потребу в окремому аналізі нормативних засад, сучасних досліджень і методів, спрямованих на виявлення та зниження когнітивної складності вебконтенту [33, 133].

Наведені бар'єри дають підстави розглядати когнітивну доступність тексту як активну властивість, що потребує не лише виявлення, а й цілеспрямованого формування. Фіксація невідповідностей у лексиці, синтаксисі чи структурі тексту становить лише діагностичний рівень роботи. Для забезпечення реальної доступності необхідні механізми оцінювання складності та її послідовного зниження до рівня, прийнятного для конкретної цільової групи.

Проблему ускладнює неоднакове сприйняття складності тексту різними читачами на когнітивному рівні, тому одні й ті самі лексичні та синтаксичні особливості можуть бути цілком прийнятними для одного читача та надмірно складними для іншого, а стан одного й того самого користувача змінюється залежно від рівня втоми, мотивації чи знайомства з темою. Статичне спрощення тексту до фіксованого рівня складності не враховує цієї варіабельності [129].

З огляду на це, забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями передбачає поєднання формального вимірювання складності з адаптивним механізмом корекції, здатним враховувати поточний стан користувача.

1.2. Нормативно-правові засади та стандарти вебдоступності

Нормативно-правові засади вебдоступності формуються не одним документом, а системою взаємопов’язаних міжнародних стандартів, спеціалізованих рекомендацій і правових актів, які визначають вимоги до цифрового контенту, інтерфейсів і способів взаємодії користувача з вебресурсом [137, 138]. Нормативно-правові засади вебдоступності важливі тим, що дають змогу відокремити вже формалізовані вимоги від тих аспектів

когнітивної доступності, які залишаються лише частково описаними або зовсім не охопленими чинними стандартами.

Центральне місце нормативно-правової бази вебдоступності посідають стандарти консорціуму W3C, насамперед Web Content Accessibility Guidelines (WCAG) [138], Authoring Tool Accessibility Guidelines (ATAG) [134] і User Agent Accessibility Guidelines (UAAG) [137]. Разом вони утворюють базову модель вимог до цифрової доступності, розподіляючи відповідальність між різними компонентами цифрового середовища, зокрема вебконтентом, інструментами та програмними засобами, через які користувач взаємодіє з контентом.

WCAG є основним нормативним стандартом оцінювання доступності вебконтенту. Він визначає критерії, що стосуються сприймання, керованості, зрозумілості та надійності цифрового ресурсу (принципи POUR), які стали міжнародною основою проєктування доступних вебсайтів і вебзастосунків [138]. У межах дослідження суттєвим є те, що WCAG охоплює текстові альтернативи, зрозумілість інструкцій, передбачуваність поведінки інтерфейсу, коректне опрацювання помилок і логіку навігації, а отже безпосередньо торкається властивостей, які впливають на когнітивну доступність текстового контенту.

ATAG і UAAG доповнюють WCAG і показують, що доступність не може розглядатися лише як властивість готової сторінки. ATAG орієнтований на інструменти створення контенту й фіксує вимоги до того, щоб авторські середовища допомагали створювати доступний вміст і самі були доступними для користувачів з інвалідністю [134]. UAAG задає вимоги до браузерів та інших користувацьких агентів, через які відвідувач отримує доступ до вебконтенту, що є важливим для повноцінного використання допоміжних механізмів і налаштувань представлення інформації [137].

Такий розподіл ролей узгоджується з підходом W3C до основних компонентів вебдоступності, згідно з яким реальна доступність виникає лише за умови узгодженої роботи контенту, браузерів і медіапрогравачів, допоміжних технологій, інструментів авторингу, а також користувацьких знань і практик [135]. Для когнітивної доступності це означає, що проблема не

обмежується формою подання тексту на сторінці. Вона залежить також від того, у якому середовищі цей текст створюється, як він доставляється користувачеві та наскільки інтерфейс і допоміжні механізми підтримують розуміння, послідовність дій і контроль когнітивного навантаження.

Отже, міжнародні стандарти вебдоступності створюють необхідну нормативну основу для аналізу цифрової доступності, однак уже на цьому рівні видно, що когнітивна доступність посідає в цій системі особливе місце. Вона лише частково підтримується загальними принципами зрозумілості та передбачуваності і водночас потребує окремого розгляду, оскільки мовна складність, когнітивне перевантаження і проблеми текстового сприйняття не завжди отримують достатньо формалізоване відображення у базових стандартах. Саме тому доцільно окремо проаналізувати потенціал і обмеження WCAG щодо когнітивної доступності [63, 147].

Попри фундаментальне значення WCAG для цифрової доступності, цей стандарт не можна вважати повністю достатнім для опису й оцінювання когнітивної доступності вебконтенту. Його сильна сторона полягає у фіксації базових вимог до зрозумілості, передбачуваності, навігації, обробки помилок і альтернативних форм подання інформації [138]. Критерії створюють мінімально необхідну основу для доступної взаємодії, однак не охоплюють повною мірою складність мовного сприйняття, рівень когнітивного навантаження, потребу в адаптивному спрощенні тексту та відмінності між користувачами з різними когнітивними профілями.

На практиці це означає, що вебресурс може відповідати більшості критеріїв WCAG, але залишатися складним для використання людьми з когнітивними та інтелектуальними порушеннями. Наприклад, навіть правильно структурована сторінка з технічно коректною навігацією не гарантує, що інструкції будуть достатньо простими, терміни зрозумілими, а логіка переходів прозорою для користувача, який має труднощі з увагою, робочою пам’яттю або сприйняттям складного тексту [59].

У межах мовно-текстового аспекту обмеження проявляються найвиразніше. Хоча WCAG містить окремі критерії, що частково враховують когнітивну доступність, у ньому відсутня чітка формалізація рівня лексичної складності, синтаксичної структури, зрозумілості термінів і контролю інформаційного навантаження, а також механізмів цілеспрямованого спрощення тексту. У контексті дослідження така прогалина має визначальне значення, оскільки саме текстовий контент часто формує основний бар’єр для сприйняття і потребує окремих підходів до аналізу та алгоритмічного оцінювання.

Серед критеріїв WCAG 2.2 найбільш безпосередньо пов'язаним із текстовою складністю є критерій 3.1.5 Читацький рівень (рівень AAA), який рекомендує надавати спрощену версію тексту, якщо його рівень складності перевищує рівень базової середньої освіти. Водночас цей критерій є необов'язковим і не входить до базових вимог відповідності, не визначає уніфікованого способу вимірювання складності тексту і не адаптований до специфіки окремих мов, зокрема української, а також не пропонує механізму локалізації конкретних бар'єрів у тексті чи алгоритму їх усунення. Наведені обмеження критерію 3.1.5 підтверджують необхідність окремого формалізованого показника когнітивної складності, яким у цьому дослідженні є ІКД.

Водночас користувачі з дислексією, СДУГ, аутизмом або інтелектуальними порушеннями мають відмінні профілі труднощів, які не завжди точно відображаються в межах єдиного набору вимог, що зумовлює потребу у більш специфічних підходах [136].

Одним із таких документів є рекомендації W3C Making Content Usable for People with Cognitive and Learning Disabilities, відомі як COGA [136]. Порівняно з базовою логікою WCAG, COGA рекомендація приділяє більше уваги зрозумілості мови, послідовності взаємодії та зниженню когнітивного навантаження, а також підтримці уваги користувача і передбачуваності дій. У центрі розгляду опиняється не формальне дотримання критеріїв, а здатність

користувача коректно сприйняти зміст, ухвалити рішення і завершити необхідну дію без зайвого напруження.

У контексті цього дослідження значення COGA полягає в тому, що він підтверджує окремий статус когнітивної доступності в системі вебстандартів, водночас показуючи межу їхньої формалізації. Рекомендації COGA є цінними як концептуальна й практична основа, проте не задають завершеного алгоритмічного інструментарію для автоматизованого аналізу тексту і не пропонують уніфікованого механізму зниження когнітивної складності вебконтенту. Ця прогалина і створює підстави для подальшого дослідження спеціальних методів виявлення й спрощення текстових бар'єрів.

Окрім системи стандартів W3C, нормативна рамка вебдоступності включає правові акти міжнародного та національного рівнів, які роблять вимоги доступності обов’язковими для окремих типів ресурсів, сервісів і організацій. У такому контексті відбувається перехід від рекомендаційної моделі до регуляторного зобов’язання, у межах якого доступність розглядається як необхідна характеристика цифрових продуктів і послуг, а не лише як бажана практика [54].

У межах права Європейського Союзу ключове значення має Directive (EU) 2016/2102, яка визначає вимоги до доступності вебсайтів і мобільних застосунків суб’єктів публічного сектору [54]. Документ встановлює обов’язковість забезпечення доступності цифрових ресурсів державних установ і водночас пов’язує правозастосування з технічними стандартами. У контексті дослідження це відображає інституційне визнання цифрової доступності як системної характеристики публічних вебресурсів.

Подальше розширення нормативної рамки відбулося через директиву European Accessibility Act (Directive EU 2019/882), яка поширює вимоги доступності на ширший спектр продуктів і послуг, зокрема у сфері електронної комерції, електронних комунікацій, банківських сервісів, електронних книг та інших цифрових сервісів [55].

Технічною основою застосування європейських вимог у сфері ІКТ є гармонізований стандарт EN 301549:2021, який визначає вимоги доступності до продуктів і послуг інформаційно-комунікаційних технологій, охоплюючи апаратні та програмні засоби, вебконтент, документи, а також мобільні й цифрові сервіси [17]. Стандарт інтегрує критерії WCAG 2.1 рівнів A та AA як обов'язкові вимоги до вебконтенту і доповнює їх специфічними вимогами до інших категорій ІСТ, що забезпечує перехід від загальних нормативних принципів доступності до технічних критеріїв оцінювання відповідності. Така структура дозволяє використовувати EN 301549:2021 як референтний документ у країнах ЄС, але навіть такий рівень технічної деталізації не формує спеціального механізму аналізу мовної складності, локалізації когнітивних бар'єрів і керованого спрощення текстового контенту, що є принциповим у межах цього дослідження.

У США ключову роль відіграє Section 508 of the Rehabilitation Act, який визначає вимоги доступності інформаційно-комунікаційних технологій у федеральному секторі [118]. Практична цінність акту пов'язана з поєднанням нормативних положень із прикладними методичними матеріалами, інструментами перевірки та рекомендаціями щодо закупівель і розробки цифрових систем. Правова норма доповнюється організаційною та методичною підтримкою, що забезпечує її реальне впровадження.

В Україні політика цифрової доступності формується в межах ширшої нормативної рамки безбар'єрності та розвитку доступних інформаційно-комунікаційних систем. Серед ключових документів варто відзначити постанову Кабінету Міністрів України від 21.07.2023 № 757 «Деякі питання доступності інформаційно-комунікаційних систем та документів в електронній формі» [3], а також оновлену Національну стратегію із створення безбар'єрного простору в Україні на період до 2030 року в редакції розпорядження Кабінету Міністрів України від 25.03.2025 № 294-р [4]. Наведені документи свідчать про закріплення питання цифрової доступності в Україні не лише декларативно, а й у форматі конкретних управлінських та нормативних рішень.

Наявні документи задають загальну обов'язкову рамку доступності, проте не пропонують завершеного спеціалізованого механізму для автоматизованого забезпечення когнітивної складності текстового контенту. Нормативно-правова база виступає необхідною відправною точкою, проте не замінює потреби у власних методах оцінювання, локалізації бар'єрів і керованого спрощення тексту.

Нормативно-правова база та система стандартів вебдоступності формують необхідну основу для забезпечення доступних цифрових ресурсів, проте не дають повного інструментарію для автоматизованого виявлення й зниження когнітивної складності текстового контенту (табл. 1.2).

Таблиця 1.2

Основні стандарти та нормативні документи у сфері вебдоступності

Документ	Рівень застосування	Об’єкт регулювання	Релевантність для когнітивної доступності	Обмеження для задач аналізу тексту
WCAG 2.2	Міжнародний стандарт W3C	Вебконтент	Базові вимоги до зрозумілості, навігації, помилок, передбачуваності	Не дає повної формалізації мовної складності та адаптивного спрощення
ATAG 2.0	Міжнародний стандарт W3C	Інструменти авторингу	Сприяє створенню доступного контенту	Не описує безпосередньо когнітивну складність готового тексту
UAAG	Міжнародний guidance/standard overview	Браузери та користувацькі агенти	Підтримує доступ до механізмів представлення і допоміжних технологій	Не задає критеріїв спрощення текстового контенту
COGA	Спеціалізовані рекомендації W3C	Когнітивний і навчальний вимір доступності	Сильно орієнтований на ясність мови, покроковість, зниження перевантаження	Не є завершеним алгоритмічним стандартом для автоматизованого аналізу
Directive (EU) 2016/2102	Правовий акт ЄС	Вебсайти і мобільні застосунки публічного сектору	Інституційно закріплює вимоги доступності	Спирається на стандарти, але не деталізує когнітивний аналіз тексту
Directive (EU) 2019/882	Правовий акт ЄС	Доступні продукти і послуги	Поширює вимоги доступності на ширше коло цифрових сервісів	Не визначає механізмів текстового спрощення

продовж. табл. 1.2

EN 301 549	Технічний стандарт ЄС	ІКТ продукти і послуги	Технічно деталізує вимоги до доступності	Фокусується на відповідності, а не на автоматизованому виявленні мовних бар’єрів
Section 508	Нормативний акт США	ІКТ у федеральному секторі	Підтримує правозастосування та методичну інфраструктуру	Не дає спеціального методу аналізу когнітивної складності тексту
Постанова КМУ № 757, Стратегія № 294-р	Національний рівень України	Доступність ІКС та безбар’єрність	Формують національну рамку цифрової доступності	Потребують методичного доповнення для задач когнітивної доступності тексту

Особливо виразно це проявляється у випадку когнітивної доступності, де загальні критерії відповідності потребують доповнення спеціалізованими методами аналізу зрозумілості, локалізації бар'єрів і керованого спрощення тексту.

1.3. Огляд сучасних досліджень у галузі вебдоступності для людей з когнітивними та інтелектуальними порушеннями

Дослідницький фокус у сфері вебдоступності змістився від переважно нормативного й техніко-орієнтованого бачення проблеми до комплексного аналізу реальної взаємодії користувача з цифровим контентом. Ранні роботи фокусувалися насамперед на відповідності вебресурсів формальним вимогам стандартів, тоді як в останні роки значно посилився інтерес до практичної доступності, когнітивного навантаження, зрозумілості контенту та можливостей автоматизованого виявлення бар'єрів [23, 24].

Наукові дослідження вебдоступності умовно поділяються на кілька взаємопов'язаних напрямів, серед яких автоматизоване оцінювання відповідності стандартам, вивчення когнітивної доступності як окремого напряму, дослідження зрозумілості текстового контенту, автоматизоване спрощення тексту та прикладні підходи для окремих груп користувачів [83].

Значна частина сучасних праць присвячена дослідженню загальної вебдоступності та автоматизованого оцінювання. У цьому напрямі увага зосереджується на автоматичному виявленні типових проблем доступності, порівнянні інструментів перевірки, оцінюванні відповідності стандартам та розробці підходів до аналізу вебконтенту [23, 24]. Дослідження такого типу мають високу практичну цінність, оскільки дозволяють масштабувати аудит великої кількості сторінок і автоматизувати виявлення типових порушень доступності.

Аналіз ефективності автоматизованих засобів оцінки вебдоступності свідчить, що автоматизоване виявлення є найефективнішим для когнітивних порушень, які мають формалізовані технічні критерії, зокрема проблем альтернативного тексту, контрастності, структури розмітки, ролей елементів, наявності підписів, навігаційних міток або окремих WCAG-критеріїв, що можуть бути верифіковані через DOM-структуру та правила перевірки [18, 23]. Для такого класу порушень автоматизовані інструменти демонструють найбільшу ефективність і відтворюваність результатів.

Однак навіть новіші роботи в цій галузі підтверджують, що автоматизоване оцінювання має суттєві обмеження, коли йдеться про когнітивну доступність, інтерпретацію змісту, зрозумілість інструкцій, лексичну складність і реальний користувацький досвід. Інакше кажучи, автоматизовані інструменти добре виявляють частину технічних бар’єрів, але значно слабше працюють там, де потрібно інтерпретувати мову, контекст, складність тексту або очікувану когнітивну реакцію користувача [108]. Окреслена відмінність формує межу між автоматизованим аудитом доступності та задачами аналізу когнітивної доступності текстового контенту.

У сучасному науковому дискурсі когнітивна вебдоступність дедалі частіше розглядається як окремий напрям, а не як доповнення до загальних підходів забезпечення доступності. Це пов’язано з тим, що для користувачів із когнітивними та інтелектуальними порушеннями труднощі виникають не лише через технічні характеристики сторінки, а й через складність сприйняття змісту,

розуміння інструкцій, утримання уваги, послідовність виконання дій і регулювання когнітивного навантаження [59, 108]. У дослідженнях цієї сфери дедалі більше уваги приділяється мовним особливостям контенту, поведінковим аспектам взаємодії та реальному досвіду користувача.

У оглядових дослідженнях когнітивна доступність характеризується як менш інституціоналізована та формалізована порівняно з доступністю для сенсорних та моторних порушень [59]. Також звертається увага на розрив між загальними рекомендаціями, орієнтацією на користувача та технічними засобами їх практичної реалізації.

Когнітивну доступність найчастіше розглядають через повторювані аспекти, пов’язані зі зрозумілістю мовного оформлення, структурою подання, логікою взаємодії, орієнтацією в навігації, підтримкою пам’яті й уваги та контролем інформаційного навантаження [108, 136]. У такому підході оцінювання вже не зводиться лише до перевірки відповідності, а стосується того, наскільки користувач може коректно сформувати уявлення про ресурс і пройти необхідну послідовність дій без помилок.

Проблематика вебдоступності у сучасних дослідженнях дедалі частіше розглядається у зв'язку з ширшим полем цифрової інклюзії. Дослідники аналізують не лише формальні бар'єри вебконтенту, а й доступ до цифрових сервісів, рівень автономності користувача, залежність від допомоги, можливість завершувати сценарії без зовнішнього втручання та вплив когнітивного дизайну на участь людини в цифровому середовищі [108, 149].

Саме тому фокус наукових робіт поступово зміщується від загальної оцінки доступності до аналізу читабельності, простої мови, полегшеного читання та інших підходів, орієнтованих на зрозумілість мовного представлення інформації [59, 131].

Сучасні праці й далі використовують індекси читабельності тексту як швидкий індикатор формальної складності, однак дедалі частіше наголошують на їх обмеженості для оцінювання фактичної когнітивної доступності [131]. Традиційні метрики добре відображають довжину речень, морфемну або

складову структуру слів і загальну поверхневу складність, але значно менш чутливі до таких характеристик, як пояснення термінів, логічна зрозумілість інструкцій, передбачуваність структури та рівень когнітивного навантаження під час взаємодії з вебконтентом.

Паралельно посилюється інтерес до підходів простої мови та полегшеного читання, орієнтованих не стільки на формальне спрощення, скільки на досягнення зрозумілості для широких груп користувачів. У сучасних працях ці підходи розглядаються як такі, що можуть істотно підвищувати доступність цифрового контенту, особливо для людей з когнітивними порушеннями, труднощами читання або обмеженим досвідом роботи зі складними текстами [21, 59]. Такі підходи не обмежуються скороченням довжини, а працюють із ясністю лексики, послідовністю викладу, поясненням термінів і структурою повідомлення.

Для вебсередовища це має особливе значення, оскільки текст у цифровому інтерфейсі виконує одночасно кілька функцій: інформує, скеровує, пояснює, попереджає, вимагає підтвердження і підтримує прийняття рішення. Відповідно, текстова складність розглядається не лише як властивість окремого фрагмента, а як чинник, який прямо впливає на можливість завершення сценарію, кількість помилок, швидкість розуміння і рівень автономності користувача [21, 108].

Логічним кроком подальшого розгляду є звернення до досліджень автоматизованого спрощення тексту та NLP-підходів у задачах доступності.

Автоматизоване спрощення тексту розглядається як один із ключових шляхів підвищення когнітивної доступності цифрового контенту. На відміну від класичних підходів до оцінювання читабельності, які лише вимірюють формальну складність, дослідження автоматизованого забезпечення спрощення тексту орієнтуються на активне перетворення тексту з метою підвищення його зрозумілості, збереження основного змісту та зниження когнітивного навантаження [21, 101].

Важливим підходом є лексичне спрощення, тобто заміна складних, рідковживаних або спеціалізованих слів на доступніші відповідники. У нових

роботах такий підхід розглядається як перспективний для задач доступності, оскільки дозволяє локально зменшувати складність без повного переписування фрагмента і краще зберігає структуру оригінального тексту [22, 101]. Лексичне спрощення формує природну точку перетину лінгвістичного аналізу, словникових ресурсів і прикладних алгоритмів автоматичного спрощення.

Поряд із цим сучасні праці активно розвивають наскрізне спрощення тексту, у межах якого перетворення виконується на рівні фрази, речення або цілого фрагмента. Такі підходи привабливі для практичного використання, оскільки потенційно дозволяють враховувати ширший контекст, перебудовувати складні синтаксичні структури та адаптувати порядок викладення тексту [21, 57]. У цьому ж підході виникають серйозні ризики семантичного спотворення, надмірної генералізації, втрати термінологічної точності та зниження розуміння тексту.

Праці останніх років з використання великих мовних моделей у задачах спрощення показують неоднозначну картину. LLM здатні ефективно змінювати рівень читабельності, перебудовувати складні фрагменти і генерувати простіші формулювання, однак навіть лінгвістично простий текст може виявитися семантично неприпустимим або недостатньо зрозумілим для користувача, що особливо критично у задачах доступності. До цього додаються нестабільність результату, залежність від формулювання запиту, ризик неконтрольованих перефразувань та потреба в додатковій валідації [57, 131].

У сучасному дослідницькому полі дедалі виразніше проявляється прагнення поєднувати сильні сторони різних підходів, зокрема локальну керованість лексичного спрощення, ширший контекст наскрізного спрощення, оцінювання читабельності, а також генеративні можливості LLM. Таке поєднання ще не завжди реалізоване як відтворюваний, пояснюваний і технологічно стабільний метод для вебконтенту, особливо в умовах конкретної мови та конкретного домену [21, 57].

Окремим дослідницьким напрямом, що перебуває на межі когнітивної доступності та технології розпізнавання емоцій, є використання фізіологічних і

поведінкових сигналів для оцінювання когнітивного навантаження під час читання. Дослідження в цій галузі показують, що рівень когнітивного напруження може бути оцінений через окорухові дані, шкірно-гальванічну реакцію, частоту серцевих скорочень та вираз обличчя користувача [31]. Методи розпізнавання виразу обличчя (Facial Expression Recognition, FER) дозволяють виявляти ознаки напруження, розгубленості або дискомфорту, тобто станів, що корелюють з підвищеним когнітивним навантаженням під час роботи зі складним текстом [46]. Для задач вебдоступності такий підхід є перспективним, оскільки дозволяє отримувати зворотний зв'язок від користувача в реальному часі без явного введення оцінки. Наявні рішення переважно орієнтовані на загальні задачі моделювання та розпізнавання емоцій і не інтегровані з алгоритмами адаптивного спрощення тексту.

Ключовим у цьому контексті є те, що NLP- та LLM-підходи створюють достатньо сильну технологічну базу для автоматизованого спрощення, проте самі по собі не гарантують ні контрольованості, ні придатності до когнітивно-чутливого перетворення тексту. Наукова задача полягає не лише у використанні сучасних моделей, а й у побудові методу, який поєднує оцінювання складності, локалізацію бар'єрів, словникові та генеративні механізми, а також правила безпечного застосування трансформацій. Узагальнення сучасних напрямів досліджень вебдоступності для людей з когнітивними та інтелектуальними порушеннями наведено в таблиці 1.3.

Таблиця 1.3

Напрями досліджень вебдоступності для людей з когнітивними та інтелектуальними порушеннями

Напрям досліджень	Що саме досліджується	Сильні сторони	Обмеження	Значення для дисертації
Автоматизоване оцінювання доступності	WCAG-порушення, типові технічні проблеми доступності	Масштабованість, швидкість, відтворюваність	Слабке покриття когнітивно-текстових бар’єрів	Базовий шар автоматизованого аудиту
Когнітивна вебдоступність	Зрозумілість, передбачуваність, когнітивне навантаження	Фокус на реальній взаємодії користувача	Недостатня формалізація і фрагментованість інструментів	Підтверджує окремий статус задачі

продовж. табл. 1.3

Доступність текстового контенту	Readability, plain language, easy-to-read	Пояснює центральну роль тексту як носія бар'єрів	Часто не переходить до локалізації і трансформації	Формує міст до аналізу складності тексту
Автоматизоване спрощення тексту	Лексичне, реченнєве і конвеєрне спрощення	Наближає оцінювання до реальної трансформації тексту	Проблеми контролю якості і переносимості	Безпосередньо релевантне авторському методу
NLP та LLM-підходи	Класифікація, ранжування, перефразування, генерація	Сильне контекстне моделювання і гнучкість	Пояснюваність, стабільність, ризик семантичного зсуву	Показує потребу в гібридному контрольованому рішенні

Окремий сегмент сучасних досліджень становлять праці, присвячені цифровій доступності для конкретних груп користувачів. На відміну від більш узагальнених оглядів доступності, ці дослідження дозволяють побачити, як різні когнітивні профілі породжують різні вимоги до тексту, інтерфейсу і способів взаємодії з вебконтентом. Саме такий зріз є важливою частиною дослідження, оскільки показує, що універсальне уявлення про доступний текст потребує уточнення на рівні користувацьких сценаріїв і типів когнітивного навантаження.

Дослідження доступності для користувачів з дислексією переважно зосереджуються на проблемах читання вебконтенту, персоналізації подання, типографічних параметрах, налаштуванні інтервалів, контрасту та спрощенні лексики. Систематичний огляд [49, 52] свідчить, що доступність для цієї групи найчастіше підвищується через поєднання змін у візуальному представленні тексту та зниження його мовної складності. Навіть за сучасного розвитку дизайну доступності текст залишається головним носієм бар'єру і потребує не лише стилістичних, а й алгоритмічних засобів адаптації.

Для користувачів із СДУГ сучасні дослідження зміщують увагу з окремих формальних характеристик сторінки на спосіб подання інформації, структуру вебконтенту, щільність повідомлень і навантаження на увагу. У працях [106, 117] показано, що для групи користувачів із СДУГ більш прийнятними є короткі, чітко організовані та візуально сегментовані інформаційні блоки, орієнтовані на

виконання дій. Це вказує на те, що когнітивна доступність у такому випадку визначається не лише змістовими характеристиками, а й локалізацією інформації, передбачуваністю структури та зниженням чинників, що відволікають.

Для користувачів з розладом аутистичного спектру (РАС) дослідження акцентують увагу на послідовності інтерфейсу, його передбачуваності, зменшенні сенсорного та інформаційного перевантаження, а також зрозумілості функціональних елементів. Поряд із класичними рекомендаціями AASPIRE, новіші праці враховують не лише експертні підходи, а й аналіз реального досвіду взаємодії, зокрема відгуків користувачів про цифрові продукти [59, 110]. Це дозволяє розглядати доступність не тільки як набір правил, а як виявлення конкретних ситуацій непорозуміння та перевантаження під час роботи з інтерфейсом.

У дослідженнях, присвячених людям з інтелектуальними порушеннями, акцент робиться на необхідності зрозумілого формулювання інформації, поетапного подання змісту, спрощення інструкцій і зменшення абстрактності. Дослідження [103] показує, що навіть адаптація інструмента оцінювання з урахуванням когнітивної доступності забезпечує точніше розуміння завдань і ефективнішу взаємодію з матеріалом. Це підтверджує, що доступність у такому випадку визначається не лише наявністю контенту, а й відповідністю способу його подання можливостям послідовного розуміння.

Дослідження користувачів з дислексією, СДУГ, аутизмом та інтелектуальними порушеннями фіксують суттєві відмінності у когнітивних особливостях окремих груп [59, 103]. Попри ці відмінності сучасні праці [52, 117] для окремих груп користувачів виявляють повторювані вимоги до вебконтенту. До спільних вимог належать ясність мови, структурна передбачуваність, контрольований обсяг інформації, локалізація проблемного фрагмента та можливість зменшення когнітивного навантаження. Повторюваність цих вимог створює підстави для побудови узагальненого

програмного методу, що не зводиться до одного діагнозу, а орієнтується на типові категорії бар'єрів.

Узагальненна структура досліджень у сфері когнітивних та інтелектуальних порушень наведена на рисунку 1.2.



Рис. 1.2. Структура сучасних досліджень у сфері когнітивної вебдоступності.

1.4. Наявні методи аналізу тексту для людей з когнітивними обмеженнями

У межах вивчення проблематики когнітивної доступності текст виступає не лише носієм інформації, а й головним джерелом бар’єрів, пов’язаних із лексичною складністю, синтаксичною щільністю, термінологічною насиченістю, неоднозначністю інструкцій і перевантаженням робочої пам’яті. Саме тому в науковій літературі сформувався широкий спектр методів аналізу тексту, які відрізняються як за типом використовуваних ознак, так і за рівнем формалізації, вимогами до даних, інтерпретованістю та придатністю до автоматизованого спрощення [87, 102].

Підходи до формального оцінювання складності тексту. Найдавніший і водночас найпоширеніший клас методів аналізу тексту становлять моделі читабельності засновані на формулах, у яких складність оцінюється через відносно прості кількісні індикатори, зокрема середню довжину речень, довжину слів, частку складних лексем, кількість складів або інші поверхневі ознаки

тексту. Сучасні дослідження показують, що такі індикатори й надалі зберігають практичну цінність завдяки швидкості обчислення, прозорості й придатності до масового аналізу великих корпусів [87, 102].

Метрики читабельності розроблялися для швидкої оцінки доступності тексту без проведення тестів на розуміння прочитаного. Наведені показники виконують функцію швидких тестових індикаторів і дозволяють порівнювати тексти між собою, відстежувати тенденції та приблизно оцінювати рівень складності.

Сучасне поле оцінювання читабельності виходить за межі класичних формул, таких як Flesch [56] чи Gunning Fog [62]. Новіші праці розглядають читабельність як багатовимірну властивість тексту, у якій поряд із довжиною речень та слів враховуються текстова зв'язність, морфологічні й синтаксичні ознаки, а також параметри, отримані засобами обробки природної мови (NLP) [102]. У роботах останніх років запропоновано моделі, що поєднують поверхневі, нейронні та гібридні ознаки для оцінювання складності на рівні речення і документа [42, 87].

Застосування класичних формул читабельності до україномовних текстів пов’язане з методологічними викликами, оскільки формули розроблялися на англомовному матеріалі, де кореляція між кількістю складів і складністю слова має інший характер, ніж в українській мові. Морфологічна насиченість української мови зумовлює більшу середню довжину слів за рахунок флексій, префіксів та суфіксів, що не завжди свідчить про підвищену складність для читача. Синтетична будова української мови призводить до того, що одне українське слово часто передає значення, яке в англійській виражається кількома окремими лексичними одиницями, і пряме перенесення кількісних порогів із англомовних формул стає некоректним.

Складо-орієнтовані формули апроксимують складність слова кількістю складів, а складність речення визначають середньою довжиною речення в словах. До цього класу належить індекс Flesch-Kincaid [56], що повертає як коефіцієнт простоти тексту, так і його інтерпретацію у вигляді відповідного

рівня освіти. Формули SMOG [92] та Gunning Fog [62] також входять до цієї групи, проте орієнтуються на концентрацію складних слів, визначених через поріг у три або більше складів. Перевагою складо-орієнтованих метрик є простота обчислення та висока можливість пояснення результатів через чітку ідентифікацію чинників, що впливають на показник. Для української мови використання таких метрик потребує адаптації алгоритмів підрахунку складів, процедур токенізації та інтерпретації порогових значень, оскільки морфологічна довжина слова та багатоскладовість не завжди відповідають реальній семантичній складності лексичної одиниці [42, 87].

Символьно-орієнтовані формули оцінюють складність слова через кількість символів замість кількості складів. Типовими представниками підходу є індекси ARI [123] та Coleman-Liau [40], що характеризуються швидкістю обчислення та простотою програмної реалізації. Для флективних мов з багатою морфологією такі формули дають систематичні відхилення, оскільки збільшення довжини словоформи в українській мові часто зумовлене граматичним контекстом і не обов’язково свідчить про незнайомість слова для читача [87, 102].

Словниково-орієнтовані підходи визначають складність не через структурні характеристики слова, а через ступінь його знайомості для цільової аудиторії читачів. Слово класифікується як складне, якщо не входить до списку найбільш уживаної або базової лексики відповідного рівня. Найвідомішим прикладом такого підходу є формула Dale-Chall [34] та її модифікації, де частка слів поза визначеним простим словником поєднується із середньою довжиною речення. Для української мови словниково-орієнтований підхід становить протиставлення структурним метрикам і наближає оцінювання до реальної когнітивної знайомості слів для читача. Підхід узгоджується з лексичною базою системи, що включає словники рідковживаної та застарілої лексики, абревіатур і стилістично маркованих одиниць. Чутливість словникового підходу до якості використовуваних списків означає, що помилки та неповнота базового словника безпосередньо впливають на результати оцінювання.

Певне окреме місце посідають системи рівневого оцінювання, такі як Lexile Framework [127], а також підходи, що поєднують формальні індикатори з освітньо орієнтованими шкалами складності. Їхня перевага полягає в можливості узгодженого ранжування текстів і побудови порогів придатності для різних груп читачів. Подібні системи добре показують загальний рівень складності, але майже не локалізують конкретний бар'єр і не дають прямої основи для керованого спрощення фрагмента [87, 102].

Довжина словоформи у флективних мовах зумовлюється граматичними чинниками, а не рідкісністю чи термінологічним характером лексичної одиниці. Слова «організація» та «організаціями» позначають одне поняття, проте друга форма становить орудний відмінок множини з додатковими літерами внаслідок відмінювання. Символьно-орієнтовані індекси, що підвищують оцінку складності зі зростанням середньої кількості символів у слові, інтерпретують граматично очікувані форми як ускладнення тексту, що не відповідає реальному когнітивному навантаженню для носія мови [102].

Із позиції вебдоступності та автоматизації спрощення метрики мають відповідати трьом властивостям, якими є можливість пояснення, обчислюваність та придатність для керування трансформаціями. Інтерпретованість забезпечує користувачеві розуміння того, чому текст отримав саме такий бал і які фактори його знижують. Автоматичні рекомендації без пояснення причин сприймаються як непрозорі та викликають недовіру, особливо у роботі з навчальним матеріалом. Підтримка інтерпретованості потребує декомпозиції метрики на окремі компоненти й фактори, оскільки користувач має бачити, яка саме стратегія покращення є доречною у конкретному випадку, чи то пояснення терміну, розгортання абревіатури, заміна лексеми, чи синтаксичне переформулювання [42, 87].

Обчислюваність метрики на українських текстах передбачає відтворюваність результату, стійкість до типових вебформатів та незалежність від зовнішніх сервісів. Аналіз виконується багаторазово на різних сторінках та фрагментах у кількох ітераціях спрощення, тому ресурсомісткі або нестабільні в

обчисленні метрики блокують практичне використання системи. Базові показники у розробленій системі обчислюються локально за допомогою правил, словників та стабільних процедур обробки природної мови, тоді як складніші компоненти підтримують кешування або контрольований запуск, що підвищує відтворюваність експериментів та дозволяє порівнювати результати до й після трансформацій [42, 87].

Придатність для керування трансформаціями виявляється у здатності метрики визначати напрям покращення тексту, а не лише оцінювати рівень його складності. В автоматичному спрощенні метрика має визначати не лише наявність складності, а й порядок конкретних змін, які забезпечують приріст доступності при збереженні змісту. Кожен вимір метрики у розробленій системі пов’язується з відповідним класом трансформацій. Лексичні погіршення активують словникові заміни або пояснення термінів, синтаксичні погіршення активують правила розбиття та перебудови речень, а зниження загальної читабельності контролюється через довжину речень і щільність багатоскладових слів. Операційно орієнтована метрика підтримує ітераційність процесу, оскільки після кожної серії змін вимірюється приріст якості та фіксується момент зупинки, коли подальші втручання дають малий ефект або зростає ризик семантичних втрат [42, 87].

Наведені формули дають змогу швидко отримати інтегральний показник складності, порівнювати тексти між собою, задавати порогові значення і виявляти фрагменти, що потенційно потребують спрощення. Такі підходи залишаються важливою основою для автоматизованих систем первинної діагностики тексту [42, 87]. Попри корисність метрик читабельності, вони не дають відповіді на запитання, які саме бар'єри формують когнітивне навантаження та яким чином їх можна цілеспрямовано усунути у процесі спрощення тексту (рис. 1.3). Для людей із когнітивними та інтелектуальними порушеннями складність тексту не зводиться до довжини слів або речень.

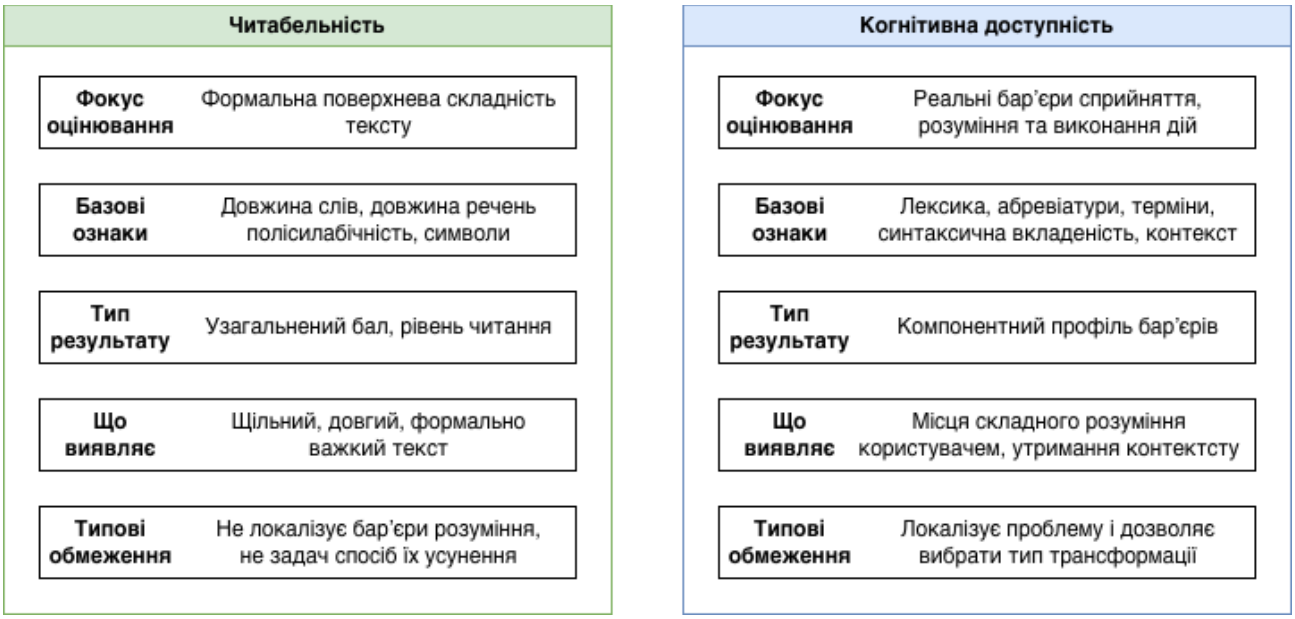


Рис. 1.3. Різниця між читабельністю та когнітивною доступністю.

Коротка абревіатура або незнайомий жаргонний елемент створює бар'єр розуміння, значно сильніший за той, який пов'язаний із довшим, але звичним словом. Формально коротке речення з вкладеними конструкціями або порушеним базовим порядком слів вимагає більших інтерпретаційних зусиль, ніж довше речення з простою синтаксичною структурою. Відповідно, когнітивна складність є багатовимірним явищем, яке окремі показники читабельності охоплюють лише частково.

Лексико-статистичні та словникові методи аналізу. Другий клас методів ґрунтується на аналізі словникового складу тексту. Складність у методах цього класу пов'язується з рідковживаною, термінологічною, багатозначною або морфологічно ускладненою лексикою, а також із частотністю появи слів у корпусі, інформаційною щільністю фрагмента й відхиленням від базового словника зрозумілих одиниць. На практиці для цього використовують частотні словники, ранги лексем, статистики TF-IDF, векторні представлення слів, подання fastText, а також спеціалізовані словники спрощення й лексичної заміни [26, 76].

Актуальні дослідження демонструють, що лексико-статистичні методи особливо корисні для локалізації потенційно проблемних слів, побудови

рейтингу кандидатів на заміну або оцінювання складності окремої лексеми в конкретному контексті. Цей напрям заклав основу для сучасних задач прогнозування лексичної складності та лексичного спрощення, зокрема у багатомовному контексті, що активно досліджуються в межах ACL, TSAR тощо [98, 99].

Серед підходів цього класу виділяються методи, що спираються на частотні й вагові характеристики слова, зокрема TF-IDF, частотне векторне представлення та споріднені статистичні представлення. Такі методи оцінюють, наскільки нетиповою або інформаційно насиченою є лексема відносно корпусу, і забезпечують первинне виявлення рідковживаних, термінологічних або потенційно складних одиниць. У задачах когнітивної доступності вони виконують функцію локального підсвічування проблемного місця в тексті, не претендуючи на розкриття семантичної ролі слова у висловлюванні [26, 76].

Ближчими до прикладного спрощення виступають словникові моделі, такі як D-Level, списки простих слів, термінологічні словники та ресурси лексичної заміни. Такі моделі дозволяють не лише фіксувати потенційну складність слова, а й пов'язувати його з конкретним класом втручань, тобто поясненням, заміною, додаванням контексту або вибором простішого відповідника. Словникові ресурси мають особливу цінність у контрольованих системах покращення тексту, де важливо пояснення кожного кроку [98, 99].

Окремий підклас становлять векторні представлення тексту, зокрема Word2Vec [95], GloVe [104], Doc2Vec [84] і fastText [28]. На відміну від суто частотних схем, наведені представлення моделюють подібність слів і фрагментів у просторі ознак, що дозволяє ранжувати кандидатів на заміну та оцінювати контекстну близькість простішого варіанта до вихідної одиниці. Для українського матеріалу особливий інтерес становить fastText, який краще працює з морфологічно варіативними словами завдяки субслівним компонентам і слугує проміжним кроком між класичною статистичною векторизацією та сучасними контекстуалізованими мовними моделями [75, 76].

Векторні представлення оперують відстанями у просторі ознак, що дозволяє виявляти не лише наявність складної лексеми, а й її потенційні замінники зі збереженням семантики [26]. Їхні можливості обмежені без урахування синтаксису, дискурсивної структури та комунікативного контексту, оскільки складне для сприйняття речення може містити лише відносно прості слова і залишатися когнітивно важким через щільну структуру, приховані відношення або перевантажені інструкції [98].

Формальні та лексико-статистичні методи утворюють необхідний нижній рівень автоматизованого аналізу тексту. Формули читабельності забезпечують швидку й пояснювану оцінку загальної складності, а лексико-статистичні підходи локалізують лексичні джерела бар'єрів і формують основу для словникових та статистичних механізмів спрощення.

Синтаксичні та засновані на правилах методи аналізу тексту. Синтаксична організація тексту становить самостійний рівень когнітивної складності, який не зводиться до складності окремих слів. Синтаксичні методи оцінювання працюють зі структурою речення, рівнем вкладеності, типами залежностей, кількістю предикативних центрів, довжиною синтаксичних ланцюгів, наявністю вставних конструкцій, умовних або причинно-наслідкових зв'язків. Наведені ознаки часто визначають, наскільки складно користувачеві побудувати цілісну ментальну модель повідомлення навіть тоді, коли окремі слова залишаються зрозумілими [42, 76].

Отримання синтаксичних ознак у сучасних системах оцінювання складності здійснюється як засобами традиційного лінгвістичного аналізу, так і через синтаксичний парсинг залежностей, складниковий парсинг (constituency parsing) та інші інструменти структурного подання речення, що дозволяє виявляти конкретні структурні джерела труднощів, серед яких надто довгі речення, багаторівнева вкладеність, неочевидні відношення між частинами фрази, перевантажені інструкції або синтаксична компресія [42, 76].

Методи, засновані на правилах, спираються на принципи простої мови, полегшеного читання, редакторські настанови та правила [90], доповнюють

синтаксичний аналіз механізмом цілеспрямованого втручання в текст [53], а їхньою ключовою перевагою залишається висока керованість і можливість пояснення [57], оскільки правила можуть прямо забороняти складні конструкції, рекомендувати короткі речення, усунення вкладеності, заміну пасивних форм, конкретизацію референтів, розбиття інструкцій на кроки та спрощення дискурсивної організації [53].

Обмеження підходів, заснованих на правилах, пов'язані з вузькістю сценаріїв застосування та залежністю від ручного супроводу, що виявляється у труднощах масштабування на великі різнорідні корпуси та недостатньому врахуванні контексту. До цього додається різниця між простою мовою та полегшеним читанням за ступенем спрощення, де полегшене читання орієнтується на значно суворіші вимоги до простоти, що робить його корисним орієнтиром для доступності, але не завжди достатнім як універсальний автоматизований режим трансформації [57, 90].

Поєднання синтаксичних і методів, заснованих на правилах, формує базовий інструментарій автоматизованої роботи з когнітивною доступністю тексту, у якому синтаксичні методи фіксують структурну складність і локалізують синтаксичні бар'єри, а методи правил забезпечують керований механізм цілеспрямованого спрощення, причому ефективність наведених методів суттєво зростає за умови їх поєднання з даними про лексику, статистичними сигналами та машинними моделями, здатними узагальнювати шаблони за межами вручну заданих правил.

Класичні методи машинного навчанн, які працюють поверх наперед визначених ознак, становлять окремий напрям аналізу тексту [19]. До цієї групи належать логістична регресія, метод опорних векторів, випадковий ліс, метод k-найближчих сусідів, наївний баєсівський класифікатор та споріднені моделі [26], які у задачах аналізу тексту зазвичай використовують векторні представлення, такі як мішок слів (Bag-of-Words), TF-IDF, статистичні лінгвістичні ознаки, та комбіновані набори характеристик з охопленням лексики, синтаксису та метаданих документа [76].

Класичне машинне навчання забезпечує баланс між точністю, швидкодією, контрольованістю та відносно низькими вимогами до обчислювальних ресурсів [19]. Лінійні моделі логістичної регресії та методу опорних векторів демонструють високу стабільність на добре підготовлених ознаках і працюють тоді, коли текст уже подано у вигляді інформативного векторного опису [26]. Логістична регресія цінується за простоту інтерпретації та стійкість як модель порівняння, тоді як метод опорних векторів часто забезпечує кращу роздільну здатність у задачах текстової класифікації за рахунок більш гнучкого розділення класів у просторі ознак [144].

Ансамблеві методи, зокрема випадковий ліс, градієнтний бустинг, XGBoost і LightGBM, краще працюють із неоднорідними наборами характеристик і дозволяють моделювати нелінійні взаємодії між ознаками [26]. Для задач аналізу складності тексту це має значення тому, що реальний бар'єр часто формується не однією метрикою, а поєднанням лексичних, синтаксичних і структурних факторів [19]. Випадковий ліс додатково дозволяє оцінювати важливість ознак і підтримує пояснюваність краще за багато інших нелінійних моделей [144].

Простіші підходи, такі як баєсівський класифікатор та метод k-найближчих сусідів, зберігають своє значення як базові моделі порівняння. Наївний баєсівський класифікатор є швидким і зручним інструментом для лексично орієнтованих задач, тоді як метод k-найближчих сусідів використовується як інтуїтивно зрозуміла модель, у якій рішення приймається на основі близькості до вже відомих прикладів. Роль наведених моделей у межах цього дослідження полягає передусім у забезпеченні контрольних орієнтирів, від яких можна відштовхуватися під час оцінювання складніших підходів [26, 144].

У порівняльних дослідженнях класичні моделі машинного навчання дедалі частіше виступають базовими орієнтирами для трансформерних рішень [26]. Такі праці показують, що на багатьох прикладних задачах традиційні алгоритми поступаються моделям за якістю узагальнення й контекстного розуміння [144], проте зберігають цінність через інтерпретованість, меншу обчислювальну

вартість і придатність до сценаріїв з обмеженими даними або жорсткими вимогами до відтворюваності [19].

Для задачі когнітивної доступності тексту класичне машинне навчання має особливу цінність через природне поєднання з інженерно зрозумілими ознаками. Якщо модель побудована на частотності, довжині речень, синтаксичній вкладеності, термінологічній щільності або словникових категоріях, її поведінка значно краще піддається поясненню, ніж у повністю наскрізних нейронних системах. Цей клас методів закономірно посідає проміжне положення між формальними показниками складності та сучасними глибокими моделями [26, 76].

Обмеження класичного машинного навчання пов'язані з залежністю від якості ручної інженерії ознак, слабким урахуванням довгих залежностей і контекстуальних смислових зсувів, а також з необхідністю окремого механізму для задач, що виходять за межі класифікації і потребують генерації спрощених формулювань. Саме ці обмеження стимулювали перехід до глибоких нейронних мереж і трансформерних архітектур у задачах обробки природної мови, які доцільно розглянути окремо [19, 26].

Методи глибокого навчання у задачах аналізу тексту знизили залежність моделей від ручної інженерії ознак і відкрило можливість самостійного вивчення корисних представлень текстових даних. До цього класу належать CNN, RNN, а також архітектури LSTM і GRU, що застосовуються у текстовій класифікації, оцінюванні складності, виявленні тематичних і стилістичних характеристик та інших задачах з опорою на узагальнення над послідовностями слів [113, 146].

CNN ефективно виділяють локальні шаблони у тексті [146], тоді як RNN разом із LSTM і GRU працюють із послідовною природою мовлення та довшими залежностями між токенами [113]. У прикладних дослідженнях поширені гібридні архітектури, такі як BiLSTM-CNN або CNN-LSTM, з поєднанням локального виділення ознак з моделюванням послідовності [145].

Застосування CNN у задачах аналізу тексту зосереджується на виявленні локальних лексичних та синтаксичних патернів, зокрема стійких

словосполучень, поєднань тригерних слів, коротких структурних шаблонів або маркерів складних інструкцій. Згорткові мережі показали свою ефективність у класифікації коротких і середніх текстових фрагментів, де значення мають повторювані локальні конфігурації ознак [113, 145].

RNN разом із LSTM і GRU пристосовані до моделювання послідовності та накопичення контексту в межах речення або фрагмента, що є суттєвим для задач когнітивної доступності, оскільки складність часто виникає не на рівні окремого слова, а у послідовному нарощенні змісту, відкладених залежностях, вставних конструкціях і багатокрокових інструкціях. Рекурентні мережі сформували важливий етап переходу від поверхневих ознак до моделей, здатних ураховувати структуру текстового потоку [113, 146].

Гібридні архітектури BiLSTM-CNN [145] і CNN-LSTM [113] використовуються у прикладних дослідженнях для одночасного врахування коротких патернів і довших контекстних залежностей. Такі моделі позначили поступовий перехід до більш контекстно чутливих систем, проте ще не забезпечили достатнього рівня прозорості й контрольованості, що принципово для задач доступності.

Порівняльні праці останніх років показують, що моделі глибокого навчання попереднього покоління перестали бути беззаперечним стандартом у задачах оцінювання складності та класифікації тексту. У багатьох задачах їх витісняють трансформерні підходи, які краще враховують глобальний контекст і мають сильніші механізми перенесення знань [26, 113].

Методи обробки природної мови на основі трансформерів. Сучасний стан обробки природної мови формується під впливом попередньо навчених мовних моделей на базі трансформерної архітектури [148]. У текстовій аналітиці це зумовило перехід від систем, що працюють переважно на ручних або локально навчених ознаках, до моделей з контекстуалізованими представленнями слів і речень [130]. До цього класу належать BERT, RoBERTa, DistilBERT, багатомовні трансформерні моделі та їхні модифікації, що сьогодні стали де-факто стандартом для багатьох задач класифікації й семантичного аналізу [44].

Трансформерні моделі суттєво краще працюють із контекстом, далекими залежностями, варіативністю мовних конструкцій і переносом знань між доменами. Для аналізу складності тексту це означає можливість враховувати не лише окремі слова чи локальні шаблони, а й глобальну побудову змісту, приховані залежності між фрагментами та контекстну роль термінів і інструкцій [130, 148].

Базову роль у цьому класі відіграє BERT, який заклав практику використання контекстуалізованих представлень слів і речень для широкого спектра задач обробки природної мови. В аналізі складності тексту BERT забезпечує врахування значення лексеми не ізольовано, а в контексті всього висловлювання, що підвищує чутливість до прихованих залежностей, термінологічної ролі слова та складних синтаксичних конфігурацій [44, 148].

RoBERTa та споріднені модифікації розвинули підхід за рахунок інтенсивнішого попереднього навчання та кращого використання корпусних даних [149]. У прикладних порівняннях наведені моделі часто демонструють вищу якість класифікації й ранжування текстів за складністю [146], особливо у сценаріях з тонкими відмінностями між близькими за формою, але різними за когнітивним навантаженням фрагментами [130].

Sentence-BERT і багатомовні трансформерні моделі підтримують роботу з семантичною близькістю, пошуком подібних формулювань, ранжуванням кандидатів на спрощення та перенесенням моделей між мовами [130]. Для українського матеріалу це має практичне значення, оскільки задача виходить за межі бінарної класифікації [146] і потребує оцінювання близькості варіантів переформулювання та вибору найпридатнішого спрощеного фрагмента [148].

Прикладну інфраструктуру для оцінювання складності тексту доповнюють інструментальні платформи обробки природної мови, серед яких Stanza, spaCy та NLTK. Платформи не виконують оцінювання складності самостійно, проте забезпечують токенізацію, лематизацію, розмітку частин мови, синтаксичний розбір залежностей та набір лінгвістичних ознак для роботи статистичних, машинних та гібридних підходів. Інструментальні платформи NLP формують

технологічну основу для побудови систем аналізу доступності тексту.

Прикладні дослідження фіксують вищі результати трансформерних моделей порівняно з класичними алгоритмами машинного навчання та попередніми архітектурами глибокого навчання [146]. Перевага виявляється у сценаріях класифікації тексту, адаптації в умовах обмежених даних, перенесення між доменами і задачах з високою чутливістю до контексту [148]. Водночас наведені моделі є ресурсомісткими, менш прозорими для інтерпретації та складнішими для стабільного впровадження, особливо у роботі з українською мовою [130].

Проміжне положення між класичними статистичними представленнями тексту і контекстуалізованими попередньо навченими мовними моделями займає fastText та споріднені підходи на основі векторних вкладень. FastText зберіг практичну цінність як швидкий і відносно легкий інструмент побудови текстових представлень у сценаріях, де критичними є швидкодія, обмежені ресурси або необхідність працювати з морфологічно варіативними словами [75].

Глибоке навчання і трансформерні моделі у задачах обробки природної мови розширили можливості автоматизованого аналізу тексту за рахунок кращого узагальнення, глибшого моделювання контексту і вищої якості класифікації. Одночасно зросли вимоги до обчислювальних ресурсів, інтерпретованості та контролю поведінки моделей, що для задачі когнітивної доступності робить принциповим поєднання потужних моделей з керованими механізмами локалізації бар'єрів і пояснюваного спрощення. Варто розглянути порівняльні характеристики NLP-підходів та моделей машинного навчання (табл. 1.4).

Таблиця 1.4

Порівняльна характеристика моделей машинного навчання та NLP-підходів

Клас / модель	**Тип ознак або представлення**	**Сильні сторони**	**Типові обмеження**
Метод опорних векторів / логістична регресія	TF-IDF, ручні лінгвістичні ознаки	Стабільність, інтерпретованість	Слабке моделюван-ня складного контексту

продовж. табл. 1.4

Випадковий ліс / k-найближчих сусідів / наївний баєсівський класифікатор	Статистичні та змішані ознаки	Прості baseline, робота з ознаками	Обмежена контекстна чутливість
CNN / RNN / LSTM / GRU	Навчені векторні подання послідовностей	Краще виявлення шаблонів і залежностей	Поступаються transformers у глобальному контексті
fastText / векторні вкладення	Легкі векторні представлення	Швидкодія, придатність до обмежених ресурсів	Менша контекстність
BERT / RoBERTa / багатомовна PLM	Контекстуалізовані представлення	Висока якість класифікації й аналізу	Вартість обчислень і складність пояснення
LLM	Генеративні контекстні моделі	Гнучке спрощення і перефразування	Нестабільність, потреба в жорсткому контролі

Методи автоматизованого спрощення тексту та великі мовні моделі. Автоматизоване спрощення тексту спрямоване не лише на оцінювання складності, а й на безпосереднє перетворення складного фрагмента тексту у доступнішу форму [98]. У межах цього напряму виділяють спрощення на лексичному рівні, на рівні речення та через систему правил, а також гібридні й конвеєрні підходи, що поєднують кілька типів трансформацій [53]. Найбільший дослідницький інтерес становлять системи, які охоплюють повний ланцюг перетворень, а саме виявлення складної одиниці, генерацію кандидатів, ранжування, перевірку обмежень і оцінювання якості спрощення [99].

Праці 2024–2025 років демонструють, що LLM ефективно покращують доступність тексту, генерують спрощені варіанти та адаптують формулювання під різні рівні складності, часто без спеціалізованого донавчання [36. 90]. Такий підхід продовжує загальну логіку методів автоматизованого спрощення, яка раніше спиралася на правила, статистичні моделі та нейронні архітектури зі спеціалізованим навчанням [87]. Перехід до великих мовних моделей суттєво розширив діапазон доступних трансформацій і знизив залежність від ручної інженерії ознак, водночас створивши нові виклики щодо контрольованості й передбачуваності генерації [36].

У задачах когнітивної доступності генеративні моделі одночасно створюють нові ризики. LLM здатні продукувати семантичний зсув, нестабільні результати, зайву варіативність, фактичні помилки або надмірне спрощення, що руйнує змістовий баланс тексту [36]. До цього додаються питання відтворюваності, контролю за рівнем втручання [98] та етичних меж використання генеративних систем у задачах доступності [57].

Прикладні дослідження дедалі частіше фіксують, що LLM мають найбільшу цінність не як автономний і неконтрольований інструмент, а як генеративний компонент у складі гібридних систем [87]. У таких архітектурах словникові, статистичні, синтаксичні та механізми, засновані на правилах, забезпечують локалізацію бар'єрів і базовий контроль [99], тоді як LLM виконують роль гнучкого переформулювача у випадках, де потрібна генерація варіантів спрощення з урахуванням контексту [90].

Поряд із текстоцентричними підходами у сучасних дослідженнях когнітивної доступності розвиваються методи комп'ютерного зору та розпізнавання виразу обличчя, що аналізують зовнішній поведінковий сигнал користувача. На відміну від методів, які працюють лише з мовними властивостями тексту, FER дозволяє враховувати реакцію конкретного користувача на конкретний фрагмент контенту в реальному часі. FER-сигнал не є самодостатнім для визначення способу спрощення тексту і потребує поєднання з методами оцінювання складності та алгоритмами трансформацій.

Мультимодальні підходи поєднують FER з іншими фізіологічними сигналами, зокрема ЕЕГ, шкірно-гальванічною реакцією та частотою серцевих скорочень, і забезпечують точніше розпізнавання емоційних станів, включно з когнітивним навантаженням [124]. Наведений підхід потребує спеціалізованого обладнання і є складним для масового практичного застосування, тоді як FER через звичайну вебкамеру залишається найдоступнішим варіантом зворотного зв'язку у реальному вебсередовищі, де користувач не має додаткових датчиків [86]. Робота в реальному часі висуває додаткові вимоги, серед яких мінімізація затримки обробки, стійкість до шуму, варіацій освітлення і різних

кутів повороту голови, а проблема нестабільності миттєвих класифікацій вирішується через часову стабілізацію сигналу за допомогою ковзного вікна або механізмів корекції множинного тестування [114, 115].

Методи FER спираються переважно на глибоке навчання. Дослідження 2023 року зафіксувало вищу на 26,95 % точність архітектури на базі Squeeze-and-Excitation з ResNet, навченої на наборі даних AffectNet та валідованої на RAF-DB, за моделі без трансферного навчання [65]. Систематичні огляди підтверджують домінування CNN, трансформерних та гібридних архітектур (CNN+RNN, BiLSTM) у задачах розпізнавання виразу обличчя зі стандартними наборами AffectNet, FER+, RAF-DB та EmotiW як еталонами для оцінювання якості [74]. У прикладних системах поширені попередньо навчені рішення, такі як DeepFace, з уніфікованим інтерфейсом до кількох базових архітектур та можливістю використання без донавчання для базового розпізнавання емоцій [119].

Інтеграція FER у задачі доступності спирається на встановлений зв'язок між виразом обличчя і когнітивним навантаженням. Системи на базі моделей, навчених на AffectNet, класифікують не лише базові емоції, а й стани залученості, розгубленості, нудьги та здивування, що безпосередньо корелюють з рівнем когнітивного навантаження [64, 116]. Типовими мімічними проявами підвищеного когнітивного навантаження при роботі зі складним текстом є нахмурення, звуження очей, напруження м'язів чола та мікровирази здивування або розгубленості, що фіксуються FER-системами як негативно забарвлені або нейтрально-напружені емоції [112]. Для вебдоступності такий підхід відкриває можливість виявляти ознаки ускладненого сприйняття тексту без явного введення з боку користувача.

Порівняльний аналіз наявних методів показує, що кожен із них охоплює лише частину задачі когнітивної доступності тексту. Формульні та статистичні методи добре дають швидку оцінку загальної складності, але слабко локалізують причини труднощів [102]. Лексико-статистичні й словникові підходи краще виявляють проблемні слова, однак без синтаксичного і дискурсивного контексту

залишаються неповними [76]. Синтаксичні та засновані на правилах методи добре працюють для контрольованого спрощення, але потребують значного ручного проєктування (табл. 1.5) [53].

Таблиця 1.5

Порівняльна характеристика класів методів аналізу тексту для людей з когнітивними обмеженнями

Клас методів	**Що аналізує / змінює**	**Переваги**	**Обмеження**	**Релевантність для дисертації**
Формальні та методи читабельності	Загальну поверхневу складність тексту	Швидкість, прозорість, простота	Погана локалізація бар'єрів	Базовий індикативний шар
Лексико-статистичні та словникові	Проблемні слова, частотність, терміни	Локалізація лексичних бар'єрів	Слабкість без синтаксичного контексту	Критично важливі для словникового шару
Синтаксичні та правилові	Структуру речень і керовані трансформації	Пояснюваність, контрольованість	Ручне проєктування, слабша масштабованість	Основа безпечного спрощення
Класичне машинне навчання	Класифікацію на ознаках	Баланс точності й інтерпретованості	Залежність від ознак, не генерує текст	Сильний базовий орієнтир і проміжний клас
Глибинне навчання	Складні патерни в послідовностях	Краще узагальнення, ніж classic ML	Вищі ресурси, менша прозорість	Важливий етап еволюції методів
Трансформерні моделі NLP	Контекст і довгі залежності	Висока якість і контекстність	Ресурсомісткість, складність інтерпретації	Перспективний аналітичний шар
LLM та спрощення тексту	Генерацію і перефразування	Максимальна гнучкість	Ризики зсуву, нестабільності і галюцинацій	Доцільні лише у контрольованому гібриді

Наявні методи аналізу і спрощення тексту створили підґрунтя для розв'язання задач когнітивної доступності, проте жоден із них окремо не забезпечує одночасно пояснюваного оцінювання складності, локалізації бар'єрів, контрольованого спрощення, контекстної адаптації та відтворюваної роботи з українським вебконтентом.

Висновки до розділу 1

У першому розділі «Аналіз проблеми дослідження вебдоступності для людей з когнітивними та інтелектуальними порушеннями» встановлено специфіку когнітивної вебдоступності як окремої науково-практичної проблеми, проаналізовано нормативно-правову базу та міжнародні стандарти, здійснено систематичний огляд сучасних досліджень, виконано критичний аналіз наявних методів аналізу та адаптації текстового вебконтенту.

Встановлено, що вебдоступність для людей з когнітивними та інтелектуальними порушеннями не зводиться до загальних вимог сенсорної або моторної доступності та становить окрему задачу, в якій вирішальне значення мають зрозумілість тексту, передбачуваність структури, прозорість інструкцій, обмеження когнітивного навантаження та можливість безпомилкового проходження цифрових сценаріїв.

Проаналізовано нормативно-правові засади та стандарти вебдоступності, зокрема WCAG, ATAG, UAAG, COGA, європейські та національні нормативні акти. Встановлено, що чинна нормативна рамка формує необхідну основу для забезпечення цифрової доступності, однак лише частково охоплює когнітивну доступність і не надає завершеного інструментарію для автоматизованого виявлення та зниження складності текстового контенту.

Здійснено систематичний огляд сучасних досліджень, який показав суттєве зростання уваги у 2022–2025 роках до автоматизованого оцінювання вебдоступності, когнітивної доступності, зрозумілості текстового контенту, автоматизованого спрощення тексту та використання NLP і великих мовних моделей. Водночас виявлено фрагментованість наявних рішень, відсутність єдиного керованого методу та недостатню орієнтацію більшості підходів на український вебконтент.

Виконано критичний аналіз наявних методів – формальних, лексико-статистичних, синтаксичних, правилових, класичних підходів машинного навчання, архітектур глибокого навчання, трансформерних моделей та великих мовних моделей. Встановлено, що жоден з них окремо не забезпечує одночасно

пояснюваного оцінювання складності, локалізації бар'єрів, контрольованого спрощення, безпечності трансформацій та відтворюваності результатів. Обґрунтовано доцільність побудови гібридного програмного методу забезпечення когнітивної доступності тексту, що поєднує індикативну оцінку складності, локалізацію бар'єрів, словникові ресурси, синтаксичні правила та кероване використання генеративних моделей.

Матеріали розділу 1 подано в таких публікаціях автора [1, 5, 6, 7, 8, 9, 12, 15].

РОЗДІЛ 2. Методи забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями

2.1. Визначення індексу когнітивної доступності тексту для людей з когнітивними та інтелектуальними порушеннями

У задачі аналізу вебдоступності освітнього контенту для людей з когнітивними та інтелектуальними порушеннями класичні метрики читабельності не забезпечують достатньої діагностичної повноти. Метрики дають числовий показник поверхневої складності тексту, проте не охоплюють тих бар'єрів, які реально перешкоджають розумінню, утриманню контексту та виконанню інструкцій у вебсередовищі. Побудова програмного методу потребує інтегрального показника, одночасно мовно придатного для українських текстів, когнітивно інтерпретованого й операційно застосовного для автоматизованого покращення тексту.

Українська мова належить до флективних мов з розвиненою морфологією, тому довжина словоформи не завжди корелює зі складністю змісту, а синтаксис і порядок слів гнучкіші, ніж у мовах класичних формул. В англійській мові довжина слова часто пов'язана з термінологією та книжною лексикою, тоді як в українській вона зумовлена флексією – відмінками, узгодженням і дієслівними формами, тому без адаптації коефіцієнти завищують оцінку складності для звичайних граматичних форм. Стандартизація формул передбачає нормалізацію тексту (апострофи, лапки, дефіси, регістр, абревіатури), узгоджену токенізацію з правилами для скорочень і позначень на кшталт API, LLM, WCAG, підрахунок складів з урахуванням української фонетики та калібрування порогів складності для українських текстів.

Оцінювання когнітивної доступності вебконтенту є багатовимірною задачею, у якій поряд зі складністю декодування текстової інформації необхідно враховувати труднощі її осмислення, ризик втрати контексту та потенційне когнітивне навантаження під час опрацювання окремих фрагментів. Предметом розгляду у цьому дослідженні є не лише складність читання як така, а й складність розуміння та ризик когнітивного перевантаження на конкретних

ділянках тексту. Текстові матеріали містять кілька категорій бар’єрів, які мають значення для осіб з когнітивними та інтелектуальними порушеннями і залишаються поза увагою класичних метрик читабельності. Класичні метрики читабельності відповідають на запитання про складність тексту як послідовності символів і речень, але не розкривають доступність текстового змісту для конкретної цільової аудиторії. Це означає необхідність оцінювання не лише процесу читання, а й розуміння інструкцій, побудови ментальної моделі, здатності утримувати контекст та зниження ризику когнітивного перевантаження. Зіставлення бар'єрів когнітивної доступності поза увагою класичних індексів читабельності з відповідними механізмами ІКД наведено у таблиці 2.1. Наведені бар'єри простежуються на коротких прикладах з відсутністю кореляції формальних ознак простоти з реальними труднощами розуміння.

Таблиця 2.1

Бар'єри когнітивної доступності поза покриттям класичних індексів та відповідні механізми ІКД

Бар’єр	**Причина неврахування класичними метриками читабельності**	**Як це враховано в ІКД**
Абревіатури	короткі слова покращують показники	детекція абревіатур і розгортання або заміна (див. 2.4)
Терміни без пояснень	довжина слова не гарантує термінологічного характеру	детекція термінів з офлайн-поясненнями або контекстними замінами
Архаїзми/застарілі слова	короткі форми не фіксуються як складні	словники архаїзмів і застарілої лексики з механізмом замін
Рідковживані слова	не завжди є довгими або багатоскладовими	словникові ресурси з пріоритизацією замін
Сленг і нецензурна лексика	формально не визначається як складна	словники з механізмом нейтралізації або заміни
Вкладеність і підрядність	довжина речення не відображає структуру	ознаки парсингу залежностей: підрядні маркери, глибина дерева, стабільність порядку слів

З урахуванням наведених бар’єрів пропонуємо інтегральний показник ІКД – коефіцієнт, що поєднує лексичні, синтаксичні характеристики та показники

читабельності тексту в єдину шкалу [0;1]. ІКД можна застосовувати при розробці діагностичного інструмента для виявлення бар’єрів, критерію оптимізації для ітераційного спрощення тексту та показника «до/після» для оцінювання ефективності внесених змін. Це дозволить виявити текстові фрагменти, які потребують спрощення, з декомпозицією загальної оцінки на окремі компоненти та фактори. На відміну від метрик читабельності, що повертають лише підсумковий бал, ІКД формує деталізовану характеристику складності через відображення лексичних джерел навантаження, структурних причин та загальної поверхневої складності читання.

Для уникнення необґрунтованого використання різних метрик читабельності необхідно порівняння їх вплив на ІКД як базового шару оцінювання складності тексту. У складі ІКД інтегровано індикатори, що відповідають кільком критеріям відбору одночасно. Інтерпретованість забезпечує прозору інтерпретацію отриманих результатів. Відтворювана адаптація під особливості української мови зберігає валідність вимірювання. Різні проєкції складності охоплюють як структурні характеристики тексту, так і параметри лексичної знайомості.

Відбір метрик читабельності будується на забезпеченні різних, частково незалежних проєкцій складності тексту. Індекс Flesch-Kincaid реагує на поєднання довжини речення та складової структури слова і слугує базовим індикатором поверхневої читабельності. Формула SMOG зосереджується на щільності багатоскладових слів і виявляє фрагменти з високою концентрацією складної лексики навіть за відносно коротких речень. Індекс D-Level, побудований на підході Dale-Chall, оцінює складність через лексичну знайомість слова за словниковими списками, а не через його морфологічну структуру, що дозволяє компенсувати обмеження складо-орієнтованих формул на українському матеріалі. Зміщення критеріїв дозволяє контролювати систематичну похибку, характерну для застосування кількісних метрик до української мови, де довжина словоформи визначається відмінюванням і граматичним узгодженням без зв'язку з семантичною складністю. Сукупне

застосування трьох метрик дає стійкішу та збалансованішу оцінку складності тексту порівняно з одним індикатором і формує порівняльний шар оцінювання для системи ІКД (табл. 2.2).

Таблиця 2.2

Функції метрик читабельності метрик тексту

Клас метрик	Метрики	Найкраще виявляють	Особливості
Складо-орієнтовані	Flesch, Flesch–Kincaid, SMOG, Fog	довгі речення, багатоскладові слова	інтерпретованість, поширеність; придатність як базового індикатора після адаптації до української мови
Символьно-орієнтовані	ARI, Coleman–Liau	довгі слова за кількістю символів	ризик системної похибки через морфологію, слабкий зв’язок із семантичною знайомістю слова
Словниково-орієнтовані	Dale–Chall, D-Level	незнайомі слова, частотність та списки	компенсація морфологічних перекосів складо- та символьно-орієнтованих підходів, узгодженість із лексичною базою системи
Корпусні та психометричні	Lexile	емпіричні рівні складності	потреба в окремому корпусному калібруванні й освітній шкалі для української мови, складність відтворення в експериментах на поточному етапі

Ключова складність перенесення класичних формул читабельності в українській мові пов'язана з коректним визначенням базових статистичних показників і підбором коефіцієнтів та порогів інтерпретації, які не вносять системної похибки через морфологічну будову мови. Адаптація у системі ІКД охоплює три напрями – лінгвістичну нормалізацію та токенізацію, підрахунок складів і багатоскладових слів для української мови, а також формування словникової бази. Вебтексти часто містять змішану графіку кирилицею та латиницею, цифри, скорочення, дефіси, апострофи й керуючі символи розмітки, тому процедура обробки відокремлює слова від пунктуації, відмежовує абревіатури та числові маркери, відтворювано виділяє речення – особливо короткі інструкції без кінцевої крапки, де викривлення середньої довжини речення прямо впливає на результат формул. У відтворюваній моделі підрахунку склад визначається як поява голосного після приголосного, послідовність

голосних трактується як один склад, а до повноцінних слів застосовується нижня межа в один склад для уникнення нульових значень.

Кількість слів із трьома й більше складами в українській мові частково зумовлена морфологічними чинниками, а не термінологічним характером лексики, тому відповідний показник виконує функцію порівняльного індикатора щільності складних словоформ, тоді як тип складної лексики точніше локалізують лексичні компоненти ІКД. Адаптація словниково-орієнтованого підходу зосереджується на якості словникової бази, яка формується з двох взаємодоповнювальних наборів – базового словника простої та набору одиниць з високою ймовірністю складності для широкої аудиторії. Такий підхід наближає оцінювання до когнітивної знайомості слова й узгоджується з лексичними компонентами системи ІКД.

Відтворюваність результатів забезпечується через фіксацію параметрів адаптації в описі алгоритмів системи. Коефіцієнти формул впливають на абсолютні значення метрик і пороги розмежування легкого та складного тексту, а одна й та сама формула за різного трактування понять слова або складу повертає різні результати, що унеможливлює порівняння експериментів без явної фіксації параметрів.

Перекалібруванню для української мови підлягають три класичні показники читабельності: Flesch-Kincaid, SMOG та D-Level. У формулах збережено оригінальну структуру, а числові коефіцієнти уточнено на українському текстовому матеріалі за процедурою.

Калібраційний корпус сформовано як стратифіковану підвибірку з корпусу Brown-UK [30] – збалансованого корпусу сучасної української мови з морфологічною розміткою та жанровою класифікацією. Корпус охоплює 88 текстових фрагментів обсягом 200–400 слів, розподілених за 11 рівнями складності (по 8 фрагментів на рівень). Рівні складності відповідають класам загальноосвітньої програми (1–11) і визначаються за джерелом тексту: навчальні матеріали відповідного класу, дитяча та юнацька література, науково-популярні та наукові публікації. Жанровий розподіл у межах кожного рівня охоплює

дитячу прозу, публіцистику, науково-популярні тексти, підручники, офіційно-ділові та наукові матеріали.

Для кожного фрагмента обчислено п'ять базових ознак: середню довжину речення у словах (ASL), середню кількість складів у слові (ASW), кількість багатоскладових слів (P), кількість речень (S) та частку складних слів (PDW). Підрахунок складів виконано на основі голосних української мови з евристикою групування послідовних голосних. Складні слова визначено за словниковою базою системи як лексичні одиниці поза списком простої лексики.

Калібрація виконується методом звичайних найменших квадратів (OLS) [85]. Для кожної з трьох формул побудовано регресійну модель, де залежною змінною є рівень складності фрагмента, а незалежними – відповідні текстові ознаки. Структуру формул збережено з оригінальних моделей, а числові коефіцієнти отримано як оцінки параметрів регресії. Валідація проведена на незалежній підвибірці Brown-UK з 30 фрагментів, стратифікованих за шістьма жанрами (дитяча проза, побутова проза, публіцистика, науково-популярні, офіційно-ділові, наукові), із перевіркою жанрового впорядкування за ранговим коефіцієнтом Спірмена [126].

Позначимо ASL – середня довжина речення у словах, ASW – середня кількість складів у слові. Класичну формулу Flesch Reading Ease [56] наведено у вигляді (2.1).

$$\mathrm{FRE} = 206835 - 1015 \times \mathrm{ASL} - 84{,}6 \times \mathrm{ASW}, \qquad (2.1)$$

де $ASW = \frac{\text{загальна кількість складів у тексті}}{\text{загальна кількість слів}}$ – середня кількість складів у слові,

$ASL = \frac{\text{загальна кількість слів у тексті}}{\text{загальна кількість речень}}$ – середня довжина речення у словах.

Константи 206835, 1015 та 84,6 отримано Р. Флешем у 1948 році методом регресійного аналізу на корпусі англомовних текстів [56]. У системі ІКД показник FRE використовується як довідковий базовий орієнтир, більші значення якого свідчать про простіший текст.

Перекалібрований для української мови показник Flesch-Kincaid має вигляд (2.2):

$$\mathrm{FK_{ukr}} = 0{,}0646 \times \mathrm{ASL} + 0{,}8555 \times \mathrm{ASW} - 1{,}1971 \quad (2.2)$$

Коефіцієнти при ASL та ASW відображають внески синтаксичної протяжності речення і складової структури слова відповідно, а вільний член виконує функцію зсуву шкали. Значення коефіцієнтів визначено методом найменших квадратів (OLS) на калібраційному корпусі (додаток А, табл. 7). Регресійна модель FK_ukr = 0.065·ASL + 0.8533·ASW + (-1.1972) пояснює 99.5 % дисперсії цільової змінної ($R^2 = 0.995$, RMSE = 0.0427). Коефіцієнт при ASL (t = 33.11) та коефіцієнт при ASW (t = 32.619) значущі на рівні $p < 0{,}001$. Менші абсолютні значення коефіцієнтів порівняно з оригінальною формулою Flesch–Kincaid зумовлені відмінною шкалою: FK_ukr відображає складність у нормованому діапазоні, тоді як оригінальна формула використовує шкалу grade level (1–16). Більші значення FK_ukr відповідають вищій складності тексту.

Для формули SMOG введемо позначення: P − кількість багатоскладових слів із трьома і більше складами, S − кількість речень у тексті.

Перекалібрований варіант формули (2.3) зберігає нормування на тридцять речень, запроваджене у класичній моделі Маклафліна для стандартизації підрахунку багатоскладових слів. Константа 30 є структурним елементом оригінальної формули і зберігається без змін. Масштабувальний множник 0.153 та вільний член -0.2137 визначено методом найменших квадратів на калібраційному корпусі (додаток А, табл. 7). Модель пояснює 99.8 % дисперсії цільової змінної ($R^2 = 0.9978$, RMSE = 0.0282). Масштабувальний множник (t = 198.293) та вільний член (t = -21.754) значущі на рівні $p < 0{,}001$.

$$SMOG_{ukr} = 0{,}1528 \times \sqrt{30 \times \frac{P}{S}} - 0{,}2058 \quad (2.3)$$

У формулі D-Level (2.4) PDW позначає частку складних слів, тобто лексичних одиниць поза списком простої лексики, сформованим у межах словникової бази системи.

$$DL_{ukr} = 0{,}0670 \times ASL + 0{,}0231 \times PDW + 0{,}1675, \quad (2.4)$$

де $PDW = \frac{\text{кількість складних слів}}{\text{загальна кількість слів}} \times 100\%$.

Коефіцієнти у (2.4) визначено методом найменших квадратів на

калібраційному корпусі. Регресійна модель DL_ukr = 0.0669·ASL + 0.023·PDW + 0.1744 пояснює 99.7 % дисперсії цільової змінної (R^2 = 0.9968, RMSE = 0.0315). Коефіцієнт при ASL (t = 44.833) відображає внесок синтаксичної протяжності, коефіцієнт при PDW (t = 34.636) – внесок лексичної складності, а вільний член задає базовий рівень шкали (t = 15.244). Усі три оцінки значущі на рівні $p < 0,001$ (додаток А, табл. 7,8).

Наведені коефіцієнти отримано методом найменших квадратів на калібраційному корпусі з 88 фрагментів (R^2 = 0.995 для FK, R^2 = 0.9978 для SMOG, R^2 = 0.9968 для D-Level; усі коефіцієнти значущі на рівні $p < 0,001$) (додаток А, табл. 7). Незалежна валідація на підвибірці Brown-UK (30 фрагментів, 6 жанрів) підтвердила коректність жанрового впорядкування: ранговий коефіцієнт Спірмена між очікуваним рангом складності жанру та середнім балом формули становить $\rho = 1.0$ для FK, $\rho = 1.0$ для SMOG та $\rho = 1.0$ для D-Level. Формули (2.2)–(2.4) виступають робочими параметрами індексу ІКД і не претендують на статус універсальних мовних констант, потребуючи подальшої валідації на ширшій вибірці текстів з експертною розміткою (табл. 2.3).

Таблиця 2.3

Параметри адаптації метрик читабельності для української мови

Метрика	Що потрібно обчислити	Критичні рішення для української мови
Flesch-Kincaid	ASL, ASW, склади	список голосних; правила токенізації; FRE лишається базовим; калібровані коефіцієнти FK_ukr: `0.0646`, `0.8555`, `-1.1971`
SMOG	P (3+ складів), S	правило 3+ складів; робота з короткими текстами; калібровані коефіцієнти `0.1528` і `-0.2058`
D-Level	PDW, ASL	словник українських слів; визначення складних слів; евристики суфіксів; калібровані коефіцієнти `0.0670`, `0.0231`, `0.1675`

Абревіатури та латинські скорочення можуть мати нульову або некоректно оцінену складову структуру в підрахунку складів, однак створювати значний бар'єр розуміння для читача. Числа та змішані записи формально не є словами,

однак суттєво впливають на виконання інструкції користувачем. Дефісні та апострофні форми потребують стабільного розбиття на токени, інакше статистичні показники варіюються між різними реалізаціями алгоритмів. Власні назви можуть бути довгими та багатоскладовими, але водночас добре знайомими читачеві, і навпаки.

Наведені граничні випадки демонструють, що метрики читабельності не охоплюють механізмів локалізації бар'єрів розуміння та їх цілеспрямованого усунення. У системі ІКД такі метрики становлять базовий шар оцінювання, який доповнюється лексичними та синтаксичними факторами, безпосередньо пов'язаними з алгоритмами трансформації тексту.

Сформульовані раніше вимоги до метрики оцінювання когнітивної доступності визначають і вимоги до її математичної структури. Інтерпретованість передбачає, що показник є розкладним на компоненти з прозорим внеском кожного. Обчислюваність передбачає локальний і відтворюваний розрахунок на українських текстах без залежності від нестабільних зовнішніх ресурсів. Придатність для керування трансформаціями передбачає, що показник підтримує локалізацію проблемних фрагментів у тексті та безпосередньо пов'язується з діями редагування, зокрема лексичними замінами, розгортанням скорочень, синтаксичними перебудовами та контролем приросту до і після втручання.

На основі наведених вимог формується індекс когнітивної доступності (ІКД) як багатокомпонентний показник з інтеграцією лексичного, синтаксичного компонентів та компонента читабельності у єдину шкалу оцінки тексту. Подальша формалізація охоплює визначення окремих компонентів, обґрунтування вагових коефіцієнтів у їх композиції, задання шкали значень і порогів інтерпретації, а також опис процедури обчислення на рівні документа та окремого фрагмента.

ІКД вводиться як нормований коефіцієнт, що відображає сукупний рівень когнітивної доступності тексту або його фрагмента і водночас зберігає можливість декомпозиції на причини зниження доступності. Показник

орієнтований не лише на вимірювання поверхневої складності, а й на виявлення джерел когнітивного навантаження, безпосередньо пов'язаних із процедурами автоматизованого спрощення.

Нехай задано текстовий фрагмент, що може бути документом, сторінкою, абзацом або іншою областю в межах вебінтерфейсу. ІКД визначається як зважена композиція трьох компонентів, а саме лексичного, синтаксичного та компонента читабельності. Така структура поєднує два рівні діагностики у межах єдиного показника. Перший рівень є діагностичним і охоплює лексичні та синтаксичні бар'єри, що допускають локалізацію у тексті та зіставлення з відповідними класами трансформацій. Другий рівень є агрегувальним і представлений компонентом читабельності, що відображає загальну поверхневу щільність тексту. Наведена конструкція дозволяє ІКД виконувати функції вимірювання та навігації для подальшого покращення.

Кожен компонент нормалізовано на інтервалі від нуля до одиниці, де одиниця відповідає максимальній доступності у відповідному вимірі, а нуль − мінімальній. Нормалізація здійснюється за схемами, адаптованими до природи кожного компонента. Для лексичного компонента обчислюється частка проблемних токенів кожної категорії відносно загальної кількості, після чого компонент формується як добуток доповнень. Для синтаксичного компонента кожна структурна ознака нормалізується через штрафну функцію із порогом насичення та агрегується зваженою сумою. Сумарний показник ІКД набуває значень на тому ж інтервалі, що забезпечує можливість порівняння результатів між фрагментами тексту. Така нормалізація дозволяє фіксувати стан тексту до і після трансформацій в ітеративному циклі покращення.

У загальному вигляді формула індексу когнітивної доступності має вигляд (2.5):

$$CAI = w_1 \cdot Lex + w_2 \cdot Syn + w_3 \cdot Read \quad (2.5)$$

де *Lex, Syn, Read* − лексичний, синтаксичний і компоненти читабельності відповідно, що набувають значень на інтервалі [0;1];

w_1, w_2, w_3 − вагові коефіцієнти, які задовольняють умову $w_1 + w_2 + w_3 = 1$.

Кожен компонент входить в індекс з власним ваговим коефіцієнтом, що забезпечує контроль його внеску та підтримує пояснення результатів і оптимізацію за окремими аспектами доступності. За відповідного аналізу значень компонентів визначається, який із них є домінантним джерелом складності і які класи трансформацій є адекватними для покращення. У традиційних коефіцієнтах читабельності підсумкове число не підтримує такого переходу до адресної корекції тексту. Нормалізація компонентів до спільного інтервалу забезпечує коректність зважування. Фіксація правил розрахунку, що охоплюють токенізацію, підрахунок складів, словникові ресурси та синтаксичні ознаки, забезпечує відтворюваність результатів між запусками системи.

Обчислення ІКД підтримується як для цілого документа, так і для обмеженого фрагмента у межах поточної зони уваги користувача. Такий режим відповідає вимогам локалізації бар'єрів, коли показник залишається чутливим до змін у конкретному сегменті тексту, а не лише до усередненого стану документа. У наступних підрозділах розглянуто визначення кожного компонента ІКД та набір даних, необхідний для його обчислення.

Лексичний компонент охоплює вплив словникового складу тексту на когнітивне навантаження. Навіть один незрозумілий токен зупиняє читання та активує пошук значення поза текстом, а для цільової аудиторії цей ефект підсилюється обмеженнями робочої пам'яті. Лексичний аналіз у складі ІКД будується як виявлення класів лексичних одиниць, що системн опідвищують когнітивне навантаження.

Вибір категорій ґрунтується на когнітивній релевантності, поширеності у вебконтенті та операційності, а саме можливості автоматичного виявлення та зв'язку зі стратегіям ипокращення. Лексичний компонент охоплює сім категорій бар'єрів (табл. 2.4). Оскільки одна одиниця може належати до кількох категорій, застосовується пріоритизація для запобігання подвійному рахунку. Детекція виконується з морфологічною нормалізацією через порівняння і за лемою, і за словоформою.

Таблиця 2.4.

Категорії лексичних бар'єрів та стратегії обробки

Категорія	**Характер бар'єру**	**Детекція**	**Стратегія спрощення**
Рідковживані слова	Низька частотність в активному словнику, сповільнення читання	Словник з морфологічною нормалізацією	Заміна частотним синонімом або описове розгортання
Терміни	Потреба у фонових знаннях, семантична складність	NER-модель + евристики (суфікси, префікси, чорні списки)	Пояснювальна вставка у дужках, локальний глосарій
Складні слова	Абстрактні іменники, книжна лексика, висока інформаційна щільність	Словник з лематизацією	Лексична заміна, перехід від віддієслівних іменників до дієслівних конструкцій
Застаріла лексика	Потреба у культурно-історичному контексті, навантаження на робочу пам'ять	Словник (obsolete, arch)	Архаїзми → сучасний відповідник; історизми → коротке пояснення
Абревіатури	Додатковий цикл обробки (розпізнавання → відновлення повної форми)	Регулярні вирази + модельний класифікатор + виключення	Розгортання при першій появі, інтерфейсна підказка
Сленг	Нефіксоване значення, залежність від контексту	Словник + евристики + класифікатор	Заміна нейтральним формулюванням, пояснення при цитуванні
Ненормативна лексика	Емоційне відволікання, стрес, зниження концентрації	Словник з лематизацією та нечітким порівнянням (85%)	Нейтралізація, цензурування або вилучення

Лексичний компонент ІКД фіксує типи лексики, які збільшують когнітивні витрати на читання і розуміння тексту та водночас піддаються керованим трансформаціям. Категорії лексичного компонента підібрані як набір операційно виявлених лексичних бар'єрів, характерних для українського вебконтенту.

Для кожної категорії i обчислюється частка проблемних одиниць відносно загальної кількості токенів у тексті (2.6):

$$r_i = min(\frac{n_i}{N}, 1), \quad (2.6)$$

де n_i – кількість виявлених одиниць категорії i, N – загальна кількість

токенів у фрагменті.

Частка r_i визначена на відрізку [0; 1], де нуль відповідає повній відсутності проблемних одиниць категорії i, а одиниця – ситуації, коли всі токени фрагмента належать до цієї категорії. Оператор мінімуму обмежує значення одиницею зверху, що забезпечує коректну нормалізацію за умов перекриття категорій, коли один токен може одночасно потрапляти до кількох категорій бар'єрів.

Лексичний компонент визначається через мультиплікативну агрегацію (2.7):

$$Lex = \prod_{i=1}^{k}(1 - r_i), \quad (2.7)$$

де добуток обчислюється за усіма категоріями лексичних бар'єрів. Вираз $(1 - r_i)$ є доповненням частки бар'єрів до одиниці: одиниця відповідає максимальній доступності категорії i за відсутності проблемних одиниць (r_i = 0), а нуль – повній втраті доступності, коли всі токени належать до категорії бар'єру ($r_i = 1$). Добуток таких множників дає загальну лексичну доступність у діапазоні [0; 1].

Синтаксичний компонент ІКД фіксує структурну організацію речень як джерело когнітивного навантаження. На відміну від лексичних бар'єрів, синтаксичні мають розподілений характер: довжина, вкладеність та порядок слів визначають, скільки елементів читач утримує одночасно. У цільової аудиторії обмеження робочої пам'яті підвищують значення синтаксичної простоти. Синтаксичні ознаки обчислюються на основі дерева залежностей, побудованого NLP-конвеєром, за чотирма групами показників: довжина та варіативність речень, підрядність, глибина дерева залежностей і стабільність канонічного порядку слів.

Синтаксичний компонент нормалізовано так, щоб одиниця відповідала простому і передбачуваному синтаксису, а нуль – структурі з високим навантаженням на читача. Така нормалізація забезпечує безпосередню інтеграцію результату в ІКД та дозволяє використовувати синтаксичний компонент як критерій оптимізації у процесі ітеративного спрощення тексту.

Синтаксичний бал обчислюється як зважена агрегація часткових оцінок за

формулою (2.8):

$$Syntactic = 0.35 \times S_{struct} + 0.35 \times C_{score} + 0.20 \times T_{depth} + 0.10 \times R_{svo} \quad (2.8)$$

де S_{struct} – оцінка довжини та варіативності речень,

C_{score} – оцінка підрядності та вкладеності,

T_{depth} – оцінка глибини дерева залежностей,

R_{svo} – оцінка стабільності канонічної структури речення.

Вагові коефіцієнти формули (2.8) визначено шляхом аналізу чутливості на калібраційному корпусі Brown-UK (88 фрагментів, 11 рівнів складності). Для кожної з 969 конфігурацій, утворених перебором ваг з кроком 0,05 за умови $w_i \geq 0{,}05$, обчислено синтаксичний бал і оцінено ранговий коефіцієнт Спірмена між балом та рівнем складності. Обрана конфігурація (0,35; 0,35; 0,20; 0,10) посіла 2-ге місце з 969 із $|\rho| = 0{,}988$, поступившись оптимуму на $\Delta|\rho| < 0{,}001$. Рівні ваги довжини речень та підрядності відображають їхню порівнянну дискримінативну здатність як двох домінантних джерел синтаксичного навантаження. Глибина дерева залежностей і стабільність порядку слів відіграють роль уточнювальних ознак, які виявляють складність у випадках, коли довжина речення не виходить за прийнятні межі.

Довжина речення є базовою ознакою структурної складності, проте для оцінки когнітивної доступності важить не лише середнє значення, а й варіативність довжин у межах фрагмента. Показник структури речень враховує обидва аспекти одночасно через середню довжину речення в токенах і стандартне відхилення довжин у межах фрагмента.

У реалізації застосовується штрафна функція, нормована на цільові значення TARGET_SENTENCE_LENGTH = 25 і TARGET_SENTENCE_VARIATION = 12. Поріг у 25 слів обрано на основі настанов простої мови [43] та настанов когнітивної доступності W3C [138]: дослідження Американського інституту преси показало, що при середній довжині 14 слів розуміння перевищує 90 %, а після 25 слів різко знижується до 50 % [48]. Ваги $\alpha = 0{,}7$ та $\beta = 0{,}3$ визначено аналогічним аналізом чутливості:

серед 17 конфігурацій з кроком 0,05 обрана комбінація забезпечила найвищий |ρ| = 0,988. Більша вага середньої довжини речення порівняно з варіативністю узгоджується з результатами Мікка [94], який експериментально підтвердив, що абсолютна довжина речення є домінантним предиктором когнітивного навантаження при розумінні тексту. Штраф обчислюється за формулою (2.9):

$$length\_penalty = 0.7 \times min(\frac{avg_{len}}{25}, 1) + 0.3 \times min(\frac{stdev_{len}}{12}, 1), \quad (2.9)$$

де avg_{len} – середня довжина речення у словах,

$stdev_{len}$ – стандартне відхилення довжин речень у межах фрагмента.

Кінцеве значення показника структури речень обчислюється як доповнення штрафу до одиниці за формулою (2.10):

$$SentenceStructure = max(0, 1 - length\_penalty). \quad (2.10)$$

Форма показника дозволяє враховувати як загальну тенденцію до коротких речень, так і відсутність окремих надто довгих конструкцій. Локальні піки довжини провокують зупинку читання, втрату контексту та необхідність повторного читання, що узгоджується з принципами редагування інструктивних текстів у стилі простої мови.

Підрядні речення, означальні звороти, інфінітивні комплементи та інші залежні фрагменти становлять основний механізм синтаксичної вкладеності. Вкладена конструкція відкладає завершення сенсу речення. Читач послідовно зустрічає незавершені синтаксичні зв'язки і утримує їх у пам'яті до появи відповідних завершальних частин. Зі зростанням вкладеності збільшується кількість одночасно відкритих зв'язків, які читач має утримувати. Формалізація цього ефекту реалізована через дві взаємодоповнювальні ознаки.

Перша ознака відображає щільність предикативних одиниць у тексті і обчислюється як частка речень з підрядністю за формулою (2.11):

$$clause\_density = \frac{subordinate\_sentence\_count}{sentence\_count}, \quad (2.11)$$

де subordinate_sentence_count – кількість речень, що містять підрядні частини,

sentence_count – загальна кількість речень у фрагменті.

Друга ознака відображає щільність маркерів підрядності на рівні синтаксичних залежностей і підрядних сполучників. Вона обчислюється за формулою (2.12):

$$marker_{density} = \frac{subordinate_{markers} + connector_{hits}}{total_{tokens}}, \quad (2.12)$$

де subordinate_markers – кількість синтаксичних залежностей підрядного типу,

connector_hits – кількість входжень підрядних сполучників,

total_tokens – загальна кількість токенів у фрагменті.

Маркери підрядності ідентифікуються через мітки синтаксичних залежностей, список типових для української мови підрядних сполучників та сполучникових слів, а також через характерні фразові маркери. Дві щільності об'єднуються у нормований показник складності підрядних одиниць за формулою (2.13):

$$clause\_complexity = min(0{,}6 \times clause\_density + 0{,}4 \times marker\_density, 1) \quad (2.13)$$

Коефіцієнти γ = 0,6 та δ = 0,4 визначено аналізом чутливості на калібраційному корпусі Brown-UK: серед 17 конфігурацій з кроком 0,05 обрана комбінація забезпечила найвищий ранговий коефіцієнт |ρ| = 0,988 (1-ше місце). Більша вага щільності речень із підрядністю порівняно з щільністю окремих складових узгоджується з даними когнітивної лінгвістики, а саме присутність підрядних частин у реченні в цілому більшою мірою впливає на когнітивне навантаження, ніж кількість окремих сполучникових складових [60, 61]. Підсумковий показник оцінки підрядних одиниць обчислюється як доповнення складності до одиниці за формулою (2.14):

$$ClauseScore = max(0, 1 - clause\_complexity) \quad (2.14)$$

Показник знижує синтаксичний бал не лише у випадках, коли підрядність зосереджена в одному реченні, а й коли підрядні конструкції розподілені у багатьох реченнях фрагмента. Наведена форма дозволяє врахувати як локальні синтаксичні труднощі у межах окремого речення, так і стабільну схильність

тексту до вкладених конструкцій.

Показник підрядності не вичерпує усієї складності синтаксичної структури. Формально коротке речення буває складним для сприйняття через вкладені уточнення, розірвані залежності та багаторівневі ланцюжки модифікаторів. Для охоплення цього класу випадків у ІКД введено показник середньої глибини дерева залежностей. Для кожного речення будується дерево залежностей, у якому обчислюється середня глибина шляху від кореня до листка. Із зростанням глибини дерева збільшується довжина ланцюгів залежностей, які читач утримує для відновлення смислових ролей.

Порогове значення MAX_TREE_DEPTH визначено аналізом чутливості на калібраційному корпусі Brown-UK. Серед 16 значень у діапазоні 5-20 значення 12 посіло 4-те місце з $|\rho| = 0{,}988$ ($\Delta|\rho| < 0{,}001$ від оптимуму). Значення 12 обрано як компроміс між здатністю до розрізнення складності та стійкістю до жанрової варіації. Нижчі пороги (8-10) підвищують чутливість до глибоких конструкцій з надмірним зниженням оцінки наукових та юридичних текстів з природно глибшою структурою залежностей. Показник глибини дерева обчислюється за формулою (2.15).

$$T_{depth} = max(0, 1 - min(\frac{avg\_tree\_depth}{12}, 1)), \quad (2.15),$$

де avg_tree_depth – середня глибина дерева залежностей для речень у межах фрагмента.

Показник доповнює попередні ознаки структурної складності у випадках, коли морфологія української мови дозволяє будувати конструкції з довгими відступами між узгодженими елементами, а довжина речення у токенах залишається в межах норми.

Українська мова допускає вільний порядок слів, проте для оцінки когнітивної доступності визначальною є близькість структури речення до канонічного шаблону з високою передбачуваністю. У ІКД введено показник співвідношення порядку слів, що визначає частку речень, у яких підмет передує присудкові, а додаток слідує за ним. Такий порядок відповідає базовому шаблону «підмет – присудок – додаток» (SVO), у якому граматичні ролі встановлюються

читачем з найменшими витратами ресурсів на інтерпретацію. Показник фіксує не нормативність мови, а ступінь передбачуваності синтаксичної структури. У вебконтенті стабільний порядок слів знижує ризик хибного встановлення граматичних ролей і полегшує сприйняття синтаксичних зв'язків читачем.

Синтаксичний компонент ІКД пов'язує діагностику з керованим редагуванням тексту через обчислювану форму кожної ознаки та її зіставлення з конкретним типом синтаксичних трансформацій для визначення наступної дії алгоритму (табл. 2.5).

Таблиця 2.5

Синтаксичні індикатори бар'єрів та керовані трансформації

Індикатор синтаксичної складності	Типовий бар'єр для читача	Стратегія спрощення у системі
висока середня довжина речень або значна варіативність довжин	утримання надмірного обсягу контексту до завершення смислу	розбиття довгих речень на 2–3 коротші з окремими твердженнями
висока частка речень із підрядністю або висока густина маркерів підрядності	множинні умови, причини та уточнення, що відкладають основну дію	виокремлення причинно-наслідкових, умовних і часових фрагментів у самостійні речення
значна глибина дерев залежностей	вкладені означальні конструкції з розірва-ними залежностями між словами	спрощення вкладених фрагментів, зокрема означальних підрядних і вставних зворотів, сегментація за пунктуаційними межами
низька частка канонічного порядку слів	непередбачуваний порядок граматичних ролей, підвищений ризик хибної інтерпретації	нормалізація порядку слів у випадках, коли зміна не впливає на смислові акценти

Третім компонентом ІКД є **показник читабельності**, що оцінює поверхневу щільність тексту. Компонент відображає, наскільки текст сприяє послідовному сприйняттю без виснаження уваги, і виконує роль агрегувального індикатора загальної поверхневої складності.

У реалізації компонента читабельності обчислюються два нормованих показники складності (2.16, 2.17). Складність слова визначається за формулою:

$$word\_complexity = min(\frac{avg\_word\_length}{8}, 1), \quad (2.16)$$

де avg_word_length – середня довжина слова в символах.

Складність речення визначається за формулою:

$$sentence\_complexity = min(\frac{avg\_sentence\_length}{20}, 1), \quad (2.17)$$

де avg_sentence_length – середня довжина речення у словах.

Поріг 20 слів для речення обрано відповідно до настанов простої мови [43] та результатів Американського інституту преси, за якими розуміння різко знижується після 20 слів [48]. Поріг 8 символів для слова враховує морфологічну специфіку української мови як мови з розвиненою системою афіксів: середня довжина слова в загальних українських текстах становить 5–6 символів, тоді як у науковому та офіційно-діловому стилях зростає до 7–8 символів через термінологічні утворення та складні словоформи [120]. Значення 8 відповідає верхній межі типового діапазону і забезпечує насичення оцінки для текстів із максимальною лексичною щільністю. Обмеження одиницею зверху запобігає впливу окремих надмірно довгих слів або речень на оцінку всього фрагмента і забезпечує стійкість компонента при обробці вебтекстів.

Бал читабельності обчислюється як добуток двох доповнювальних показників простоти за формулою (2.18):

$$Readability = (1 - word\_complexity) \times (1 - sentence\_complexity), \quad (2.18)$$

Мультиплікативна форма забезпечує високе значення показника лише у випадках одночасно помірної довжини слів і речень. Якщо один з факторів досягає критичних значень, показник знижується незалежно від стану іншого, що запобігає хибному зарахуванню тексту як читабельного при локальному перевантаженні одного з рівнів. Нормалізація показника на інтервалі [0; 1] дозволяє використовувати його як компонент ІКД без додаткових перетворень.

Додатково у системі визначено класичні метрики читабельності, адаптовані для української мови (FK_ukr, SMOG_ukr, DL_ukr), для зіставлення з ІКД та верифікації трансформацій. Процедуру калібрування наведено у Додатку А таблиця 7.

Після формалізації трьох компонентів ІКД виникає задача вибору вагових коефіцієнтів для їх агрегування в інтегральний показник. Ваги компонентів ІКД

визначено методом безпосереднього оцінювання, який передбачає безпосереднє задання експертами відносної значущості кожного компонента без побудови матриць попарних порівнянь. Такий підхід є доцільним для задач з невеликою кількістю критеріїв та чітко визначеною змістовою інтерпретацією кожного з них, що відповідає структурі ІКД.

Для проведення безпосереднього оцінювання сформовано групу з 10 фахівців з чотирьох профільних напрямів предметної області дослідження – 3 особи з лінгвістики та психолінгвістики, 3 особи з вебдоступності та інклюзивного UX-дизайну, 2 особи з корекційної педагогіки та логопедії, 2 особи з інженерії обробки природної мови.

Кожен експерт оцінив відносну важливість трьох компонентів ІКД за 10-бальною шкалою, де вищий бал відповідає більшій важливості компонента для забезпечення когнітивної доступності тексту. Десятибальна шкала обрана як стандартний інструмент методу безпосереднього оцінювання, що забезпечує достатню роздільну здатність для диференціації трьох компонентів [109]. Для зниження суб'єктивності оцінювання бали згруповано у п'ять парних рівнів (табл. 2.6).

Таблиця 2.6

Шкала оцінювання відносної важливості компонентів ІКД

Бал	Рівень важливості	Інтерпретація
1–2	Не важливо	Компонент не впливає на когнітивну доступність тексту
3–4	Мало важливо	Компонент має незначний вплив на доступність
5–6	Помірно важливо	Компонент помітно впливає на доступність
7–8	Важливо	Компонент суттєво впливає на когнітивну доступність
9–10	Критично важливо	Компонент є визначальним для забезпечення доступності

Експерти оцінювали компоненти незалежно один від одного, без попереднього співвіднесення балів між собою; виставлення однакових балів кільком компонентам допускалося. Зведені оцінки експертів подано в таблиці 2.7.

Таблиця 2.7

Експертні оцінки компонентів ІКД

ID	b_1 (C_1)	b_2 (C_2)	b_3 (C_3)	Сума
E1	9	7	5	21
E2	8	8	6	22
E3	7	9	5	21
E4	8	7	6	21
E5	7	6	5	18
E6	8	7	5	20
E7	9	7	4	20
E8	10	8	6	24
E9	7	7	5	19
E10	8	6	4	18

Для усунення впливу індивідуального масштабу оцінювання бали кожного експерта нормалізовано за формулою (2.19) діленням на суму його оцінок.

$$a_{kj} = \frac{b_{kj}}{b_{k1}+b_{k2}+b_{k3}}, \quad (2.19)$$

де b_{kj} – оцінка k-го експерта для j-го компонента, $j \in \{1, 2, 3\}$.

Така нормалізація компенсує систематичні відмінності в експертних шкалах: одні експерти схильні виставляти стабільно вищі бали, інші – стабільно нижчі, проте важливими для агрегування є лише відносні пропорції між компонентами в межах оцінок одного експерта.

Розглянемо процедуру нормалізації для експерта E1, що виставив оцінки $b_{1,1} = 9$, $b_{1,2} = 7$, $b_{1,3} = 5$ на прикладі. Підставлення значень у формулу (2.19) дає такі нормалізовані оцінки:

$$a_{1,1} = 9 / (9 + 7 + 5) = 9 / 21 \approx 0{,}4286;$$

$$a_{1,2} = 7 / 21 \approx 0{,}3333;$$

$$a_{1,3} = 5 / 21 \approx 0{,}2381.$$

Сума нормалізованих оцінок дорівнює 0,4286 + 0,3333 + 0,2381 = 1,0000, що підтверджує коректність нормалізації. Аналогічна процедура застосована до оцінок усіх експертів.

Групові ваги компонентів wj обчислено як середнє арифметичне нормалізованих оцінок усіх K = 10 експертів за формулою (2.20). Розрахунок для

лексичного компонента:

$$w_1 = (0{,}4286 + 0{,}3636 + 0{,}3333 + 0{,}3810 + 0{,}3889 + 0{,}4000 + 0{,}4500 + 0{,}4167 + 0{,}3684 + 0{,}4444) / 10 \approx 0{,}3975$$

Аналогічно отримано ваги синтаксичного компонента ($w_2 \approx 0{,}3527$) і компонента читабельності ($w_3 \approx 0{,}2498$). Сума групових ваг $w_1 + w_2 + w_3 = 1{,}0000$ підтверджує нормованість підсумкових коефіцієнтів. Зведені нормалізовані оцінки за всіма експертами та підсумкові групові ваги наведено в таблиці 2.8.

Таблиця 2.8

Нормалізовані експертні оцінки та групові ваги компонентів ІКД

ID	a_1 (Лексичний)	a_2 (Синтаксичний)	a_3 (Читабельність)	Перевірка суми
E1	0,4286	0,3333	0,2381	1,0000
E2	0,3636	0,3636	0,2727	1,0000
E3	0,3333	0,4286	0,2381	1,0000
E4	0,3810	0,3333	0,2857	1,0000
E5	0,3889	0,3333	0,2778	1,0000
E6	0,4000	0,3500	0,2500	1,0000
E7	0,4500	0,3500	0,2000	1,0000
E8	0,4167	0,3333	0,2500	1,0000
E9	0,3684	0,3684	0,2632	1,0000
E10	0,4444	0,3333	0,2222	1,0000
w_j	0,3975	0,3527	0,2498	1,0000

Групові ваги округлено до сотих часток, оскільки при 10 експертах і цілочисельній шкалі похибка вибірки перевищує різницю між точним і округленим значенням, а надлишкова точність створює ілюзію надійності, яку метод не забезпечує. Округлені значення зберігають нормованість (сума дорівнює 1,00) і дають підсумкові вагові коефіцієнти $w_1 = 0{,}40$ для лексичного компонента, $w_2 = 0{,}35$ для синтаксичного компонента та $w_3 = 0{,}25$ для компонента читабельності. Розподіл ваг відображає узагальнену експертну позицію щодо відносного внеску компонентів у когнітивну доступність вебтексту. Найвищу вагу отримав лексичний компонент як джерело первинних зупинок розуміння з найширшим репертуаром адресних трансформацій. Синтаксичний компонент має другу за величиною вагу через вплив вкладеності, підрядності й нестабільного порядку слів на робочу пам'ять читача. Менша вага

компонента читабельності відповідає його поверхневому, узагальнювальному характеру: класичні індекси читабельності спираються на формальні параметри довжини слів і речень, тоді як лексичний і синтаксичний компоненти діагностують конкретні бар'єри розуміння, що піддаються адресним трансформаціям. Підсумкові ваги компонентів ІКД зведено в таблиці 2.9.

Таблиця 2.9

Визначення вагових коефіцієнтів ІКД методом безпосереднього оцінювання

Компонент ІКД	Нормована вага	Інтерпретація ролі у моделі
Лексичний (C_1)	0,40	Первинний бар'єр розуміння, орієнтований на адресні словникові трансформації
Синтаксичний (C_2)	0,35	Визначає навантаження на послідовне розуміння, робочу пам'ять і сприйняття вкладених конструкцій
Читабельність (C_3)	0,25	Виконує узагальнювальну функцію оцінки загальної щільності тексту

Отже, експертна процедура підтвердила доцільність конфігурації, у якій лексичний і синтаксичний компоненти мають вищий пріоритет порівняно з поверхневою читабельністю, що узгоджується з цільовою логікою ІКД як інструмента локалізації та подальшого усунення когнітивних бар'єрів.

Застосування ІКД як діагностичного інструмента потребує шкали інтерпретації, що переводить числове значення індексу у якісний рівень доступності тексту і підтримує прийняття рішень на етапах аналізу, правки та повторної оцінки (табл. 2.10).

Таблиця 2.10

Шкала інтерпретації значень ІКД

Значення ІКД	Інтерпретація
$0.80 \leq$ ІКД ≤ 1.00	Відмінна доступність
$0.60 \leq$ ІКД < 0.80	Хороша доступність
$0.40 \leq$ ІКД < 0.60	Помірна доступність
$0.20 \leq$ ІКД < 0.40	Низька доступність
$0.00 \leq$ ІКД < 0.20	Відсутня доступність

Шкала застосовується для пріоритизації трансформацій: фрагменти з низькими значеннями обробляються першочергово. Категоризація є інтерпретаційним шаром над неперервним значенням, тому рішення про трансформацію враховує також профіль компонентів та тип тексту.

Інтерпретація значень ІКД залежить від довжини аналізованого фрагмента. Для коротких текстів часткові метрики компонентів мають підвищену дисперсію, оскільки одне складне слово або одна вкладена конструкція призводить до помітної зміни значень компонентів, а разом з ними і інтегрального індексу. Таке коливання є властивістю статистичної моделі на малих вибірках, а не помилкою ІКД, однак воно вимагає уважної інтерпретації. Для коротких фрагментів застосовується режим локальної діагностики з ідентифікацією конкретного бар'єра, а для абзаців та інструкцій – режим узагальненої категоризації. Значення ІКД, близькі до межі інтервалу категоризації, інтерпретуються як зона невизначеності, у якій рішення про трансформацію спирається на компонентний аналіз, а не на категорію.

Інтерпретація значень ІКД вимагає уникати оптимізації показника без фактичного усунення бар'єрів. Зростання значення індексу має виступати наслідком зміни конкретних характеристик тексту, а не модифікацій, що покращують статистику без відповідного впливу на когнітивну доступність. Аналіз загального значення супроводжується розбором компонентів та локалізацією проблемних позицій у тексті, що дає змогу ідентифікувати джерела покращення і проконтролювати збереження сенсу.

ІКД є формалізованим показником когнітивної доступності, і його застосування має низку обмежень, що визначають межі коректної інтерпретації результатів. Значення ІКД залежить від якості попередньої лінгвістичної обробки тексту. Помилки токенізації, лематизації або синтаксичного розбору залежностей спотворюють часткові ознаки, що обчислюються для лексичного та синтаксичного компонентів. Компонентна структура індексу послаблює цей вплив через поєднання різних джерел інформації, а базовий показник читабельності обчислюється на простих статистичних ознаках. У випадках з

нестандартною пунктуацією, змішаними записами чи фрагментованими повідомленнями точність показника знижується, що враховується під час інтерпретації результатів.

Інтерпретація значень ІКД вимагає уникати оптимізації показника без фактичного усунення бар'єрів. Зростання значення індексу має виступати наслідком зміни конкретних характеристик тексту, а не модифікацій, що покращують статистику без відповідного впливу на когнітивну доступність.

Значення ІКД залежить від якості попередньої лінгвістичної обробки тексту. Помилки токенізації, лематизації або синтаксичного розбору залежностей спотворюють часткові ознаки, що обчислюються для лексичного та синтаксичного компонентів. У випадках з нестандартною пунктуацією, змішаними записами чи фрагментованими повідомленнями точність показника знижується, що враховується під час інтерпретації результатів.

Безпосереднє оцінювання, проведене у межах дослідження, спрямоване переважно на визначення вагових коефіцієнтів компонентів і не замінює повної валідації ІКД як психометричного інструмента, що потребує цільових експериментів з вимірюванням часу читання та суб'єктивного розуміння тексту на репрезентативному корпусі.

Метод безпосереднього оцінювання забезпечив визначення вагових коефіцієнтів компонентів ІКД ($w_1 = 0{,}40$; $w_2 = 0{,}35$; $w_3 = 0{,}25$) на основі узгоджених оцінок групи з 10 фахівців. Узагальнену структуру обчислення та агрегування компонентів ІКД подано на рисунку 2.1.

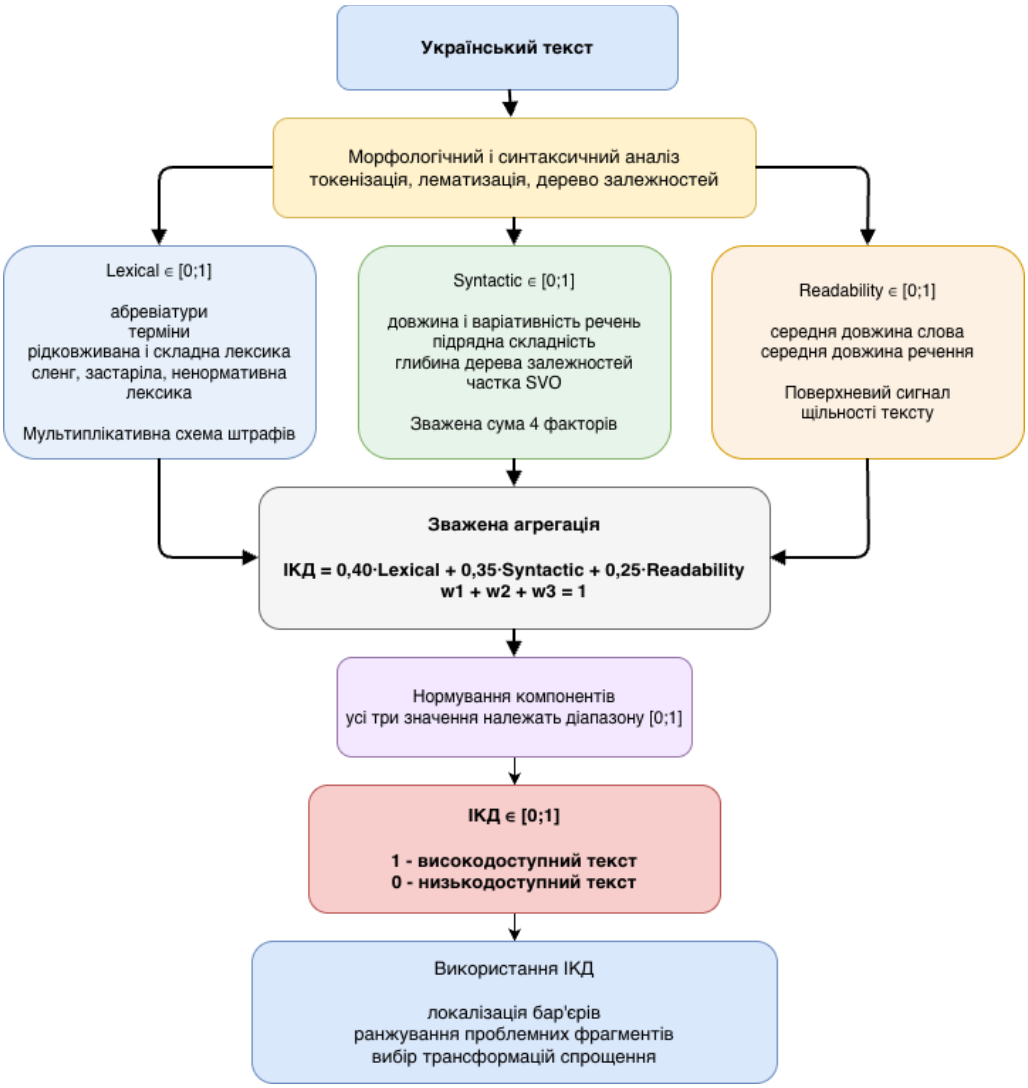


Рис. 2.1. Структура обчислення та агрегування компонентів індексу когнітивної доступності ІКД.

2.2. Метод визначення стабільної емоції користувача

Одна й та сама трансформація тексту має різний ефект для користувача у спокійному стані та у стані когнітивного перевантаження або емоційної дезорієнтації. Підсистема спрощення тексту має враховувати не лише лінгвістичні властивості вебконтенту, а й контекст його сприйняття, що змінюється у часі й потребує автоматизованого вимірювання.

Стабілізований емоційний показник виступає зовнішнім керуючим параметром методу адаптивного спрощення тексту. Вимірювання емоційного стану реалізовано в окремому модулі розпізнавання емоцій з обробкою відеопотоку з камери користувача. Функції модуля охоплюють детекцію

обличчя, класифікацію емоції на рівні окремого кадру та часову стабілізацію результату з урахуванням впевненості моделі класифікації.

Емоційний стан є одним із чинників, що визначає якість взаємодії користувача з інформаційним середовищем. Він впливає на здатність утримувати увагу, підтримувати послідовність дій, відновлювати контекст після переривання, а також на толерантність до невизначеності й щільності поданого тексту. У цільової аудиторії цей вплив проявляється сильніше. Навіть за незмінної складності тексту зміна стану призводить до зростання кількості помилок читання, пропуску смислових фрагментів, передчасного завершення взаємодії та некоректної інтерпретації інструкцій або повідомлень.

Джерелом у запропонованому методі є емоційний стан та його агрегована оцінка, що обчислюється за відеопотоком з камери користувача. Вибір емоційного стану як індикатора, зумовлений його зв'язком з поточним рівнем напруження та дезорієнтації читача, а відеопотік забезпечує отримання даних у реальному часі без модифікації тексту та без збору додаткових текстових ознак, що можуть містити персональні дані.

У роботі вводиться поняття стійкої емоції як класифікаційного рішення з властивостями інерційності та відтворюваності. Стійка емоція обчислюється як стабілізоване рішення за агрегованою впевненістю моделі та заданою кількістю послідовних спостережень у часовому вікні. Наведена вимога до стабільності дозволяє включити модуль розпізнавання емоцій до контуру керування програмної системи як керуючий компонент.

Зв'язок модуля розпізнавання емоцій з підсистемою адаптивного спрощення реалізовано через безпосередній вплив керуючих параметрів на вибір стратегії трансформацій. Стійка емоція та похідний агрегований стан визначають режим роботи підсистеми спрощення. У нейтральному стані користувача підсистема спрощення робить підготовку точкових лексичних замін та локальних синтаксичних редагувань. У стані когнітивного навантаження або вираженої емоційної дезорієнтації застосовуються основні зміни, що підвищують розуміння. У стані когнітивного перевантаження або емоційної

дезорієнтації користувач має можливість зупинити роботу алгоритму. Такий зв'язок забезпечує динамічну адаптацію процесу спрощення до поточних умов сприйняття тексту користувачем.

Вимоги до методу та обґрунтування відмови від миттєвої емоції. Метод забезпечення доступності тексту реалізує механізм прийняття рішень з впливом на спосіб подання вебконтенту з потребою у вхідному показнику стану користувача з властивостями стабільності та керованої затримки. Використання миттєвої емоції як вхідного показника призводить до високочастотних коливань значень показника та регулярних перемикань режиму спрощення. У цільової аудиторії такі перемикання ускладнюють формування стійкого уявлення про інтерфейс та підвищують когнітивні витрати на повернення до перерваного читання.

Причини нестабільності оцінки емоції на рівні кадру є систематичними і не зводяться до помилок окремої моделі класифікації. До них належать зміни освітлення, повороти голови, часткові перекриття обличчя, різна відстань до камери, випадкові артефакти детекції обличчя, а також короткочасні мімічні реакції, що не відповідають тривалому стану користувача під час читання. Навіть за коректного розподілу ймовірностей категорій емоцій у кадрі вибір домінантної мітки перетворює цей розподіл на дискретне значення, у якому коливання впевненості моделі змінюють результат класифікації. Миттєва емоція не задовольняє вимогу відтворюваності та не може використовуватися як змінна керування у методі адаптивного спрощення.

Параметр має бути інтерпретованим, що дозволяє аналізувати причини входження системи у певний режим. Стійкість у часі забезпечується обмеженою частотою перемикань та інерційністю значення. Параметризованість дозволяє калібрувати алгоритм під різні умови використання, апаратні характеристики й типи вебконтенту без модифікації коду. Працездатність за відсутності коректного кадру зберігається у випадках, коли обличчя не визначається або впевненість моделі класифікації є низькою. Придатність до інтеграції забезпечується чіткими інтерфейсами, узгодженою семантикою й підтримкою

збору телеметричних даних для перевірки властивостей алгоритму.

Поняття стійкої емоції вводиться як стабілізоване у часі рішення, що підтверджується агрегованою впевненістю моделі та накопиченою кількістю послідовних спостережень. Реалізація стійкої емоції відповідає переходу від миттєвого класифікатора до фільтрованого процесу оцінювання, що виконує згладжування часового ряду класифікаційних рішень, відкидає окремі спостереження з низькою впевненістю та формує підсумкове рішення після виконання умов активації. Такий підхід узгоджує часовий масштаб вимірювання з часовим масштабом керування у підсистемі адаптивного спрощення.

Метод визначення стійкої емоції потребує двох каналів спостереження з різною часовою динамікою. Короткостроковий канал реагує на зміни стану з потрібною чутливістю, але містить значну частку шумових коливань. Довгостроковий канал підтверджує тенденцію значень параметра і знижує ймовірність хибних перемикань режиму. Комбінація двох каналів забезпечує компроміс між оперативністю реакції та стабільністю результату і дозволяє запровадити правило підтвердженого переходу для негативних станів.

Виділення модуля розпізнавання емоцій у самостійний компонент відповідає декомпозиції системи за зонами відповідальності. Підсистема комп'ютерного зору опрацьовує апаратні залежності та обмеження реального часу, а підсистема забезпечення доступності тексту опрацьовує лінгвістичні трансформації та правила зупинки. Виділення емоційного модуля в окремий процес стандартизує формат вихідних даних, забезпечує параметризацію без модифікації основного алгоритму спрощення та підтримує накопичення телеметрії для експериментальної перевірки стабільності й чутливості алгоритму. Метод повертає узгоджену мітку стійкої емоції разом з проміжними результатами для короткострокового і довгострокового каналів спостереження, що забезпечує інтерпретованість значень та підтримує подальше калібрування.

На рівні обробки кадру виконується детекція обличчя за допомогою каскадного класифікатора Хаара [132]. Виділена область інтересу подається на класифікацію для визначення домінантної емоції. Розподіл ймовірностей

використовується для оцінки впевненості моделі та подальшого накопичення значень у каналах спостереження. Введено фільтр мінімальної площі області інтересу, що знижує кількість помилкових спрацьовувань детектора у випадках з невеликим або частково видимим обличчям у кадрі.

Підготовка кадру для класифікатора виконується через послідовну нормалізацію зображення для зменшення впливу варіацій освітлення та кольорової зйомки. Кадр перетворюється у одноканальне представлення у відтінках сірого, що усуває залежність результату від кольорової складової зображення.

Додатковим джерелом похибки є відхилення обличчя від фронтального положення. Повороти голови, відведення погляду від камери або часткове перекриття обличчя призводять до некоректного або неповного виділення області інтересу. Стабілізація у таких випадках реалізована через контрольований пропуск кадрів. За відсутності детекції обличчя або при недостатній площі області інтересу поточний кадр не враховується у формуванні рішення. Метод не приймає рішення за відсутності валідного спостереження, а переводить оцінювання у режим затухання попередніх значень впевненості з можливим переходом до стану невизначеності.

Геометрична корекція кожного відхилення положення обличчя не виконується, оскільки такі перетворення створюють артефакти зображення та підвищують обчислювальну вартість обробки кадру. Стратегія методу поєднує консервативні критерії валідності області інтересу з часовою стабілізацією рішень. Метод орієнтується на підтверджені у часі стани та знижує ризик хибних перемикань, спричинених короткочасними відхиленнями погляду або пози користувача.

Модель комп’ютерного зору має забезпечувати працездатність без ручного донавчання у межах реалізації методу та якість оцінок, прийнятну для подальшої часової стабілізації. У методі реалізовано механізм поступової деградації функціональності для випадків, коли поточна версія бібліотеки не підтримує явний вибір моделі під час класифікації. У такому випадку метод автоматично

переходить на режим за замовчуванням, що зберігає працездатність системи без модифікації зовнішніх інтерфейсів.

Вибір значень за замовчуванням підкріплено емпіричними спостереженнями телеметричних даних. Накопичені часові ряди для різних моделей класифікації дозволяють порівнювати стабільність значень, тривалість входу у стійкий режим та частоту перемикань за однакових сценаріїв взаємодії з користувачем. Зібрані артефакти використовуються для калібрування параметрів алгоритму стабілізації та для перевірки властивостей стійкої емоції на реальних послідовностях кадрів.

Часова стабілізація та формалізація стійкої емоції. Підсистема адаптивного спрощення тексту реалізована як система керування з вхідним параметром стану користувача та виходом у вигляді трансформацій тексту. Перемикання режимів спрощення супроводжується зміною подання тексту, появою додаткових пояснень та переформулюваннями, що потребує повторного встановлення зв'язку з контекстом з боку читача. Ці витрати визначають вимогу до вхідного параметра щодо стабільності у часі та узгодженого темпу реакції системи. Поняття стійкої емоції формалізується як часовий процес обчислення керуючого параметра, що знижує дисперсію миттєвих оцінок та забезпечує підтверджені у часі переходи між станами.

Миттєва емоційна оцінка на рівні кадру не задовольняє вимогам стабільності керуючого параметра. Варіативність оцінки виникає з двох причин. Статистична причина пов'язана з коливаннями ймовірностей між близькими категоріями емоцій без відповідності фактичним змінам стану читача у короткому інтервалі. Технічна причина охоплює пропуски детекції обличчя, артефакти окремих кадрів та неоднорідність якості відеопотоку. Сукупність цих факторів формує короткочасні імпульси у часовому ряді миттєвих оцінок, які не повинні безпосередньо переключати режим роботи підсистеми спрощення.

Стійка емоція формалізується як керуючий параметр, що змінює значення лише за наявності накопичених у часі підтверджених свідчень про зміну стану читача. Часова інерційність значення параметра запобігає надлишковим

перемиканням режиму роботи підсистеми спрощення. Стійка емоція має задовольняти вимозі відтворюваності, за якої повторення подібних сценаріїв взаємодії з користувачем формує порівнювані значення керуючого параметра через фіксовані правила згладжування часового ряду та активації переходу між станами.

Експоненційне затухання обрано як основний механізм часової стабілізації керуючого параметра. Підхід забезпечує постійну обчислювальну складність обробки кожного кадру, не потребує збереження вікна попередніх кадрів та коректно опрацьовує інтервали відсутності валідних спостережень. На відміну від жорсткого ковзного вікна, експоненційне затухання формує плавний перехід між значеннями керуючого параметра у часі без штучних меж між інтервалами спостереження, що відповідає вимогам до адаптивних інтерфейсів. Підхід також підтримує явні умови активації та деактивації переходу між станами через пороги впевненості та мінімальну кількість послідовних спостережень, що забезпечує параметризованість поведінки алгоритму та придатність до калібрування.

Основним елементом методу стабілізації виступає агрегатор міток з інерційним накопиченням внесків від кадрів через експоненційне затухання. Агрегатор зберігає поточне накопичене значення для кожної мітки та переводить часовий ряд миттєвих оцінок у стабілізоване значення керуючого параметра.

Часова динаміка агрегатора описується через функцію стану $s_t(l)$, що накопичує внески $c_t(l)$ для кожної мітки l на кроці t, де значення $c_t(l) \in [0, 1]$ відповідає нормалізованій впевненості моделі для поточного кадру. Оновлення стану виконується у два етапи. На першому етапі до попереднього значення стану застосовується коефіцієнт затухання d, що моделює забування попередніх спостережень. На другому етапі до затухлого стану додається внесок поточного кадру, масштабований множником (1 − d), що забезпечує нормалізований вплив нового спостереження та обмежує швидкість накопичення значень. Об'єднана формула оновлення стану має вигляд (2.20):

$$s_t(l) = d \times s_{t-1}(l) + (1 - d) \times c_t(l), \qquad (2.20)$$

де $d \in (0,1)$ – коефіцієнт затухання,

$c_t(l) \in [0,1]$ – нормалізована впевненість моделі для поточного кадру.

Алгоритм передбачає механізм очищення стану від міток з історично застарілими внесками через нижній поріг θ_min. Якщо значення стану $s_t(l)$ після затухання опускається нижче θ_min, відповідна мітка видаляється зі словника агрегатора. Очищення стану запобігає тривалому впливу рідкісних або випадкових міток на вибір домінантного класу після одиничного спостереження. Правило очищення визначається за формулою (2.21):

$$s'_t(l) = \begin{cases} s_t(l), \text{якщо } s_t(l) \geq \theta_{min} \\ 0, \text{якщо } s_t(l) < \theta_{min} \end{cases}, \quad (2.21)$$

де θ_{min} – нижній поріг видалення застарілих міток.

Активація стійкої мітки виконується після досягнення мінімальної кількості валідних спостережень n_{min}. Після цієї умови агрегатор вибирає мітку з максимальним накопиченим значенням стану та перевіряє її на поріг активації θ_{active}. У випадку, коли значення стану обраної мітки нижче порогу, агрегатор повертає результат як невизначений, що інтерпретується підсистемою спрощення як відсутність достатньої впевненості для зміни режиму. Така схема вводить керовану затримку між появою стійкої тенденції у часовому ряді міток та її фіксацією як активного значення керуючого параметра, що знижує кількість помилкових перемикань. Правило активації записується у вигляді (2.22):

$$e^* = \begin{cases} \arg max_l \ s_t(l), \text{якщо } n_t \geq n_{min} \text{ та } s_t(e^*) \geq \theta_{active} \\ \emptyset, \text{інакше} \end{cases} \quad (2.22)$$

де e* – поточне значення стійкої мітки,

θ_{active} – поріг активації,

n_t – лічильник валідних кадрів у поточному вікні спостереження,

n_{min} – мінімальна кількість спостережень для активації.

За відсутності валідної області обличчя у кадрі або несформованих внесків агрегатор переходить у режим тільки-затухання, у якому лічильник валідних спостережень не збільшується, проте до поточного стану застосовується затухання та повторна перевірка умов активації. Такий режим коректно

опрацьовує пропуски у відеопотоці та забезпечує поступове повернення агрегатора до невизначеного стану у випадках зникнення або дестабілізації джерела даних.

Метод агрегації ймовірностей міток емоцій враховує можливу наявність кількох облич у кадрі. Ймовірність для кожної мітки агрегується з верхньою межею для збереження однакового діапазону незалежно від кількості детектованих облич у кадрі. Паралельно домінантна миттєва емоція кадру фіксується як мітка з найбільшим значенням нормалізованої впевненості та записується у телеметрію як первинне спостереження.

Метод використовує два канали стабілізації з різними наборами параметрів для узгодження вимог чутливості та стабільності. Короткостроковий канал параметризовано через коефіцієнт затухання, поріг активації та мінімальну кількість спостережень, а довгостроковий – через аналогічний набір параметрів із більшою інерційністю. Підсумкова стійка емоція визначається як мітка довгострокового каналу, а за її відсутності використовується мітка короткострокового каналу. Така конфігурація надає пріоритет стабільності за умови достатнього накопичення підтверджень і зберігає здатність до оперативної реакції на різку зміну стану користувача.

Перехід від емоції до агрегованого стану та правило консенсусу.

Для задачі когнітивної доступності важливим є не стільки відтінок емоції, скільки наявність ознак ускладнення сприйняття. Тому у сервісі вводиться агрегований стан, який узагальнює емоційний простір до трьох режимів, що відображають умови взаємодії. Такими режимами є спокійний, дезорієнтований та перевантажений.

Перехід від емоції до стану задається ваговим відображенням, що визначає, як окремі емоції вносять вклад у ймовірність станів. Наприклад, нейтральна та позитивна емоції повністю підтримують спокійний стан, тоді як негативні емоції розподіляють вклад між сплутаністю та перевантаженням. За відсутності визначеного відображення для мітки емоції використовується значення за замовчуванням зі зниженням ризику хибної інтерпретації для

рідкісних або некоректно визначених категорій емоцій.

Накопичення станів виконується тим самим механізмом часової стабілізації, що і для емоцій. Для кожного кадру будується внесок у простір станів, після чого ці внески подаються на агрегатор станів. У результаті формується стійкий стан у двох часових каналах, що дозволяє застосовувати однакові критерії стабільності як до емоції, так і до стану (2.23).

$$S_t(state) = \sum_e w(e, state) \times s_t(e), \quad (2.23)$$

де w(e, state) – вагові коефіцієнти відображення емоційного стану.

Для зниження ризику хибної адаптації режиму спрощення алгоритм застосовує правило негативного консенсусу. Правило формалізує умову, за якої короткостроковий канал фіксує негативний стан, а ця оцінка підтверджується довгостроковим каналом або довгостроковий канал перебуває у нейтральному режимі спостереження. Критерій враховує можливу неактивацію довгострокового каналу на момент оцінювання та одночасно запобігає переведенню системи у режим максимальної обережності за одиничного негативного спостереження короткострокового каналу. Значення негативного консенсусу повертається клієнту як окрема ознака та використовується як умова активації консервативної стратегії спрощення і як критерій додаткової перевірки коректності виконаних трансформацій. Схема формування стабільного значення керуючого параметра подана на рисунку 2.2.

Калібрування алгоритму стійкої емоції виконується через налаштовувані параметри алгоритму. Параметри задають вибір моделі комп'ютерного зору, пороги детекції обличчя, коефіцієнт затухання, пороги активації, мінімальну кількість кадрів та розмір буфера телеметрії. Такий підхід відповідає практиці інженерного експериментування, оскільки забезпечує порівняння конфігурацій на однакових сценаріях взаємодії без модифікації алгоритму та збереження відтворюваних профілів параметрів.

Валідація методу визначення стійкої емоції у роботі визначається як перевірка керованості значення параметра, а не як психодіагностичне оцінювання користувача.

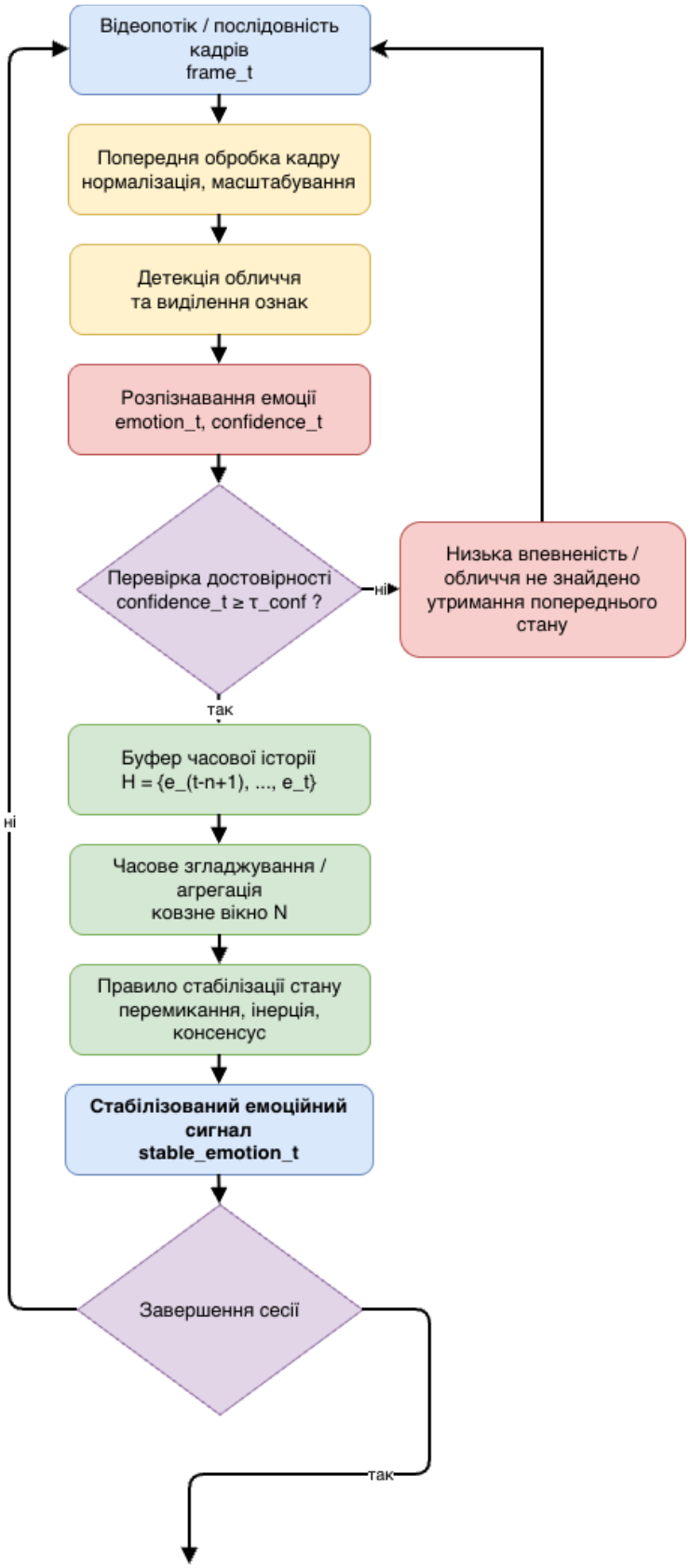


Рис. 2.2. Схема формування стабілізованого емоційного сигналу.

Серед перевірених властивостей виділено стабільність значення за

відсутності суттєвих змін стану, час входу у стійкий режим, частоту перемикань між станами та коректність спрацювання правила негативного консенсусу. Телеметричні дані, що генеруються сервісом, забезпечують вимірювання наведених характеристик на реальних сценаріях взаємодії та порівняння конфігурацій алгоритму.

Базові метрики керованості значення стійкої емоції визначаються через параметри телеметричної серії. Нехай N позначає кількість записів у часовому ряді, T – тривалість серії у секундах.

Метрики обчислюються за формулами (2.24–2.26):

$$C = \frac{|\{i: e_i \neq \emptyset\}|}{N}, \quad (2.24)$$

$$U = 1 - C, \quad (2.25)$$

$$R = \frac{|\{i: e_i \neq e^*_{i-1}\}|}{T}, \quad (2.26)$$

де C – покриття значення стійкої емоції за серією,

U – частка записів з невизначеним значенням,

R – частота переходів між стійкими емоціями (Гц),

e_i^* – значення стійкої емоції у і-му записі телеметрії.

Метрики не оцінюють психологічну коректність розпізнаної емоції, проте перевіряють відповідність вихідного значення вимогам до контуру керування підсистемою адаптивного спрощення.

Метрики C та U описують компроміс між доступністю значення керуючого параметра та обережністю алгоритму. Метрики R та L описують компроміс між чутливістю та стабільністю реакції алгоритму. У контексті вебдоступності обидві групи метрик мають однакове значення, оскільки надлишкова чутливість підвищує кількість перемикань і погіршує сприйняття інтерфейсу через його часті зміни, а надлишкова інерційність знижує здатність системи реагувати на тривалі негативні епізоди стану користувача (табл. 2.11).

Протокол калібрування параметрів стабілізації побудовано на основі телеметрії, контрольованих сценаріїв та визначеного набору метрик (табл. 2.12).

Таблиця 2.11

Метрики керованості сигналу стійкої емоції та агрегованого стану

Показник	Позначення	Обчислення	Інтерпретація
Покриття стійкою емоцією	C_e	$\frac{N(stableEmotion \neq null)}{N}$	Частка часу, у яку можливе керування на основі емоції
Покриття станом	C_s	$\frac{N(consensusState \neq null)}{N}$	Частка часу, у яку можливе керування на основі стану
Частота перемикань емоції	R_e	$\frac{K_e}{(\frac{T}{60})}$	Кількість змін стійкої емоції за хвилину
Частота перемикань стану	R_s	$\frac{K_s}{(\frac{T}{60})}$	Кількість змін стану за хвилину
Середня довжина інтервалу сталості емоції	L_e	mean(runLength(stableEmotion))	Інерційність та передбачуваність режиму, визначеного емоцією
Середня довжина інтервалу сталості стану	L_s	mean(runLength(consensusState))	Інерційність та передбачуваність режиму, визначеного станом
Частка інтервалів невизначеності	U	$\frac{N(stableEmotion = null)}{N}$	Частка часу без активного рішення системи

Таблиця 2.12

Протокол калібрування параметрів стабілізації стійкої емоції

Етап	Дані	Результат	Критерій прийнятності
Формування сценаріїв	Часові серії з камери в типових умовах	Набір еталонних серій	Сценарії покривають нейтральні та негативні стани користувача
Вибір області параметрів	Діапазон значень навколо значень за замовчуванням	Набір конфігурацій	Конфігурації не призводять до повної відсутності визначених значень керуючого параметра
Запуск серій і збір телеметрії	Експорт часового ряду спостережень	Файли телеметрії	Відтворювані умови виконання серій
Обчислення метрик	Серії та Таблиця 2.9	Набір значень метрик	Усі метрики мають числові значення для кожної серії
Вибір конфігурації	Порівняння метрик	Обрана конфігурація	Мінімізація R за заданих нижніх меж C і L

Базовою одиницею експерименту виступає запис часової серії у типових сценаріях використання, серед яких читання нейтрального тексту, читання тексту з підвищеною когнітивною складністю та взаємодія з частими відхиленнями погляду користувача від камери. Для кожної серії фіксується конфігурація параметрів алгоритму, після чого обчислюються метрики C, R, L та U. Калібрування виконується у просторі параметрів затухання, порогів активації та мінімальної кількості спостережень для короткострокового і довгострокового каналів, що задаються налаштовуваними параметрами.

Метою калібрування параметрів стабілізації є мінімізація частоти перемикань керуючого параметра за умови збереження достатнього покриття серії та прийнятної інерційності реакції. У такій постановці стійка емоція виступає не лише результатом розпізнавання, а інструментом керування, що адаптується під вимоги системи спрощення та характеристики потоку даних.

Калібрування виконано на телеметричному корпусі з трьох серій (399 кадрів, $\bar{c} = 0{,}77$, fps ≈ 5 к/с) за моделями VGG-Face, FaceNet та VGG з підвищеним порогом. Для кожного каналу стабілізації дизайн-параметр θ визначено за необхідним рівнем впевненості (від 58 % до 71 % від $\bar{c}$), а коефіцієнт затухання α обчислено аналітично як максимальне значення, що задовольняє обмеження часу реакції $T_{react} \leq 5$ кадрів (1,0 с) та ефективного вікна спостереження $\tau\_eff \leq \tau\_max$ (додаток А, табл. 4). Верифікація на телеметрії підтвердила покриття C від 0,46 до 0,61, частоту перемикань R від 27 до 39 хв$^{-1}$ та тривалість стабільного рішення L від 1,5 до 2,8 с.

Модуль визначення стійкої емоції виступає компонентом системи когнітивної вебдоступності з формуванням вхідного показника стану користувача для адаптивного спрощення тексту у реальному часі. Призначення модуля охоплює стабілізацію значення у часі з керованою інерцією, параметризованими порогами активації та правилом негативного консенсусу між короткостроковим та довгостроковим каналами замість класифікації емоції на рівні окремого кадру.

2.3. Спеціалізовані словники синонімічних замін української мови

У запропонованому методі забезпечення когнітивної доступності тексту словники синонімічних замін виконують функцію базового лексичного механізму керованого, відтворюваного та інтерпретованого спрощення. На відміну від генеративних підходів, словниковий рівень використовує наперед підготовлені та перевірені варіанти замін, що знижує ризик спотворення змісту тексту під час трансформацій останнього.

У структурі методу словники виступають окремим функціональним компонентом системи. Призначення словників полягає у первинній заміні проблемних лексичних одиниць, виявлених під час аналізу тексту, серед яких рідковживані слова, складна лексика, сленг, абревіатури, застарілі мовні форми та ненормативна лексика. Для кожної з цих категорій словник задає набір допустимих варіантів спрощення, що обмежує алгоритм від довільних або семантично нестійких перетворень.

Словники забезпечують детермінованість результату трансформацій тексту. Наявність наперед підготовленого набору перевірених відповідників для проблемної лексеми дозволяє системі виконувати заміну без повторної генерації та без звернення до зовнішніх сервісів. Така організація знижує варіативність результатів, скорочує час обробки тексту і формує відтворювану основу для експериментальних порівнянь конфігурацій методу за однакових умов.

Кожна словникова заміна оформлена як явний запис з пріоритетними відповідниками, словоформами та службовими атрибутами, що формує інтерпретованість трансформацій тексту. Прозора структура запису підтримує перевірку результатів роботи системи, аналіз окремих замін та коригування записів за потреби, що знижує ризик зміни смислових акцентів висловлювання при невдалих варіантах заміни.

Адаптація спрощення до морфологічних особливостей української мови потребує спеціального оформлення словникових записів. Пара «складне слово – простий відповідник» не вичерпує потреб української мови, у якій одна лексема реалізується у десятках словоформ. Записи у словнику організовано з

підтримкою пошуку як за лемою, так і за конкретною словоформою, що забезпечує застосування замін у реальних текстах без порушення граматичної узгодженості.

Словники у запропонованому методі виступають джерелом готових синонімічних замін і стабілізаційним компонентом системи спрощення. Контрольованість лексичних перетворень, незалежність від зовнішніх генеративних сервісів, накопичення перевірених результатів та структурованість записів формують основу для подальшого розвитку методу забезпечення когнітивної доступності тексту.

Словникова система методу забезпечення когнітивної доступності тексту побудована за багатокатегоріальною структурою, яка зумовлена різнорідністю походження лексичних бар'єрів, неоднаковим ступенем їх впливу на розуміння та потребою у відмінних стратегіях спрощення для кожної категорії. Єдиний універсальний словник для всіх типів проблемної лексики не враховував би ані семантичних особливостей слів окремих категорій, ані характеру їх функціонування у тексті.

Словниковий рівень методу організовано як систему взаємопов'язаних категоріальних словників, кожен з яких орієнтований на окремий клас когнітивних бар'єрів. До системи входять словники рідковживаної, застарілої та складної лексики, сленгу, абревіатур та словники для роботи з ненормативними мовними одиницями. Окрему підсистему утворює механізм пояснення термінів, що реалізує не синонімічну заміну, а коротке розкриття значення спеціального поняття у тексті.

Кожна категорія лексики створює власний тип труднощів для користувача: рідковживані слова знижують впізнаваність лексики, складні слова підвищують навантаження на обробку значення, сленг порушує стилістичну нейтральність викладу, абревіатури ускладнюють декодування скорочених форм, застаріла та ненормативна лексика створює додаткові бар'єри для сприйняття тексту. Словникова система містить не лише варіанти замін, а й структурно відображає природу лексичних труднощів через категоризацію записів.

Словник рідковживаної лексики містить слова сучасної мови з низькою частотністю вживання, які знижують впізнаваність тексту для користувача. Проблема категорії пов'язана не з порушенням мовної норми, а зі слабкою активністю слів у словниковому запасі читача. Словник складної лексики охоплює слова, нормативні з граматичної точки зору та поширені у певних жанрах, що вимагають додаткових когнітивних ресурсів на сприйняття через структуру, довжину, словотвірні особливості або стилістичну складність.

Словник сленгу формує окрему категорію через високу контекстну залежність сленгових одиниць, їхнє емоційне забарвлення та обмежену прозорість значення для широкої аудиторії у формальних типах тексту. У системі спрощення сленгові одиниці замінюються на стилістично нейтральні відповідники. Словник абревіатур виконує іншу функцію, орієнтовану на розкриття скорочень через подання повної форми, а не на підбір синонімічних варіантів.

Застаріла лексика об'єднує слова з історично віддалених, архаїчних або стилістично застарілих мовних шарів. Такі одиниці трапляються у культурно-історичних, художніх та гуманітарних текстах і потребують окремого підходу до спрощення, що зберігає зміст висловлювання та наближає форму подання до сучасної мовної норми. Ненормативна лексика об'єднана в окрему словникову групу з функцією нейтралізації грубих або ненормативних мовних форм у тексті.

Особливу роль у структурі словникової системи відіграє підсистема термінів. На відміну від інших категорій, для термінологічної лексики не завжди можливо або доцільно використовувати прямий синонім. У багатьох випадках заміна терміна іншим словом може призвести до втрати точності. Для цієї категорії застосовується інший підхід через коротке пояснювальне розкриття значення слова у тексті. Такий підхід зберігає термін як носій предметного змісту та одночасно знижує бар'єр для його розуміння читачем.

Категоріальна структура словникової системи побудована за принципом відповідності між типом лексичного бар'єра та способом його подолання. Така організація формує цілісну систему лексичних ресурсів, узгоджену з логікою

методу забезпечення когнітивної доступності тексту, на відміну від несистематизованої сукупності замін. Структуру і характеристики словникових груп узагальнено в таблиці 2.13. Для кожної категорії визначено тип когнітивного бар'єра, на який орієнтований словник, тип ключа запису (лема або словоформа), характер заміни та основні методи формування записів.

Таблиця 2.13

Категоріальна структура словникової системи

Категорія словника	Тип когнітивного бар'єра	Тип ключа	Характер заміни	Спосіб формування
rare	Низька впізнаваність рідковживаної лексики	lemma	Нейтральний спрощений відповідник	Лематизація, batch-поповнення, ручна та експертна валідація
complex	Підвищена лексична та словотвірна складність	wordform	Лексична або описова заміна	Генерація кандидатів, ручний добір, LLM/batch, QC
slang	Стилістично знижена та контекстно залежна лексика	lemma	Стилістично нейтральний відповідник	Лематизація, ручне поповнення, LLM/batch, валідація
abbr	Ускладнене декодування скорочень	lemma	Розшифрування скорочення	Зовнішні джерела, ручне уточнення, QC
obsolete_group	Історично віддалена та застаріла лексика	mixed	Сучасний або пояснювальний відповідник	Rule-based, генерація, ручне доопрацювання, QC
profanity_group	Ненормативна, нецензурна та вульгарна лексика	mixed	Нейтральна безпечна заміна	Ручне формування, baseline, batch-поповнення, QC
term_definitions	Термінологічна складність	lemma	Коротке пояснювальне розкриття	Ручне формування та контрольоване поповнення

На рисунку 2.3 показано обсяг робочих словників лексичних замін за основними категоріям. Найбільшу частку записів містить словник складної лексики, що відповідає її широкій представленості у вебтекстах та різноманітності словотвірних і стилістичних форм. Словники архаїчної та лайливої лексики формують другу за обсягом групу через широкий спектр відповідних мовних одиниць у відкритих джерелах та потребують їх систематичного покриття. Словник сленгу та словник абревіатур мають менший обсяг, що пов'язано з обмеженою стабільністю сленгових форм у часі та наявністю усталеного ядра скорочень. Словники нецензурної, застарілої та

вульгарної лексики містять найменшу кількість записів, що відображає вужчий обсяг відповідних мовних шарів та переважно ручний характер їх формування.

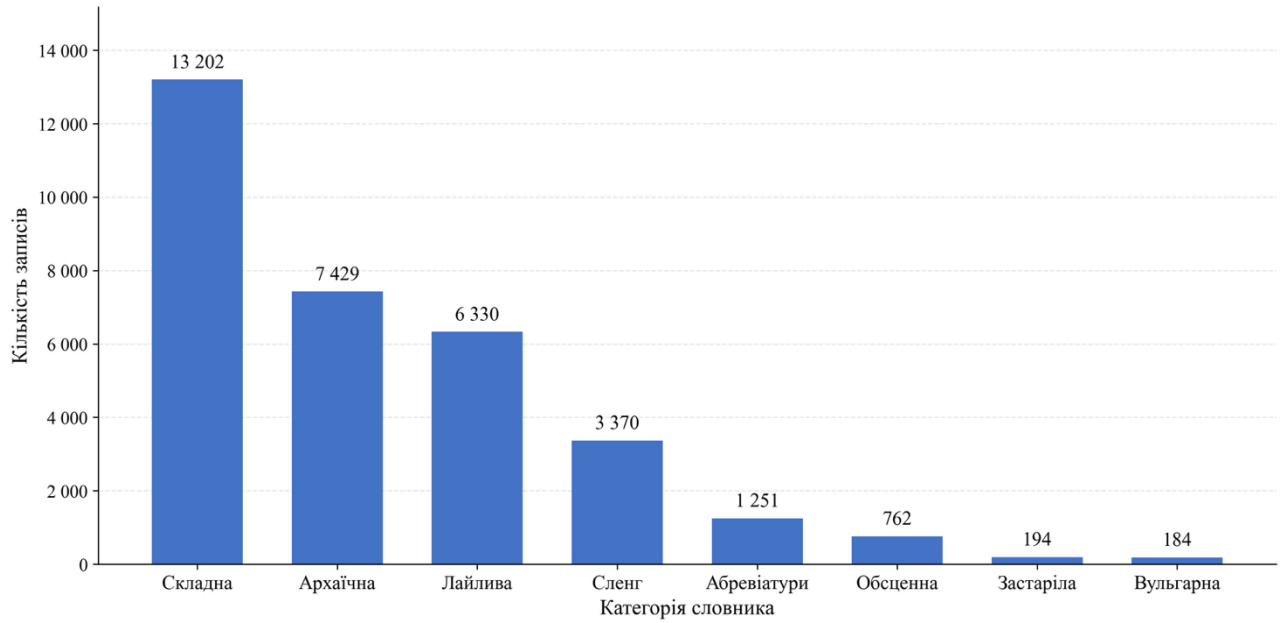


Рис. 2.3. Кількісні характеристики словників замін за категоріями.

Словникова система реалізована як два взаємопов'язані рівні. Рівень першоджерело містить усі словоформи проблемних лексем і призначений для детекції бар'єрів у тексті. Рівень замін містить базові форми (леми) з відповідними синонімами та забезпечує підстановку замін. Такий поділ зумовлений морфологічними особливостями української мови, у якій одна лема у середньому реалізується через приблизно 8,5 словоформи у тексті, що унеможливлює пряме зіставлення з лемами без морфологічного розбору. Загальний обсяг словників першоджерел становить 467 872 словоформи, словників замін – 54 631 лему.

Винятком виступає категорія застарілої лексики, у якій обсяги першоджерел та замін збігаються через переважну фіксацію застарілих слів в одній формі внаслідок їх виходу з активного вжитку. Категорія рідковживаної лексики демонструє близькі значення обох рівнів (21 909 та 22 052 одиниці відповідно), що пояснюється низькою словозмінною активністю рідковживаних лексем у реальних текстах.

Словниковий запис у методі забезпечення когнітивної доступності

виступає структурованою одиницею знання про можливе спрощення конкретної лексеми. Запис містить варіанти замін, правила їх використання у тексті, схему пріоритизації кандидатів та механізм роботи з різними словоформами вихідного слова.

Ключова лексична одиниця запису представлена лемою або словоформою залежно від категорії словника. Подання у вигляді лем застосовується для зведення різних словоформ до спільної опорної форми та накопичення знання на рівні лексеми. Подання через словоформи застосовується у випадках з потребою у збереженні безпосереднього зв'язку з конкретною формою у тексті або за некоректного відновлення заміни лише через лему.

Кожен словниковий запис містить упорядкований список потенційних замін з визначеним пріоритетом для кожного варіанта. Пріоритизація дозволяє системі обирати основний варіант заміни у звичайному режимі та одночасно зберігати інші можливі заміни для подальшого аналізу, ручного редагування або уточнення у конкретному контексті. Словниковий запис фіксує не одну відповідність, а керовану множину допустимих варіантів спрощення вихідної лексеми.

Словниковий запис містить інформацію про словоформи опорної лексеми, що пов'язує запис з конкретними реалізаціями слова у тексті та підтримує точні підстановки у випадках, коли автоматичне морфологічне узгодження є недостатнім або небажаним для забезпечення граматичної коректності. Інформація про словоформи входить до структури запису разом з пріоритизованими замінами, що формує одночасно семантичний і граматичний рівень опису спрощення.

Структура словникового запису містить службові атрибути, що включають відомості про походження заміни, стан її перевірки та додаткові ознаки технологічного супроводу словника. Атрибути розрізняють вручну підтверджені та автоматично згенеровані записи, відстежують процес поповнення словника і визначають, які записи допускаються до робочого режиму без додаткової перевірки. Службові атрибути забезпечують процедурний рівень опису поряд із

лінгвістичним наповненням запису.

Модель запису фіксує контекстні та стильові характеристики кандидатів на заміну. Одна лексема має кілька спрощених відповідників, придатних для різних типів тексту, через що кожен кандидат отримує атрибути комунікативної ситуації, у якій його застосування є коректним. Контекстні та стильові характеристики мають значення для категорій сленгу, абревіатур, застарілої лексики та термінологічних одиниць, у яких невдалий вибір заміни знижує точність або порушує жанрову доречність тексту.

Модель словникового запису побудована як багатокомпонентна структура, що поєднує лексичний ключ, набір пріоритетних замін, словоформний рівень і службові атрибути контролю. Така організація формує словник як інструмент керованого спрощення тексту з підтримкою валідації, відстеження походження записів та узгодження замін з контекстом, на відміну від простого переліку синонімів. Узагальнену структуру словникового запису та функції його компонентів наведено в таблиці 2.14.

Метод формування словників включає послідовність етапів від збору лексичних даних, через нормалізацію та групування одиниць, побудову первинних кандидатів замін, їх очищення і фільтрацію, експертну валідацію до інтеграції готових записів у робочу словникову систему. Послідовність етапів забезпечує перехід від неструктурованого набору проблемної лексики до формалізованого словника, придатного для використання у програмному комплексі спрощення тексту.

На першому етапі формується словникова база для окремих категорій проблемних лексичних одиниць. Джерелом виступають категоріальні масиви лексики зі словами, їхніми формами, лемами та допоміжними мовними ознаками. На цьому рівні словник виконує функцію вхідного лексичного ресурсу без готових замін. Розмежування первинних джерел та робочих словників відокремлює етап виявлення проблемної лексики від етапу формування рішень для її спрощення зі збереженням чистоти вхідних даних та можливістю повторного використання первинних джерел для різних стратегій спрощення.

Таблиця 2.14

Структура словникового запису

Елемент запису	Зміст	Обов'язковість	Роль у системі
Ключ словникового запису (lemma або wordform)	Лема або словоформа, до якої прив'язується запис	Так	Опорна одиниця для пошуку та групування записів
priority_replacements	Упорядкований список кандидатів заміни з пріоритетами	Так	Постачає алгоритму основний та альтернативні варіанти спрощення
replacement	Текст конкретної заміни у складі кандидата	Так	Використовується як безпосередній результат лексичної трансформації
priority	Позиція кандидата у порядку переваги	Так	Визначає основний варіант заміни серед кандидатів
context	Контекстне призначення заміни	Рекомендовано	Обмежує використання заміни відповідно до типу тексту
formality	Стильова характеристика кандидата	Рекомендовано	Забезпечує стилістичну доречність заміни
word_forms	Мапа точних словоформних відповідників	Ні	Підтримує застосування замін без помилок морфологічного узгодження
metadata	Службові відомості про джерело, пере-вірку та стан запису	Ні	Підтримує контроль якості, модерацію та технічний супровід словника
isVerify, needs_review	Ознаки перевіреності або потреби у рев'ю	Ні	Визначають допуск запису до робочого режиму

На другому етапі зібрані одиниці нормалізуються до уніфікованого подання, очищуються від технічних відмінностей запису та групуються за канонічною формою. Групування за лемами усуває дублювання записів, накопичує знання на рівні опорної одиниці та формує зв'язок між лемою і поверхневими реалізаціями слова у тексті.

На третьому етапі формуються варіанти замін через стратегії з варіюванням залежно від типу словника. Частина категорій використовує вручну підготовлені або побудовані на правилах відповідники, інші категорії спираються на зовнішні джерела та напівавтоматичні засоби добору, а масштабне поповнення окремих словникових груп виконується через залучення генеративної моделі. Третій етап формує впорядкований набір варіантів з можливістю подальшого уточнення або коригування у наступних етапах процедури.

На четвертому етапі виконується очищення та уніфікація кандидатів. На цьому рівні усуваються самозаміни, дублікати, технічно пошкоджені фрагменти, надмірно довгі та стилістично непридатні варіанти. Контекстні та стильові атрибути записів нормалізуються, перевіряється наявність необхідних службових полів. Очищення та уніфікація перетворюють сукупність попередньо згенерованих варіантів на придатний для використання словниковий матеріал з внутрішньою узгодженістю записів, що є обов'язковою умовою подальшої автоматичної обробки.

П'ятий етап реалізує експертну перевірку варіантів замін. Автоматично запропонована заміна не приймається у систему лише за фактом генерації. Запропоновані варіанти проходять перевірку на семантичну адекватність, зрозумілість, стилістичну відповідність та безпечність для змісту тексту. Перевірка має ключове значення для складної, термінологічної, застарілої та емоційно маркованої лексики з ризиком спотворення змісту висловлювання або порушення жанрових меж тексту через невдалу заміну. Підтверджений експертом запис переводиться у робочий стан та використовується алгоритмом без додаткових обмежень.

Шостий етап реалізує інтеграцію підтверджених записів у робочу словникову систему. Інтегрована лексична одиниця стає частиною відтворюваного ресурсу, що використовується у наступних ітераціях алгоритму. Кожен цикл формування словника збільшує обсяг накопичених знань системи замість завершення разовим результатом. Така властивість забезпечує масштабованість методу через поступове розширення словникової бази та зменшення частки випадків, у яких потрібне зовнішнє генерування варіантів.

Особливістю запропонованого методу виступає поєднання лексикографічного, алгоритмічного та експертного рівнів формування словників. Лексикографічний рівень спирається на структуровані лексичні джерела і морфологічне впорядкування записів. Алгоритмічний рівень підтримує поповнення словника програмними процедурами генерації, очищення, індексації та контролю якості. Експертний рівень регулює остаточне

введення нових замін у стабільний словниковий ресурс через процедуру перевірки. Поєднання трьох рівнів усуває обмеження ручного та автоматизованого підходів і використовує переваги кожного з них у єдиній процедурі формування словникової бази.

Метод формування словників синонімічних замін у запропонованій системі побудовано як керовану багатоетапну процедуру з охопленням збору лексичних даних, нормалізації, генерації варіантів, очищення, експертної валідації та інтеграції записів у робочий словниковий шар. Така структура процедури забезпечує відтворюваність словникової бази, її системне розширення та придатність до використання у задачах автоматизованого забезпечення когнітивної доступності українських вебтекстів. Послідовність основних етапів формування словників синонімічних замін у межах запропонованого методу узагальнено на рисунку 2.4.



Рис. 2.4. Схема формування та валідації словників синонімічних замін.

Контроль якості словникових записів виступає окремим етапом методу формування словників, оскільки записи поповнюються через ручні, напівавтоматичні та генеративні процедури. Формально коректна заміна

виявляється семантично неточною, стилістично невдалою або непридатною для використання у текстах різних жанрів без перевірки якості. Словниковий ресурс у системі є об'єктом постійного контролю, а не статичним набором даних.

Перший рівень контролю реалізує формальну перевірку структури словникових записів. На цьому рівні оцінюється наявність обов'язкових полів, коректність подання ключа, упорядкованість списку замін та допустимість службових атрибутів. Формальна перевірка забезпечує використання словникового запису у програмному контурі без технічних помилок, проте не гарантує якості змісту, через що доповнюється змістовими процедурами очищення.

Другий рівень контролю реалізує очищення кандидатів замін. На цьому рівні усуваються самозаміни, дублікати, порожні та технічно пошкоджені значення, варіанти з надмірно довгими фразами, неприйнятні службові слова та записи, що порушують внутрішні правила подання словника. Стильові та контекстні атрибути нормалізуються до уніфікованої форми. Очищення на рівні контролю якості забезпечує внутрішню узгодженість словників, що формуються з кількох джерел або через генеративне поповнення, та виявляє відхилення, які не зафіксовані на попередньому етапі формування записів.

Третій рівень контролю реалізує оцінювання покриття словників та виявлення суперечностей між окремими групами замін. На рівні цілого ресурсу контролюються повнота покриття категоріальних списків, наявність колізій між групами, співвідношення між сирими джерелами та робочими словниками, а також рівень готовності словникової бази до експериментального використання. На цьому рівні словник аналізується як інструмент з власними кількісними та структурними характеристиками без зведення до сукупності окремих лексичних записів.

Формальна та статистична перевірка недостатня для словника, що застосовується у задачі забезпечення когнітивної доступності. Експертна валідація залишається обов'язковим етапом контролю якості, у межах якого оцінюється збереження змісту після заміни, зрозумілість запропонованого

варіанта для цільового користувача, відсутність небажаних стилістичних зсувів та збереження якості сприйняття тексту. Термінологічні одиниці, застаріла лексика, слова з емоційним забарвленням та випадки багатозначності потребують детальнішої експертної перевірки, у яких автоматичний добір варіанта не забезпечує достатньої точності рішення.

Експертна валідація виконує функцію межі між шарами варіантів та робочих записів словникової системи. Автоматично згенерований або напівавтоматично отриманий запис не переходить у стабільний словник лише за фактом його формування системою. До моменту підтвердження запис залишається попереднім кандидатом і відфільтровується з робочого використання. Підтверджений запис набуває статусу робочого словникового елемента та застосовується в алгоритмі спрощення тексту.

Поєднання масштабованості поповнення словників із вимогою надійності реалізовано через двоступеневу структуру. Автоматизовані інструменти забезпечують швидке розширення словникової бази, після чого експертна перевірка визначає допуск нових записів до робочого ресурсу. Словник накопичує відібраний та стабілізований набір мовних рішень замість необмежених генеративних варіантів.

Контроль якості та експертна валідація виступають складниками методу формування словників синонімічних замін. Процедури контролю забезпечують структурну коректність записів, змістову придатність кандидатів, узгодженість словникової системи та безпечне використання нових замін у робочому алгоритмі.

Словники синонімічних замін виступають базовим джерелом даних для лексичного перетворення у запропонованому методі без статусу ізольованого лексичного ресурсу. Інтеграція побудована за принципом пріоритетної роботи словникового рівня, що забезпечує швидке, відтворюване та контрольоване отримання кандидатів на заміну. Кожна виявлена проблемна одиниця співвідноситься з відповідним словниковим шаром на першому кроці алгоритму, після чого система переходить до інших способів побудови заміни лише за

відсутності придатного запису у словнику.

Інтеграція побудована за принципом відповідності між типом лексичного бар'єра та логічною групою словника. Кожна виявлена на етапі аналізу проблемна одиниця має категоріальну належність, що визначає словниковий ресурс для пошуку заміни. Такий розподіл уникає змішування різних типів замін та забезпечує окремі правила обробки для рідковживаної, складної, застарілої, сленгової та ненормативної лексики. Словники інтегруються в алгоритм як упорядкована система категоріальних ресурсів, а не як єдина плоска база.

Алгоритм пошуку у словниках враховує морфологічну природу української мови. За наявності леми для проблемного слова пошук виконується на рівні номінативної форми. Категорії, що потребують збереження зв'язку з поверхневою реалізацією, обробляються через пошук за словоформою. Перехід від словоформи до опорної леми реалізовано через окремі індекси, що поєднують компактне накопичення знань на рівні лексеми з коректною обробкою реальних текстових форм.

Алгоритм отримує впорядкований набір варіантів зі знайденого словникового запису без єдиної фіксованої заміни. Пріоритетний варіант використовується як основний, альтернативні кандидати залишаються для інших режимів застосування, ручного втручання або подальшого розширення методу. Словниковий запис містить також точні словоформні відповідники, які застосовуються замість загального морфологічного узгодження за потреби. Інтеграція словника в алгоритм виконується як процедура вибору та застосування структурованого запису, а не як просте звернення до таблиці відповідностей.

Кешування результатів словникового пошуку забезпечує практичну ефективність алгоритму. Повторне оброблення тих самих лексем у процесі ітераційного спрощення тексту виникає регулярно, через що збереження результатів попередніх звернень знижує кількість повторних операцій і прискорює роботу системи. Кешування зберігає словниковий рівень стабільним і малоресурсним у використанні, що підтримує його роль основного механізму

лексичного спрощення для великих текстів та у багатокрокових сценаріях обробки.

Інтеграція словників у метод визначає роль генеративних моделей у загальній архітектурі. За відсутності придатної заміни у словнику метод залучає зовнішній механізм генерації варіантів як розширення словникового рівня без його заміни. Підтверджені після перевірки результати генерації переходять до словникового ресурсу і надалі використовуються як стабільні записи. Словник виконує роль накопичувального ядра системи, тоді як генеративний компонент працює як допоміжний інструмент поповнення.

Інтеграція словників у програмний алгоритм спрощення реалізує зв’язок між лексичним ресурсом, процедурою аналізу та механізмом застосування замін. Вона охоплює категоріальний вибір словника, пошук за лемою і словоформою, роботу з пріоритетними кандидатами, використання словоформних відповідників, кешування результатів та контрольований перехід до генеративних засобів за відсутності словникового запису. Схему роботи словникового шару подано на рисунку 2.5.

Словники синонімічних замін виступають базовим лексичним компонентом методу забезпечення когнітивної доступності тексту. Словниковий шар має багатокатегоріальну структуру, спирається на формалізовану модель запису, формується через керовану багатоетапну процедуру та підтримується засобами контролю якості та експертної валідації.

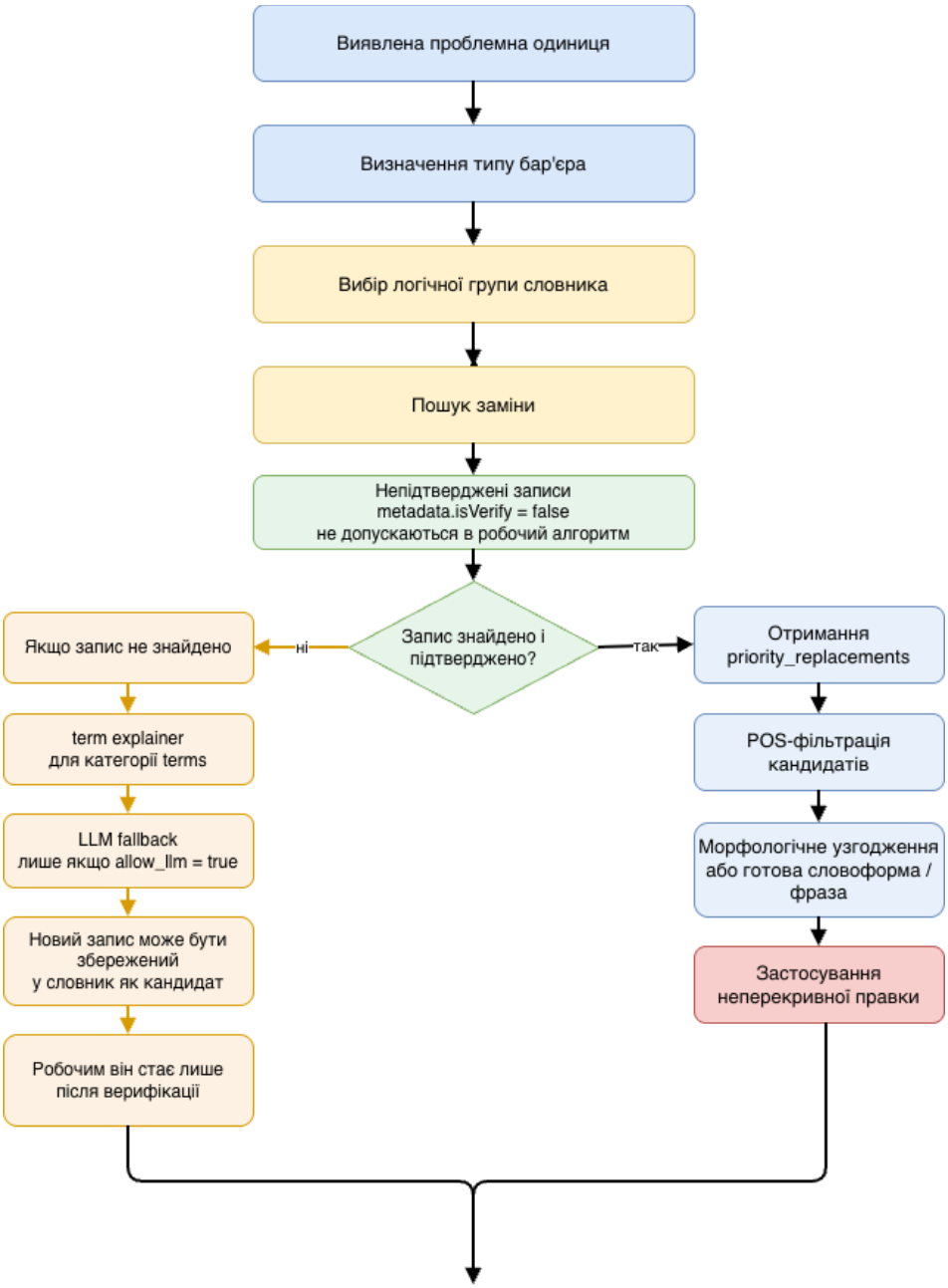


Рис. 2.5. Інтеграція словникового шару в метод спрощення.

Інтеграція словників метод забезпечує відтворюваність лексичних перетворень, знижує залежність від зовнішньої генерації та формує накопичуваний ресурс перевірених замін для подальшого розвитку системи.

2.4. Узагальнений метод забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями

Узагальнена модель забезпечення виступає керованим процесом перетворення тексту з обґрунтуванням кожного кроку через вимірюваний критерій. Вхід і вихід методу формалізовано у таблиці 2.15. Контекст взаємодії k впливає на політику обережності, вибір класів трансформацій та правила зупинки.

Показник стабільної емоції виступає контекстним вхідним показником для активації режиму допомоги та врахування під час повторних запусків процесу спрощення (рис. 2.6).



Рис. 2.6. Інтеграція стабілізованого показника емоційного стану у механізм адаптивного спрощення тексту.

Систему позначень методу наведено у таблиці 2.15. Задача методу формулюється як вибір варіанта y з множини допустимих варіантів $Y(x, S, k, \theta)$ з досягненням максимуму функціонала корисності J .Формалізація методу потребує єдиної системи позначень та визначення сутностей оптимізації разом з обмеженнями процесу.

Таблиця 2.15

Система позначень та основні змінні узагальненої моделі забезпечення

Позначення	Зміст	Джерело або спосіб формування
x	Вхідний текст або фрагмент	Текст документа або область S
S	Область аналізу і спрощення	Документ, сторінка або курсорний режим
$D(x,S)$	Результат детекції бар'єрів	Пакетний аналіз з фіксацією координат входжень
$C(x,S)$	Набір кандидатів на трансформації	Ранжування результатів D з урахуванням режиму
y	Результат після трансформацій	Застосування патчів до x у межах S
$J(y,x,k)$	Функціонал корисності	Комбінація приросту якості та штрафів
k	Контекстний стан взаємодії	Керуючий параметр режиму обережності

Функціонал J охоплює приріст когнітивної доступності, обсяг застосованих змін, ризик семантичного зсуву та порушення структурних обмежень процедури перетворення тексту (2.27).

$$y^* = argmax_{y \in Y(x,S,k,\theta)} J(y,x,S,k), \quad (2.27)$$

де $Y(x,S,k,\theta)$ – множина допустимих кандидатів покращеного тексту.

Функціонал корисності J побудовано як зважена комбінація приросту когнітивної доступності та штрафних доданків, що компенсують побічні ефекти трансформацій (2.28). Позитивним доданком є приріст ΔІКД на області S, штрафні доданки охоплюють зміну довжини тексту, щільність правок та ризик семантичного зсуву.

$$J(y,x,S,k) = \Delta\text{ІКД}(y,x,S) - \lambda_{len} \times \Delta len(y,x) - \lambda_{sem} \times risk_{sem}(y,x) - \lambda_{patch} \times \rho_{patch}(y,S), \quad (2.28)$$

де $\Delta\text{ІКД}(y,x,S) = \text{ІКД}(y,S) - \text{ІКД}(x,S)$ – приріст доступності,

λ_{len}, λ_{sem}, λ_{patch}– вагові коефіцієнти відповідних штрафів.

Окремо від м'яких штрафів функціоналу J існують обмеження множини $Y(x,S,k,\theta)$, що виключають з розгляду неприпустимі заміни. До таких обмежень належать заборона зміни числових значень без явної валідації, збереження власних назв і термінів у випадках ризику небезпечної підміни, недопущення перекривних замін та дотримання меж області аналізу.

Якість тексту вимірюється на трьох рівнях відповідно до сценарію роботи з вебконтентом. Сторінковий рівень оцінює якість у межах окремої сторінки і використовується для навігації, порівняння результатів редагування та формування узгоджених звітів. Блочний рівень оцінює весь блок тексту, відображає глобальну складність та виступає інтегральним критерієм для правила зупинки. Локальний рівень підтримує користувача під час читання або редагування через швидке виявлення найближчого бар'єра у контексті поточної позиції курсора.

Сторінка у методі задається як діапазон індексів у тексті. Метод використовує однакову логіку детекції та метрик для документа і для сторінки; відмінність полягає лише у межах області S. Для підтримки сценарію інтерактивного спрощення доцільно мати метод, який визначає найближчу проблему відносно позиції курсору або фокусу читання. У загальному випадку позиція визначається як індекс у тексті, який має бути відображений у структурній одиниці. Після цього формується множина проблемних місць, які мають координати в цій структурі. Рішенням є вибір кандидата з мінімальною відстанню за визначеною метрикою, яка враховує не лише кількість символів, а й межі речень і абзаців, оскільки саме ці межі визначають когнітивні переходи при читанні.

Детекція проблем у моделі відокремлена від процесу спрощення через різну природу цих задач. Вона відповідає на питання локалізації бар'єра та визначення його типу. Для різних рівнів деталізації формуються відповідні ознаки. На рівні сторінки або блоку застосовуються агреговані статистики, у локальному режимі використовуються локальні показники у конкретному реченні або групі слів. Результатом детекції є структурований список кандидатів з координатами у тексті, типом проблеми та оцінкою її важливості.

У методі детекція виступає побудовою розмітки поверх тексту. Кожен визначений елемент задається щонайменше діапазоном індексів у вихідному рядку, що забезпечує дві функції детекції. Підсвічування проблемних місць в інтерфейсі та інтерпретованість рекомендацій реалізуються через координатну

прив'язку елементів. Формування кандидатів для спрощення спирається на точність меж елементів, що визначає коректність заміни та можливість морфологічного узгодження. Рисунок 2.7 узагальнює процес спрощення тексту у режимах документа, сторінки та курсора.

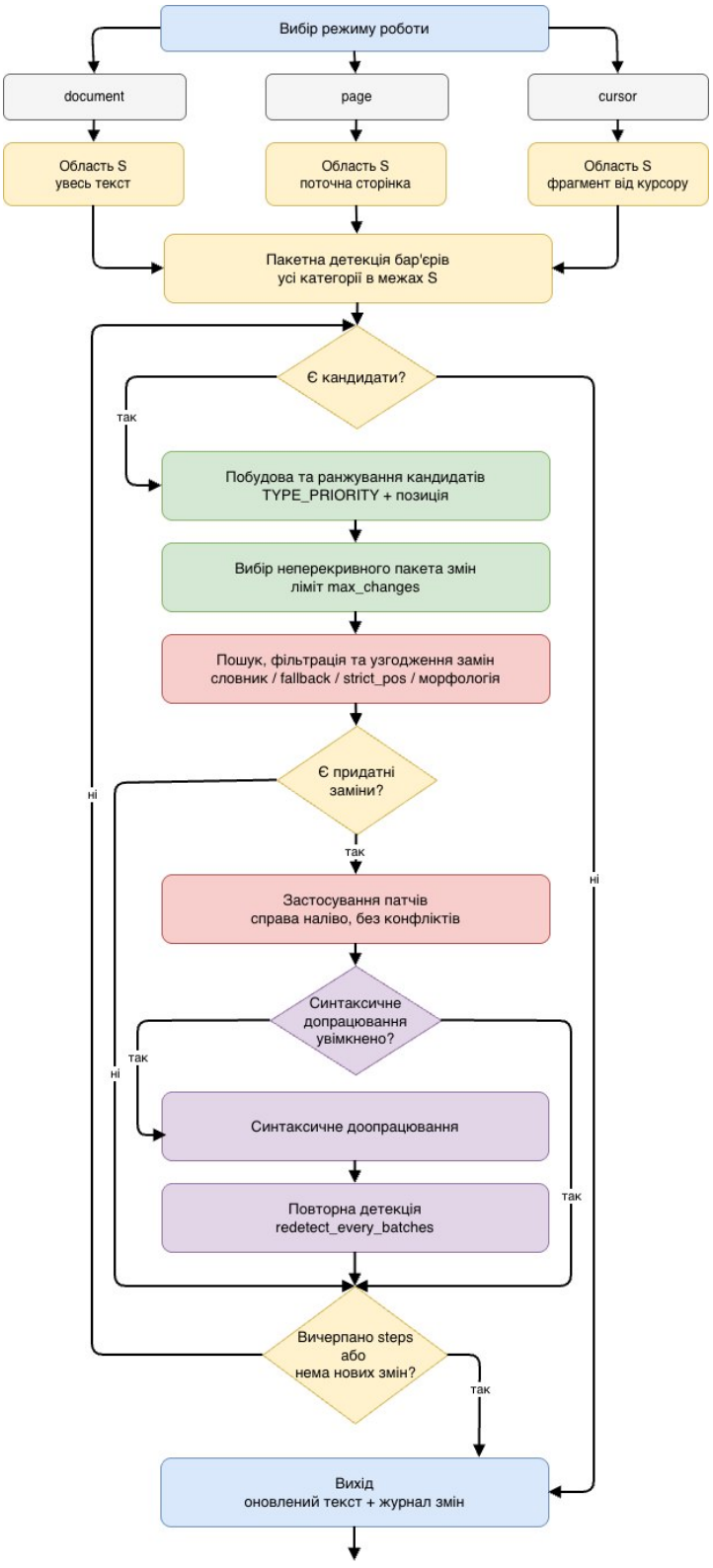


Рис. 2.7. Схема спрощення тексту.

Вибір режиму визначає межі області обробки зі збереженням загальної структури алгоритму. Для вибраної області виконується пакетна детекція бар'єрів, формується ранжована множина кандидатів, застосовуються неперекривні трансформації, після чого виконується повторна детекція і за потреби синтаксичне доопрацювання. Завершення ітерації контролюється умовами зупинки, що забезпечують керованість спрощення, обмеження надлишкових змін тексту та відтворюваність результату.

Деталізація елементів подана у таблиці 2.16 з узагальненням типів проблем та технічних засобів обробки у методі спрощення.

Таблиця 2.16

Категорії бар’єрів і технічні підходи до їх детекції

Категорія	**Процедура детекції**	**Тип детекції**	**Основні джерела неоднозначності**
Ненормативна лексика	Словниковий індекс з морфологічним розбором, нечітке зіставлення за потреби	Детермінована з евристиками	Варіативність форм і помилки токенізації
Застаріла лексика	Індекс словника за лемою і формою	Детермінована	Полісемія та доменні варіанти значення
Рідковживані слова	Індекс словника за лемою і формою	Детермінована	Доменні терміни і власні назви
Складні слова	Індекс словника за лемою і формою	Детермінована	Контекстна складність і багатослівні одиниці
Сленг	Класифікатор з оцінкою і порогом, додаткові евристики для спеціальних токенів	Імовірнісна	Залежність від предметної області та стилю спілкування
Терміни	Класифікація токенів з порогами, фільтри та евристики виключень	Імовірнісна з фільтрами	Власні назви, загальні іменники, різні предметні області
Абревіатури	Гібрид регулярних виразів та моделі машинного навчання з координатами	Змішана	Пунктуація, короткі слова, контекст скорочення

Таблиця ілюструє принцип комбінування різних типів детекції у межах одного методу. Для категорій ненормативної лексики, застарілої, рідковживаної та складної лексики детерміновані підходи забезпечують стабільний та інтерпретований результат. Для сленгу, термінів і частини абревіатур контекст

впливає на сам факт існування бар'єра, через що застосовуються імовірнісні детектори з порогами і фільтрами.

Політика ранжування кандидатів детекції в методі визначає порядок застосування спрощень в ітераційній процедурі. Нейтральний контекст використовує ранжування за очікуваним приростом якості тексту. Негативний контекст враховує ризик дестабілізації сприйняття користувача, через що пріоритет отримують проблеми зі значним внеском у складність та можливістю локального виправлення без перебудови великих фрагментів тексту.

Заміна або трансформація у методі виступає керованою дією переведення тексту до нового стану. Дія обмежується правилами безпеки, серед яких збереження змістової цілісності, незмінність числових даних і фактологічних елементів та обмежений обсяг доданих пояснень. Метод генерує декілька кандидатів трансформації для кожної знайденої проблеми з подальшим відбором за критерієм якості та обмеженнями ризику.

Метод використовує два класи джерел кандидатів трансформації. Детермінований клас спирається на словники замін та правила морфологічного узгодження за схемою, описаною вище у підрозділі 2.3. Генеративний клас спирається на велику мовну модель, що формує переформулювання або пояснення для випадків недостатнього словникового покриття. Класи виступають взаємодоповнюючими у межах методу через поєднання контрольованості детермінованих замін з природністю генеративних трансформацій.

Застосування заміни описується у методі як правка у тексті з межами фрагмента та цільовим рядком. Декілька правок за одну ітерацію виконуються за умови, що їхні межі не перекриваються, інакше заміни взаємно спотворюють координати одна одної. Правки застосовуються у порядку від правого краю до лівого краю для збереження коректності позицій решти кандидатів. Процедура не залежить від кількості символів у замінах та зберігає відтворюваність результату через обчислення координат кандидатів у базовому тексті без перерахунку після кожної локальної операції.

Генеративний рівень замін активується як резервний механізм за умови відсутності придатного кандидата на детермінованому рівні або потреби у переформулюванні фрагмента. Доступ до генеративних замін у реалізації керований через значущий вплив запитів до LLM на витрати виконання та зниження повторюваності результату трансформації. Використання LLM узгоджується з режимом алгоритму, параметром дозволу на запити та додатковими обмеженнями середовища виконання, що контролює моменти активації генеративного рівня. Деталі двох конфігурацій LLM та їхньої інтеграції у контур спрощення подано далі у підрозділі.

Таблиця 2.17 фіксує політику застосування трансформацій. Базова роль детермінованого рівня забезпечує повторюваність результатів та накопичення знань у словниках зі зниженням варіативності виходу і спрощенням процедури валідації. Окремі політики для термінів та абревіатур знижують ризик семантичного зсуву через переважну безпечність пояснення над підбором синоніма у технічних текстах та залежність розкриття скорочення від контексту.

Таблиця 2.17

Джерела кандидатів замін і правила застосування за категоріями

Категорія	**Основний механізм**	**Резервний механізм**	**Основні ризики**
Ненормативна лексика, сленг, рідковживані, складні, застарілі	Словник замін за лемою і словоформами	LLM за наявності дозволу на запити	Семантичний зсув та граматична некоректність
Абревіатури	Словник розкриття або заміни, застосування без морфологічного узгодження для багатослівних результатів	LLM за наявності дозволу на запити	Надлишкова довжина та некоректне розкриття без контексту
Терміни	Офлайн пояснення з локального словника	LLM для переформулювання фрагмента за окремими правилами	Втрата фахової точності або дублювання пояснення

Генеративні моделі формують кандидатів замін з подальшою фільтрацією, валідацією та трансформацією підтверджених результатів у записи словника для повторного використання. Відокремлення етапу генерації від етапу прийняття

рішення знижує вплив стохастичності LLM на підсумковий результат і узгоджується з вимогою відтворюваності методу.

Метод формулюється як параметризований алгоритм для забезпечення відтворюваності. Параметри визначають деталізацію оцінювання, правила ранжування проблем, агресивність замін, обмеження на перебудову синтаксису, пороги зупинки та політику використання LLM. Зміна значень параметрів безпосередньо впливає на поведінку системи у режимі виконання.

Параметри описуються як елементи формальної конфігурації методу без зведення до технічних деталей реалізації. Параметри методу регулюють компроміс між трьома цілями системи. Перша ціль – приріст когнітивної доступності, друга – стабільність процедури спрощення з передбачуваністю дій та обмеженою частотою перемикань режимів, третя – обмеження ризику спотворення змісту та збільшення когнітивного навантаження через надлишкові зміни тексту. Параметризація відповідає налаштуванню регулятора у керованій системі через збереження цілі методу зі впливом на характер її досягнення.

Синтаксичне спрощення належить до типів трансформацій з підвищеним когнітивним ризиком через прямий вплив зміни акцентів та логічних зв'язків у тексті на побудову смислової моделі тексту читачем. Метод обмежує кількість ітерацій синтаксичного спрощення та використовує поріг мінімального покращення для усунення дрібних переписувань без компенсації когнітивної вартості переходу до нової структури речення. Часті зміни синтаксичної організації тексту ускладнюють зіставлення версій під час читання, підвищують навантаження на робочу пам'ять та послаблюють зв'язність сприйняття навіть за невеликого приросту формальної метрики з відповідними вимогами до обмежень синтаксичних трансформацій у задачах когнітивної доступності.

Аналогічну роль обмежувача ризикованих трансформацій виконують параметри використання LLM, що керують активацією генеративного рівня, вибором моделі та надійністю запитів до зовнішніх клієнтів. Параметри великих мовних моделей узгоджуються з політикою обережності та режимом користувача. У режимі обережності метод віддає перевагу детермінованим

перетворенням з активацією великої мовної моделі лише за відсутності безпечних словникових кандидатів. Нейтральний стан допускає ширшу активацію генеративного резервного рівня за умови фіксованих правил фільтрації та перетворення успішних результатів у словникові записи для повторного використання.

До параметрів надійності методу належать обмеження часу очікування та повторні спроби з впливом на якість взаємодії з користувачем та включенням до системи керування методом. Збільшення кількості повторних спроб підвищує затримку та порушує інтерактивність, скорочення часу очікування підвищує частку порожніх відповідей та знижує ефективність резервного рівня. Параметри надійності фіксуються у протоколах експериментів разом з параметрами якості та узгоджуються з вимогами до часу відповіді у різних режимах деталізації.

Правила зупинки запобігають нескінченним та неефективним ітераціям алгоритму. У системі керування методом правила зупинки виступають елементом безпеки через визначення моменту припинення змін для уникнення зростання когнітивного навантаження від надлишкового редагування тексту. Правила зупинки у методі залежать від режиму роботи, доступності детекцій та наявності значущого приросту якості тексту. Правило зупинки одночасно обмежує обчислювальні витрати методу та захищає текст від послідовності малих перетворень з низьким внеском у якість і ризиком зниження передбачуваності результату (2.29).

$$\begin{aligned} &STOP = 1, \text{ якщо } n \geq n\_max, \text{ або } \Delta ІКД\_n < \varepsilon, \text{ або } |B\_n| = 0, \\ &STOP = 0, \text{ інакше,} \end{aligned} \tag{2.29}$$

де n – поточна ітерація,

n_max – ліміт ітерацій,

ε – мінімальний допустимий приріст ІКД,

|B_n| – кількість виявлених бар’єрів на кроці n.

Таблиця 2.18 представляє правило зупинки методу як багатокомпонентну конструкцію без зведення до одного ліміту ітерацій. Поєднання критеріїв розрізняє ситуації технічної необхідності зупинки та ситуації небажаності

подальших змін з позиції корисності або ризику. Критерії зупинки виступають формалізованим елементом політики безпечного застосування методу у задачах когнітивної доступності.

Таблиця 2.18

Критерії зупинки ітераційного покращення тексту

Критерій	**Інтерпретація**	**Вплив на результат**
Досягнення ліміту ітерацій	Ітерації обмежені параметром steps	Гарантує завершення процедури за обмежений час
Відсутність кандидатів	Множина C порожня або всі кандидати відхилені	Запобігає формальним змінам без детекції бар'єрів
Недостатній приріст	Приріст показника якості нижчий за заданий поріг	Запобігає малим перетворенням з низьким внеском у якість
Ризик перевищує поріг	Оцінки ризику семантичного зсуву або граматичної некоректності перевищують допустимі значення	Запобігає деградації змісту та стилю
Режим обережності	Контекст k вказує на потребу мінімізувати зміни	Підсилює консервативні політики та знижує інтенсивність замін

Узагальнення компонентів методу з урахуванням двох інформаційних потоків, ітераційної схеми, параметрів керування та критеріїв зупинки подано на рисунку 2.8. Схема відображає взаємозв'язок між діагностичним потоком ІКД, контекстним потоком стану користувача, етапами обробки тексту та правилами безпеки трансформацій у єдину архітектуру методу забезпечення когнітивної доступності. Загрози валідності методу пов'язані з доменною залежністю лексичних детекторів та словників, варіативністю генеративних моделей та залежністю від зовнішньої інфраструктури. Окремим фактором виступає вибір шкал експертного оцінювання. Бінарна шкала «валідно або невалідно» призводить до незбалансованості класів та робить окремі показники узгодженості нестабільними. Метод поєднує декілька форм оцінки з фіксацією протоколів експериментів для зниження неоднозначності інтерпретації результатів. Відтворюваність висновків у задачах інженерії програмного забезпечення залежить від строгості експериментального протоколу разом з якістю коду реалізації.

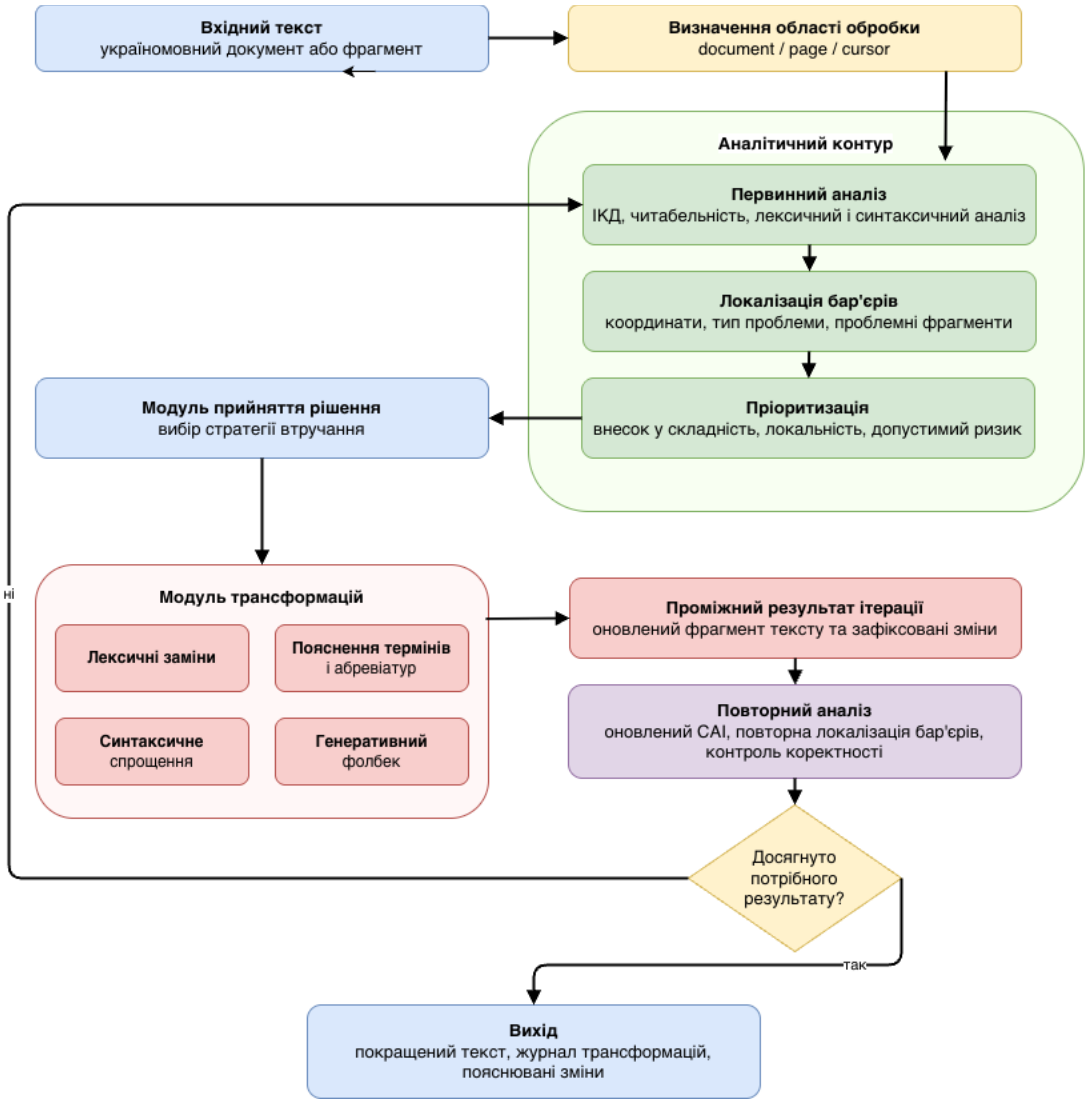


Рис. 2.8. Узагальнена функціональна модель методу забезпечення когнітивної доступності тексту.

Узагальнений метод забезпечення когнітивної доступності побудовано як керований алгоритм ітераційного перетворення тексту з поєднанням детекції бар'єрів та застосування трансформацій у єдиний процес. Метод описує режими деталізації оцінювання на рівні сторінки, документа та інтерактивного курсорного режиму, де локалізація найближчого бар'єра підтримується віконною семантикою та пріоритизацією кандидатів. Індекс когнітивної доступності формалізовано як інтегральний показник для вимірювання якості тексту та навігації подальших трансформацій.

Висновки до розділу 2

У другому розділі «Методи забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями» запропоновано індекс когнітивної доступності тексту, розроблено метод визначення стабільної емоції користувача, сформовано спеціалізовані словники синонімічних замін, побудовано узагальнений метод забезпечення вебдоступності тексту як керований ітераційний процес.

Обґрунтовано недостатність класичних метрик читабельності для оцінювання когнітивної доступності україномовних текстів через мовну непридатність формул та діагностичну неповноту щодо специфічних когнітивних бар'єрів цільової аудиторії. Запропоновано індекс когнітивної доступності тексту (ІКД) як інтегральний нормований показник у діапазоні [0; 1] з поєднанням лексичного, синтаксичного компонентів та компонента читабельності. Виконано перекалібрування формул Flesch-Kincaid, SMOG та D-Level для української мови методом найменших квадратів на корпусі Brown-UK (88 фрагментів, $R^2 \geq 0{,}995$). Незалежна валідація на 30 фрагментах шести жанрів підтвердила коректність жанрового впорядкування ($\rho = 1{,}0$). Вагові коефіцієнти компонентів ІКД визначено методом безпосереднього оцінювання групою з 10 фахівців. Розроблено п'ятирівневу шкалу інтерпретації значень ІКД для підтримки прийняття рішень щодо трансформацій тексту.

Розроблено метод визначення стабільної емоції розуміння тексту користувачем як зовнішнього вхідного показника адаптивного спрощення. Введено поняття стійкої емоції як стабілізованого у часі класифікаційного рішення на основі експоненційного затухання з двоканальною архітектурою спостереження – короткостроковим каналом для оперативної реакції та довгостроковим каналом для підтвердження тенденцій. Запроваджено правило негативного консенсусу між каналами для переходу до допоміжного режиму та визначено протокол калібрування параметрів стабілізації на основі телеметрії.

Розроблено спеціалізовані словники синонімічних замін для когнітивної адаптації україномовного тексту з багатокатегоріальною структурою, що

охоплює рідковживану, складну, застарілу, сленгову та ненормативну лексику, абревіатури і терміни. Розроблено керовану багатоетапну процедуру формування словників з трирівневим контролем якості. Інтеграцію словників у метод побудовано за принципом пріоритетної роботи словникового рівня з контрольованим переходом до генеративних моделей.

Побудовано узагальнений метод забезпечення вебдоступності тексту як керований параметризований ітераційний процес з формалізованим функціоналом корисності J, що поєднує приріст когнітивної доступності зі штрафними доданками за зміну довжини тексту, щільність правок та ризик семантичного зсуву. Метод реалізує три режими деталізації обробки з пакетною детекцією бар'єрів семи категорій та гібридною архітектурою замін – детермінованим словниковим рівнем як базовим механізмом та резервним генеративним рівнем на основі великих мовних моделей. Формалізовано багатокомпонентне правило зупинки з урахуванням ліміту ітерацій, відсутності варіантів, недостатнього приросту якості та режиму обережності.

Матеріали розділу 2 подано в таких публікаціях автора [2, 5, 8, 11, 14, 16, 37].

РОЗДІЛ 3. Програмна реалізація забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями

3.1. Алгоритми та програмна реалізація забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями

Реалізація алгоритму складається з п'яти послідовних функціональних етапів (рис. 3.1).

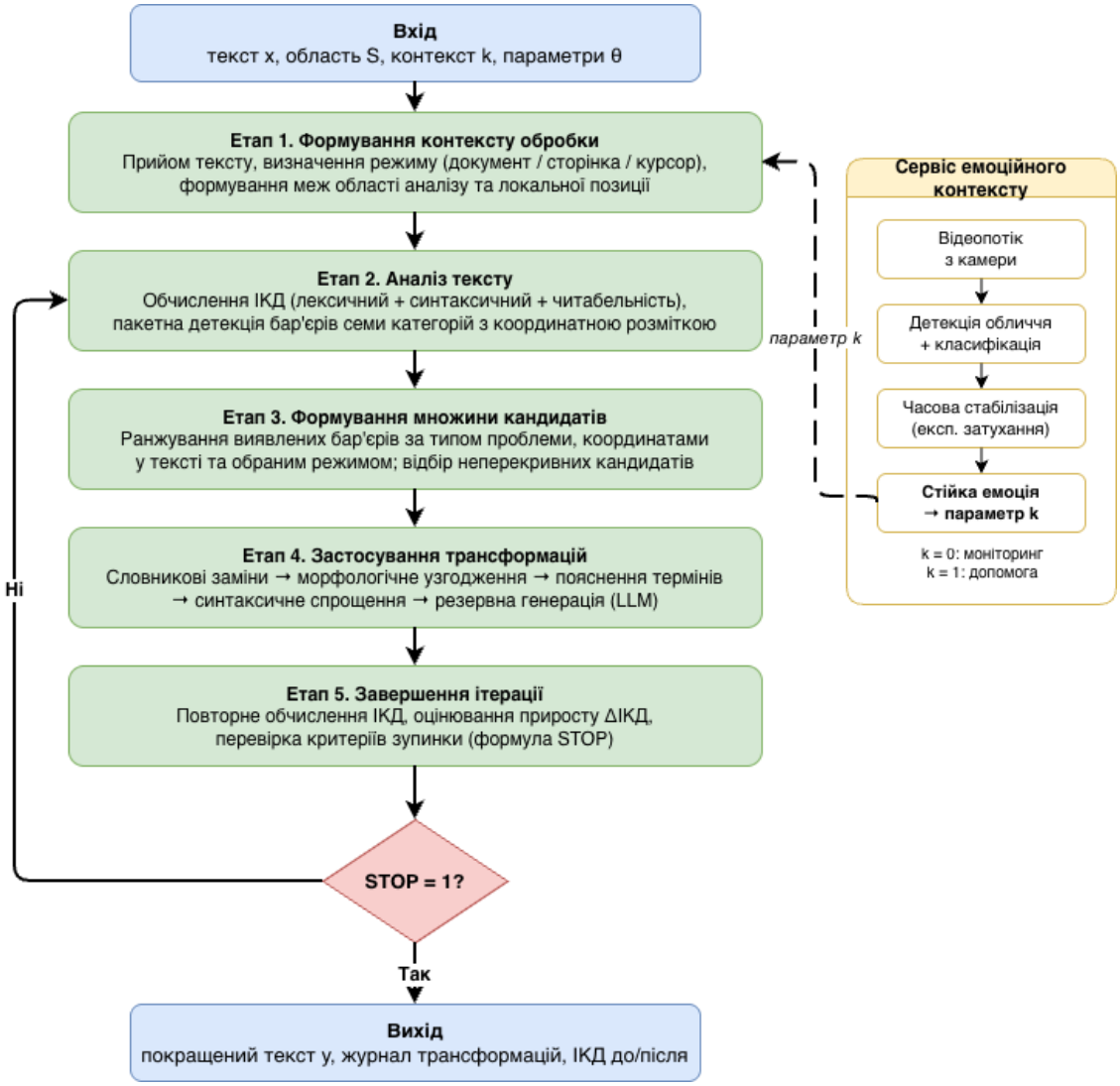


Рис. 3.1. Послідовність функціональних етапів алгоритму.

Етап формування контексту обробки приймає від користувача текст, режим роботи та локальну позицію у документі за потреби. Етап аналізу тексту обчислює показники когнітивної доступності та виявляє проблемні фрагменти. Етап формування множини кандидатів для трансформації ранжує знайдені

варіанти за типом бар'єра, координатами у тексті та обраним режимом спрощення. Етап застосування трансформацій реалізує допустимі лексичні, пояснювальні, синтаксичні або генеративні зміни тексту. Етап завершення ітерації виконує повторне оцінювання результату з перевіркою критеріїв зупинки та завершує процедуру або переводить текст у наступну ітерацію. Сервіс емоційного контексту функціонує паралельно з основним текстовим механізмом та формує стабілізований емоційний показник користувача для допоміжної взаємодії без безпосереднього втручання у текстові трансформації.

Програмна реалізація методу формує алгоритмічну систему з підпорядкуванням архітектури комплексу виконанню конкретних кроків узагальненого методу. Архітектурні рішення підрозділу забезпечують керованість алгоритму, відтворюваність результатів, розмежування відповідальності між підсистемами та підтримку різних режимів спрощення тексту.

Програмна реалізація алгоритму валідується через функціональні результати разом з об'єктивним вимірюванням витрат виконання та перевіркою інтерактивності у різних режимах роботи. Серверний модуль реалізує інструментування часу виконання детекторів та ітерацій з поверненням. Клієнтська частина відображає виміряні значення як діагностичний індикатор, що забезпечує відтворюваність аналізу продуктивності та пошуку джерел затримок без залежності від суб'єктивної оцінки швидкодії. Така схема підтверджує керованість алгоритму через локалізацію витрат на рівні окремих детекторів, ітерацій та підсистем без зміни користувацького протоколу взаємодії.

Рисунок 3.2 демонструє багаторівневу архітектуру програмного комплексу як архітектуру виконання алгоритму. Клієнтський рівень формує контекст запуску та координує взаємодію між підсистемами, шар API-контрактів фіксує межі передавання параметрів та результатів, сервіс аналізу та спрощення реалізує основний ітераційний контур методу, сервіс емоційного контексту функціонує як незалежний допоміжний мікросервіс.

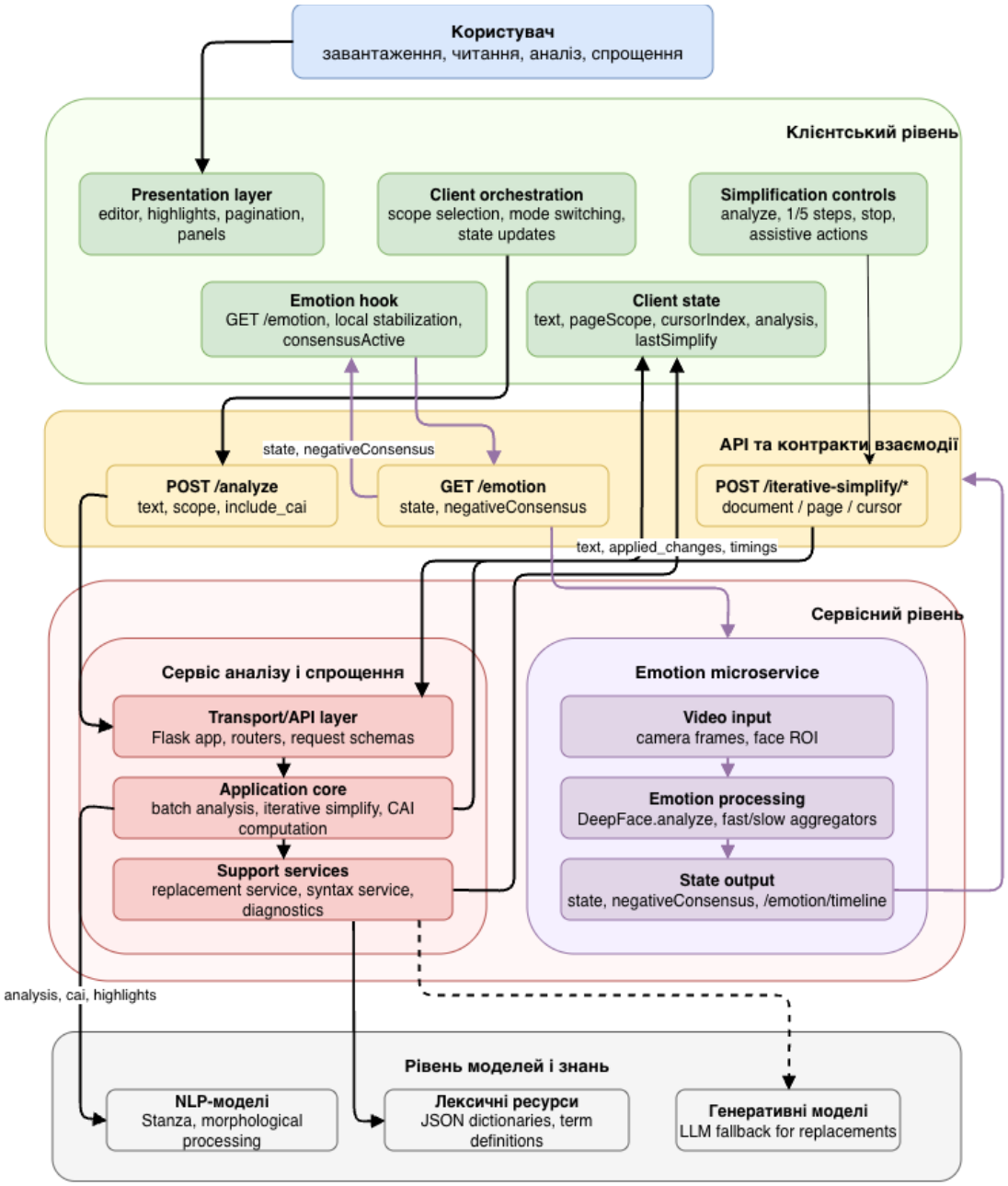


Рис. 3.2. Архітектура комплексу та потоків даних між підсистемами.

Ресурсно-модельний шар забезпечує обидва серверні контури лінгвістичними ресурсами, моделями та параметризованими механізмами генерації, наскрізна діагностична телеметрія підтримує відтворюваність та контроль поведінки системи.

Реалізація алгоритму у репозиторії організована як набір компонентів, що відповідають архітектурним рівням. Клієнтський контур реалізовано як вебзастосунок з точками інтеграції у компонентах редактора та управління станом. Серверний контур реалізовано як Flask-застосунок з реєстрацією

маршрутів у alhoritm/app.py та логікою пакетного аналізу й ітераційного спрощення у alhoritm/routers/analyze.py та alhoritm/routers/iterative_simplify.py. Сервіс емоційного контексту реалізовано як окремий Flask-сервіс у emotion-api/emotion.py для обробки відеопотоку та повернення стабілізованого стану. Контракти даних у клієнтській частині формалізовано типовими структурами у types/api.ts для підтримки узгодженості форматів на рівні компіляції та зниження імовірності порушення інтеграції (табл. 3.1). Технологічний стек комплексу сформовано з урахуванням вимог алгоритму до обробки україномовного тексту, роботи з відеопотоком у реальному часі та підтримки інтерактивного вебінтерфейсу (додаток А, табл. 3).

Таблиця 3.1

Компоненти програмної реалізації алгоритму, їх ролі та інтерфейси взаємодії

Архітектурний компонент	**Роль у системі**	**Внутрішня структура та відповідальність**	**Інтерфейси взаємодії**
Клієнтський рівень	Представлення тексту та взаємодія з користувачем	Рівень представлення, клієнтська координація, локальний стан, вибір режимів документа, сторінки, курсора, активація асистивного інтерфейсу	Editor, AppStateContext, SettingsPanel, SimplificationWidget, useFetchEmotion
Шар API-контрактів	Фіксація меж обміну даними між клієнтом та сервісами	Стабільні формати запитів та відповідей для аналізу, спрощення та емоційного каналу	POST /analyze, POST /iterative-simplify/*, GET /emotion, типи react-frontend-app/src/types/api.ts
Сервіс аналізу та спрощення	Реалізація основного алгоритмічного контуру	Транспортний рівень API на Flask, пакетний аналіз, обчислення ІКД, ітераційне спрощення, допоміжні сервіси для замін та синтаксису	alhoritm/app.py, routers/analyze.py, routers/iterative_simplify.py, services/*, utils/replacement_service.py
Сервіс емоційного контексту	Формування стабілізованого керуючого параметра стану користувача	Захоплення відеопотоку, детекція обличчя, аналіз емоцій, короткостроковий та довгостроковий агрегатори, negativeConsensus, журнал часової динаміки	emotion-api/emotion.py, GET /emotion, GET /emotion/timeline

продовж. табл. 3.1

Рівень моделей та знань	Надання лінгвістичних, словникових та генеративних ресурсів	NLP-моделі, морфологічний аналіз, словники замін, термінологічні ресурси, генеративний резервний рівень	alhoritm/nlp, alhoritm/data/replacements, alhoritm/data/source, pymorphy2, Stanza, LLM clients
Наскрізне спостереження	Підтримка вимірюваності, трасування та відтворюваності	timings_ms, detector_timings_ms, журнал емоційного стану, журнали трансформацій та паспорти запусків	відповіді iterative-simplify, /emotion/timeline, клієнтська діагностика

Обчислення індексу когнітивної доступності виконується у методі з використанням NLP-конвеєра, стабілізований показник стану користувача формується в окремому мікросервісі emotion-api, ітераційне покращення тексту виконується у компоненті iterative_simplify, словниковий шар реалізовано через ReplacementService з лемною адресацією та морфологічним узгодженням. Кожен теоретичний компонент методу має відповідну програмну реалізацію у межах комплексу.

У програмному комплексі текст виступає керованим об'єктом з відтворюваними, інтерпретованими трансформаціями, прив'язаними до конкретної координати у вихідному рядку. Реалізація використовує передачу абсолютних індексів та журнал трансформацій як частину результату обробки.

Маршрутизація реалізована через REST API з трьома маршрутами: POST /analyze (пакетна детекція бар'єрів), POST /iterative-simplify (ітераційне спрощення) та GET /emotion (стабілізований стан користувача). Контракти типізовано у TypeScript для перевірки на рівні компіляції.

Модульна композиція комплексу базується на трьох інженерних механізмах підтримки незалежного розвитку компонентів. Контейнеризація через Docker фіксує версії середовища виконання для кожного сервісу окремо з відтворюваністю експериментів на різних машинах та ізоляцією залежностей. Версіонування API-контрактів через типізовані структури react-frontend-

app/src/types/api.ts та схеми OpenAPI на серверній стороні підтримує сумісність клієнта з різними версіями сервісів через перевірку на рівні компіляції TypeScript. Окреме розгортання сервісу emotion-api на нестандартному порту підтримує запуск множини паралельних конфігурацій модуля з різними бекендами комп'ютерного зору (DeepFace з VGG-Face, Facenet, OpenFace). Така організація комплексу зменшує час циклу експериментальної ітерації та спрощує атрибуцію змін якості до конкретного компонента системи.

Клієнтська частина організує взаємодію з користувачем та формує точне представлення тексту для алгоритмічних процедур через синхронізацію візуального стану з координатами серверного модуля. Редактор, посторінкова навігація, підсвітка проблем та результат спрощення оперують єдиним набором індексів для збереження керованості процесу спрощення.

Реалізація редактора використовує принцип мінімального перетворення тексту перед передаванням даних на серверний рівень. Редактор зберігає змістовну структуру тексту у клієнті без модифікацій, формуючи лише метадані для серверного запиту. Такий підхід знижує ризик розбіжностей між клієнтським та серверним представленням тексту у курсорному режимі, спрощує відтворюваність експериментів та забезпечує коректний аналіз журналів трансформацій (рис. 3.3).

Метадані серверного запиту формуються з полів клієнтського стану, що зберігають поточну позицію користувача у тексті, межі області обробки та результати попередніх запусків алгоритму.

Клієнтський інтерфейс підтримує керовані способи навігації та контекстної взаємодії з мінімальним навантаженням на користувача. Реалізація використовує два взаємодоповнюючі механізми: посторінкову навігацію без прокрутки та навігацію за словами у межах сторінки. Перший механізм знижує когнітивне навантаження через обмеження області сприйняття до однієї сторінки тексту, другий формує точні індекси та межі області для серверного спрощення у сторінковому та курсорному режимах.

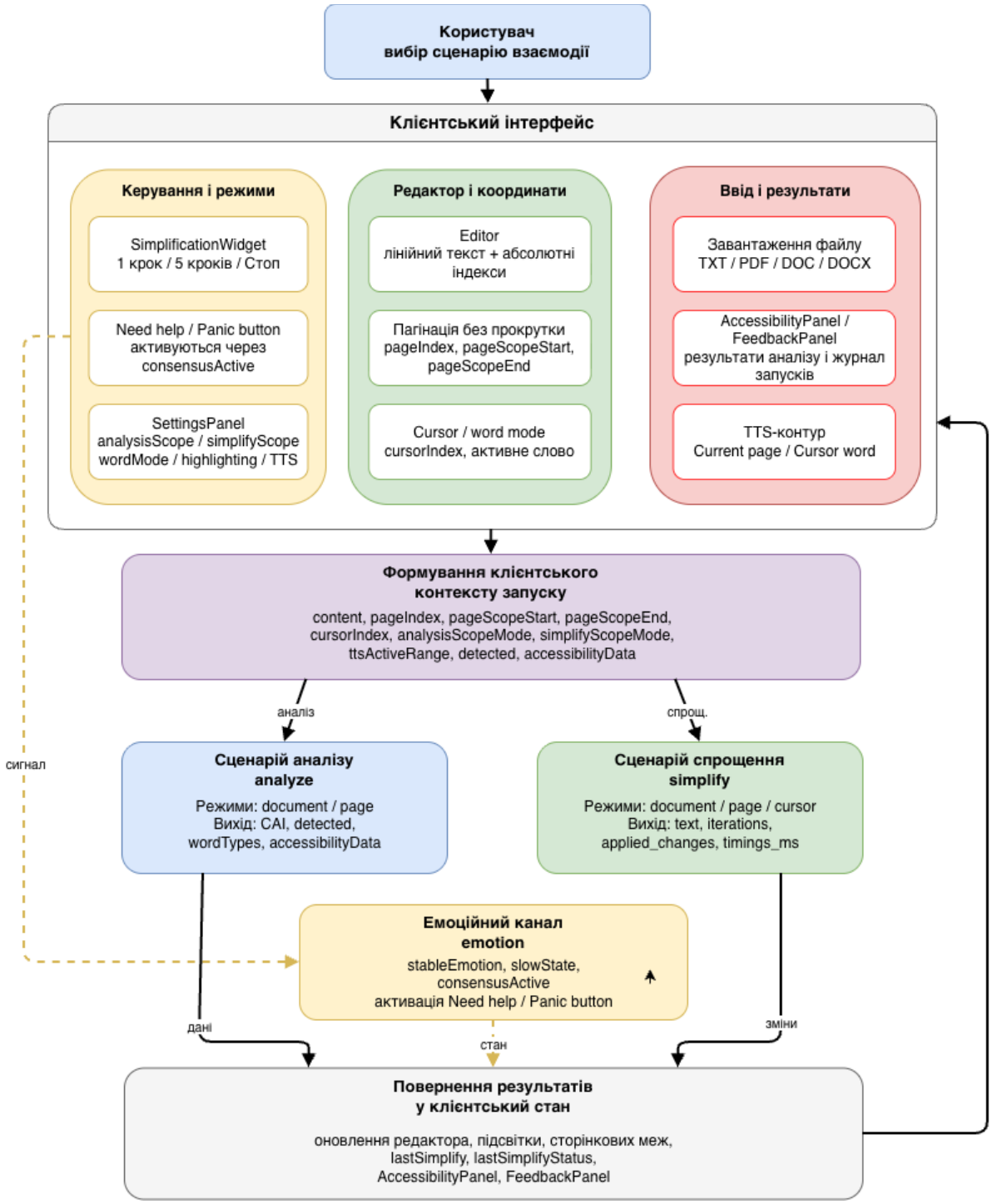


Рис. 3.3. Екранні сценарії клієнтської взаємодії та взаємозв'язок режимів аналізу і спрощення.

Підсвітка проблемних фрагментів застосовується на рівні окремих слів за абсолютними координатами з фіксованим порядком пріоритету категорій. У режимі високої контрастності використовується контурний індикатор замість колірного.

Сторінка у такому режимі виступає не лише візуальною одиницею навігації, а керованим діапазоном індексів вихідного рядка, що передається серверному модулю як межі області аналізу. Узгодження візуального представлення з алгоритмічним контуром через діапазон індексів забезпечує однакову логіку детекції та замін для документного, сторінкового та курсорного режимів зі зміною лише меж області обробки. Документний режим охоплює весь текст, сторінковий обмежується поточною сторінкою, курсорний працює з правою частиною від позиції користувача (рис. 3.4).

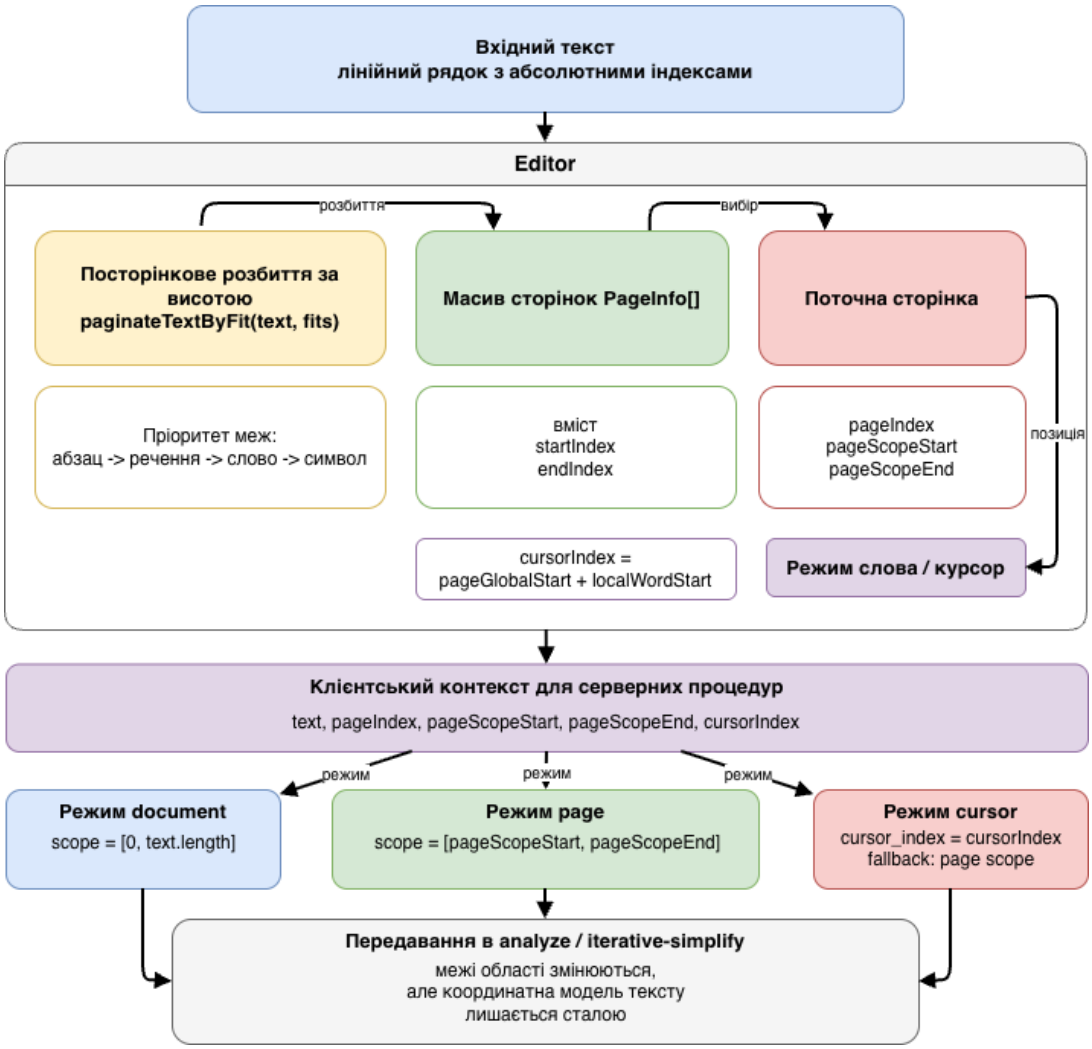


Рис. 3.4. Механізм формування сторінок та відповідність межам області аналізу.

У проєкті навігація реалізована як операція представлення тексту, що відокремлює задачі взаємодії від задач аналізу. Сторінка визначається як

фрагмент, що вміщується у видимий блок без прокручування, а не як фіксована кількість символів. На клієнтській стороні будується предикат блоку за висотою фрагмента, після чого текст розділяється на підрядки з пріоритетом меж абзаців і речень через двійковий пошук за позицією кінця фрагмента (src/components/editor/Editor.tsx, src/utils/textPagination.ts). Такий підхід знижує кількість когнітивних переходів при довгому скролі та формує зручну одиницю навігації для користувача.

У курсорному режимі область обробки формується за алгоритмом, що враховує межі речень для збереження лінгвістичної цілісності фрагмента. Від позиції курсора серверний модуль визначає праву межу вікна шляхом пошуку найближчого розділового знаку кінця речення (крапка, знак оклику, знак питання) у межах максимальної ширини вікна 1200 символів. Якщо розділовий знак знайдений, межа встановлюється після нього; якщо ні, використовується жорстка межа 1200 символів. Така схема запобігає розриву речення посередині, що є важливим для коректної роботи синтаксичних детекторів, які аналізують дерево залежностей у межах речення. Аналогічне обмеження у 1200 символів застосовується до синтаксичного етапу окремо, що дозволяє контролювати латентність при інтерактивній взаємодії.

Сервіс емоційного контексту виступає окремим каналом адаптації інтерфейсу до стану користувача. Клієнтська частина отримує стабілізований керуючий параметр через окремий сервіс, що працює незалежно від основного контуру обробки тексту. Параметр не змінює текст напряму, проте впливає на режим обережності та інтерфейсні підказки користувачу. Стабілізований стан замість миттєвої емоції запобігає різким змінам поведінки інтерфейсу та забезпечує умовне перемикання між консервативною та активною політиками асистивної взаємодії.

Емоційний показник стану користувача відображається у клієнті як ненав'язливі рекомендації без блокування дій користувача. Цей підхід зберігає автономію користувача та паралельно пропонує підтримку за зростання когнітивного навантаження під час читання.

Серверна частина реалізує основну логіку комплексу та обчислювальний контур, спільний для різних режимів взаємодії з користувачем. Архітектура серверної частини організована навколо окремих маршрутів для аналізу, детекції та ітераційного спрощення, що формально розділяє етапи обробки та забезпечує повторюваність результатів. Спільне ядро серверного рівня визначає правила ранжування, перетворення та валідації незалежно від режиму виконання, забезпечуючи узгодженість результатів між різними сценаріями обробки.

У серверній реалізації застосовано два структурні шаблони, що визначають організацію залежностей та маршрутизацію запитів. Шаблон Service Locator реалізований через клас ServiceRegistry – централізований реєстр з підтримкою прямої реєстрації екземплярів через метод register_instance та фабричних функцій з відкладеною ініціалізацією через метод register_factory.

Для NLP-конвеєрів застосовано шаблон відкладеної ініціалізації через клас _LazyPipelines. Stanza Pipeline виступає ресурсоємним об'єктом, оскільки завантаження нейромережевих ваг займає кілька секунд та потребує значного обсягу оперативної пам'яті. Ініціалізація конвеєра відкладається до першого запиту та виконується одноразово з кешуванням результату у словнику за ключем рядка processors під захистом threading.Lock. У разі завершення ініціалізації з помилкою функція повертає None, а детектори переходять у деградований режим з безпечними значеннями за замовчуванням, що запобігає відмові сервісу за відсутності моделей у середовищі виконання. Аналогічний підхід застосовано до NER-моделі бібліотеки HuggingFace, де функція get_ner_pipe() перевіряє глобальну змінну та ініціалізує модель лише за першим зверненням з двоступеневим ланцюжком резервних варіантів.

Stanza виступає одним з небагатьох NLP-фреймворків з нейромережевою моделлю для української мови, навченою на корпусах Universal Dependencies для морфологічного та синтаксичного аналізу. Альтернативні бібліотеки NLTK та spaCy не підтримують українську мову на рівні повного синтаксичного розбору або мають вужче морфологічне покриття. Уніфікований інтерфейс Stanza повертає результат обробки у вигляді структурованих об'єктів з полями lemma,

upos, feats та deprel для потреб детекторів серверного комплексу.

Конфігурація конвеєра з рядком процесорів tokenize, mwt, pos, lemma використовується у лексичних детекторах та морфологічному аналізаторі для розбиття тексту на речення та токени, розгортання складених токенів, визначення частини мови за Universal POS-теґами, визначення морфологічних ознак (відмінок, число, рід, вид) та лематизації слів.

Лексичний компонент ІКД спирається на результати лематизації Stanza. Поле lemma кожного токена порівнюється з шістьма словниками лексичних категорій (ненормативна лексика, застаріла лексика, складна лексика, рідковживана лексика, сленг, абревіатури). Використання леми замість словоформи скорочує розмір словників та забезпечує покриття всіх відмінюваних форм.

Лексичні детектори використовують точний збіг за лемою/словоформою. Для ненормативної лексики додатково застосовується нечіткий пошук (thefuzz, відстань Левенштейна, поріг 85%) для виявлення навмисних спотворень написання.

Дерево залежностей Stanza використовується для аналізу структурної складності речень у синтаксичному компоненті ІКД. Зі стандарту Universal Dependencies вибираються мітки підрядних конструкцій. Глибина дерева залежностей вимірюється рекурсивним обходом від кореня як кількісний показник синтаксичної вкладеності. Метрики нормуються відносно цільових значень довжини та варіативності речень з подальшим включенням до синтаксичного компонента інтегрального індексу ІКД.

Виявлення технічних та фахових термінів реалізовано через нейромережеву NER-модель на основі архітектури RoBERTa з моделлю RomanSavitskyi/ukr-term-detection, навченою на корпусі українських фахових текстів з BIO-анотацією. У разі недоступності хмарної моделі застосовується двоступеневий резервний механізм з локальною моделлю та базовою публічною моделлю youscan/ukr-roberta-base. Для прийняття рішення застосовується адаптивний поріг впевненості зі стандартним значенням 0.55 та підвищеним

значенням 0.85 для токенів без ознак іменника або складеного терміна.

Зниження кількості хибних спрацьовувань NER-моделі забезпечується чорними списками власних назв (близько 50 записів) та загальновживаних слів (близько 100 записів), а також морфологічними евристиками з перевіркою характерних суфіксів (-ація, -ізація, -ування, -ення, -ання, -ість, -ність, -ство), префіксів (авто-, мікро-, макро-, нано-, гіпер-) та компонентних слів (система, модель, алгоритм, метод, технологія).

Словниковий та нейромережевий методи використовуються разом для покриття різних типів термінологічного матеріалу. Словниковий детектор забезпечує детерміновані результати для відомих термінів, NER-модель покриває нові, складені та контекстно залежні терміни. У разі перетину координат двох детекторів пріоритет надається словниковому входженню як детермінованому.

Обчислення інтегрального індексу когнітивної доступності ІКД реалізовано у функції calculate_cognitive_accessibility_index() з об'єднанням трьох незалежно обчислених компонентів за фіксованими ваговими коефіцієнтами w_1, w_2, w_3, встановленими експертним оцінюванням. Лексичний компонент обчислюється як добуток коефіцієнтів доступності для кожної із семи категорій бар'єрів за формулою 3.1.

$$Lexical = \prod(1 - r_k), \quad (3.1)$$

де r_k позначає частку слів відповідної категорії k у загальній кількості слів тексту.

Мультиплікативна схема обрана замість адитивної через сильніше зниження доступності при поєднанні кількох категорій бар'єрів одночасно порівняно з впливом кожної категорії окремо. Синтаксичний компонент виступає зваженою сумою чотирьох факторів. Структура речень з вагою 0.35 враховує штраф за перевищення цільової довжини 25 слів та варіативності 12 слів. Підрядна складність з вагою 0.35 обчислюється на основі щільності підрядних зв'язків за стандартом Universal Dependencies та лексичних маркерів. Глибина дерева залежностей з вагою 0.20 нормується до максимального

значення 12. Частка речень зі структурою «підмет-присудок-додаток» отримує вагу 0.10. Компонент читабельності обчислюється як добуток двох штрафних коефіцієнтів за середньою довжиною слова та середньою довжиною речення. Нормуючі константи 8 для довжини слова та 20 для довжини речення відповідають граничним значенням, за якими текст стає практично недоступним для цільової аудиторії.

Результат функції містить числове значення ІКД, категорійну інтерпретацію за порогами та компоненти зі складовими факторами та вагами. Такий формат відповіді показує користувачу підсумкову оцінку разом з конкретними причинами зниження доступності тексту для подальшого редагування. Підсистема замін реалізує застосування трансформацій на основі результатів детекції бар'єрів. Лексичний рівень спирається на словникові ресурси та механізми морфологічного узгодження для забезпечення граматичної коректності замінених форм. Синтаксичний рівень виконується після лексичних змін зі спрямуванням на зменшення структурної складності речень. Така послідовність знижує ризик руйнування контексту через виконання синтаксичних перетворень після стабілізації лексичного складу тексту.

Запуск синтаксичного етапу контролюється параметрами та умовами активації для запобігання надмірним змінам у тексті та збереження стильової цілісності. Синтаксичні трансформації виступають найбільш ризикованим видом перетворень з обмеженим застосуванням під окремим контролем у серверній логіці.

Параметри ітераційного циклу розподіляються на дві групи. Параметри steps та max_changes належать до групи масштабу змін з прямим впливом на латентність процедури та сприйняття стабільності користувачем у курсорному режимі. Параметри confidence_threshold та strict_pos утворюють групу точності з керуванням кількістю допущених до перетворень варіантів та рівнем консервативності морфологічної перевірки. Параметр allow_llm переводить метод у гібридний режим зі значним впливом на варіативність виходу з фіксацією у протоколах експериментів та узгодженням з режимами обережності.

Діапазони та значення за замовчуванням фіксуються у серверній конфігурації з додатковими обмеженнями на рівні обробника запиту для запобігання неконтрольованим запускам. Параметр confidence_threshold має значення за замовчуванням 0.7 та використовується як поріг для ймовірнісних детекторів. Параметр strict_pos активований за замовчуванням для ітераційного спрощення, тоді як значення параметрів steps та max_changes залежать від режиму обробки. Документний режим використовує значення за замовчуванням steps 5 та max_changes 20 з верхніми межами steps 20 та max_changes 50. Сторінковий та курсорний режими використовують значення за замовчуванням steps 5 та max_changes 10 з верхніми межами steps 20 та max_changes 25. Курсорний режим додатково обмежує ширину вікна детекції та синтаксичного етапу максимумом 1200 символів для кожного з вікон зі зниженням латентності під час інтерактивної взаємодії з користувачем.

Курсорний режим виконує повторну детекцію після кожної ітерації через потребу в актуальному стані вузької області обробки після кожної зміни. Документний та сторінковий режими виконують повторну детекцію кожні три ітерації зі зниженням обчислювальних витрат та збереженням актуальності списку кандидатів для великих областей тексту. У разі вичерпання придатних кандидатів до планового моменту повторної детекції виконується позачергова перебудова списку з одноразовою спробою продовження. Ітераційний цикл завершується за критерієм відсутності змін у разі невиявлення нових придатних кандидатів повторною детекцією.

Контракти даних визначають структури обміну між клієнтом та сервером з узгодженим форматом результатів аналізу та спрощення. Протоколи обміну включають штатні відповіді разом з контролем помилок для передбачуваності поведінки за некоректного вхідного запиту або порушення меж параметрів. Серверна частина виконує валідацію типів та діапазонів параметрів з поверненням відмови та поясненням у разі невідповідності. Частина параметрів приводиться до допустимих меж у визначених режимах замість повної відмови для стабілізації системи за інтерактивного використання.

Координатна прив'язка забезпечує відповідність позицій у серверних результатах візуальним фрагментам тексту у клієнтському інтерфейсі через абсолютну індексацію у вихідному рядку. Розбіжність у підрахунку індексів призводить до неправильного редагування тексту, що робить точність координатної прив'язки інженерним інструментом контролю якості замін. Принцип мінімального перетворення на клієнті разом з фіксацією індексів у метаданих запуску підтримує відтворюваність експериментів та локалізацію джерел помилок у разі розбіжностей.

Журнал трансформацій та метадані запусків формують доказовий шар алгоритму. Журнал трансформацій фіксує кожну застосовану зміну з координатами заміни, типом проблеми, джерелом кандидата та фактичним текстом до і після перетворення для експертного аналізу відповідності змін заявленим правилам та збереження семантичної цілісності тексту. Метадані запуску містять режим виконання, фактичні обмеження параметрів, кількість ітерацій та часові характеристики обробки, формуючи запис запуску як основу для порівняння конфігурацій методу у наукових експериментах та практичному контролі якості.

Ітераційне спрощення реалізоване у модулі iterative_simplify.py як цикл з фіксованою послідовністю операцій на кожному кроці. Кожна ітерація передбачає виклик пакетного аналізу для побудови списку кандидатів за поточним станом тексту, ранжування кандидатів за пріоритетом категорії та позицією, перевірку обмежень кожного кандидата та застосування трансформацій у порядку справа-наліво для збереження координат решти кандидатів.

Пріоритизація кандидатів виступає механізмом визначення порядку застосування змін на основі типу проблеми, впливу на когнітивну доступність та позиції у тексті. Такий підхід забезпечує передбачуваність процесу та інтерпретованість послідовності виконаних трансформацій. Пріоритизація виконує інженерну функцію уникнення ситуацій блокування суттєвої заміни менш пріоритетною.

Правки без перекривання меж виступають технічним правилом коректного редагування. Кандидати з перекриванням меж не застосовуються у межах однієї ітерації для запобігання пошкодженню індексів та зниження ризику семантичних помилок. Зміни виконуються послідовно без руйнування структури тексту навіть за близького розташування кандидатів (табл. 3.2).

Ранжування реалізовано як побудова списку кандидатів у фіксованому порядку категорій з подальшим сортуванням за позицією у тексті. Правило відсутності перекривання меж реалізовано через два рівні перевірок з роботою у базовій системі координат останньої детекції та у поточній системі координат з накопиченим зміщенням після застосованих замін. Така комбінація забезпечує застосування кожної зміни до коректного фрагмента тексту з виключенням конфліктних редагувань з одного пакету змін. Фіксований порядок категорій визначає послідовність потрапляння кандидатів до черги застосування у вигляді ненормативної лексики, абревіатур, сленгу, термінів, застарілої лексики, складної лексики та рідковживаної лексики.

Таблиця 3.2

Правила ранжування кандидатів та критерії відбору правок без перекривання меж

Критерій	Роль	Наслідок
Тип проблеми	Порядок обробки	Фіксований пріоритет: ненормативна лексика, абревіатури, сленг, терміни, застаріла лексика, складна лексика, рідковживана лексика
Позиція у тексті	Стабільність та інтерпретованість	Сортування від початку області до кінця в межах однакового типу
Перекриття у базовій області	Коректність вибору	Відсікання кандидатів з накладанням на прийнятий базовий інтервал
Перекриття у поточному тексті	Захист від конф-ліктів після замін	Відсікання кандидатів з накладанням з урахуванням накопичених зміщень
Накопичення зміщення	Узгодження координат	Корекція наступних позицій через різницю довжин фрагментів до та після заміни
Повторна детекція	Узгодження карти проблем	Перебудова списку кандидатів після серії замін за частотою повторної детекції

Синтаксичний етап виконується після лексичних замін зі спрямуванням на зменшення структурної складності речень. Роль етапу полягає у покращенні

читацької зрозумілості без зміни основного змісту тексту. Синтаксичні зміни впливають на стиль та логічні зв'язки тексту, тому етап виконується обмежено під окремим контролем серверної логіки.

Контроль змін реалізується через мінімальний поріг покращення і обмеження кількості ітерацій. Це зменшує ризик надмірної перебудови речень і зберігає стабільність результату. У контексті когнітивної доступності це важливо, оскільки читачу легше сприймати невеликі узгоджені зміни, ніж глибоке переписування структури.

Критерії зупинки визначають момент завершення ітераційного процесу. Серед критеріїв виступають досягнення ліміту ітерацій, відсутність придатних кандидатів, недостатній приріст показників якості та сигнали ризику трансформацій. Критерії оберігають систему від надмірних змін зі збереженням керованості результату спрощення (табл. 3.3).

Таблиця 3.3

Критерії зупинки ітераційного процесу

Критерій	Сенс	Вплив
Обмеження кроків	Гарантія завершення	Кількість виконаних кроків не перевищує steps після обмеження режимом
Обмеження змін у кроці	Контроль інтенсивності	Кількість застосованих змін у пакеті не перевищує max_changes
Відсутність застосованих змін	Завершення без прогресу	Цикл завершується за відсутності придатних замін після формування кандидатів
Одноразова повторна детекція	Захист від хибного нульового пакету	Передбачає перерахунок кандидатів один раз за вичерпання списку без змін
Перевірка меж області	Захист коректності	Некоректні start, end або cursor_index блокують запуск на рівні валідації
Обмеження синтаксичного етапу	Контроль ризиків	Синтаксичне покращення обмежене параметрами syntax.max_iterations та syntax.min_improvement

Лексичні ресурси формують основу детермінованих замін зі зниженням стохастичності результату спрощення. Словники містять основні категорії проблемної лексики як джерело кандидатів для трансформацій. Для кожної категорії сформовано окремий словник з елементами та їх рекомендованими

замінами або поясненнями для незалежного розвитку словників та підтримки різних доменів застосування. Організація словників побудована навколо пошуку за лемою з підтримкою словоформ для зниження кількості дублювань та спрощення супроводу ресурсів (додаток А, табл. 1). Значення у словниках орієнтовані на когнітивну доступність з пріоритетом простоти та зрозумілості замін над лексичною близькістю для забезпечення передбачуваної якості результату.

Пошук замін у словниковому шарі реалізовано як детерміновану ієрархію з чотирьох рівнів з поступовим розширенням охоплення через менш точні стратегії. Перший рівень виконує пряме порівняння вхідного слова з ключами словника. Другий рівень використовує лему від Stanza для пошуку замін відмінюваних форм без дублювання записів у словнику. Третій рівень для категорій ненормативної, рідковживаної лексики, сленгу та абревіатур додатково перевіряє відображення словоформи до канонічної форми для покриття випадків неканонічної форми від лематизатора Stanza. Четвертий рівень використовує попередньо побудований індекс словоформ з відображенням кожної відмінюваної форми з JSON-словника на базовий запис. Записи з прапором isVerify зі значенням false виключаються з результатів для маркування ненадійних замін без видалення зі словника. Ієрархія забезпечує покриття зі збереженням пріоритету точних збігів над евристичними.

Реалізацію морфологічного узгодження забезпечує бібліотека pymorphy2 з українською мовною моделлю, ініціалізована як єдиний екземпляр MorphAnalyzer(lang='uk'). Морфологічні ознаки вихідного токена (відмінок, число, рід, частина мови) отримуються на етапі POS-розмітки Stanza та передаються у функцію генерації форми замінника. pymorphy2 перебирає парадигму словозміни кандидатної леми та повертає словоформу з найближчою відповідністю цільовому граматичному профілю. У разі неможливості згенерувати коректну форму система використовує лему без узгодження для уникнення граматично некоректних правок.

LLM у алгоритмі виступає резервним рівнем генерації для забезпечення

якості та гнучкості замін поза покриттям детермінованого словникового рівня (рис. 3.5). Інтеграція LLM поєднує переваги генеративних моделей з вимогою відтворюваності експериментів. Зменшення повторних викликів реалізовано через повторне використання вже отриманих результатів та переведення успішних пропозицій у детермінований словниковий ресурс. Така схема використовує LLM як джерело розширеного покриття зі зниженням витрат та стабілізацією поведінки системи.

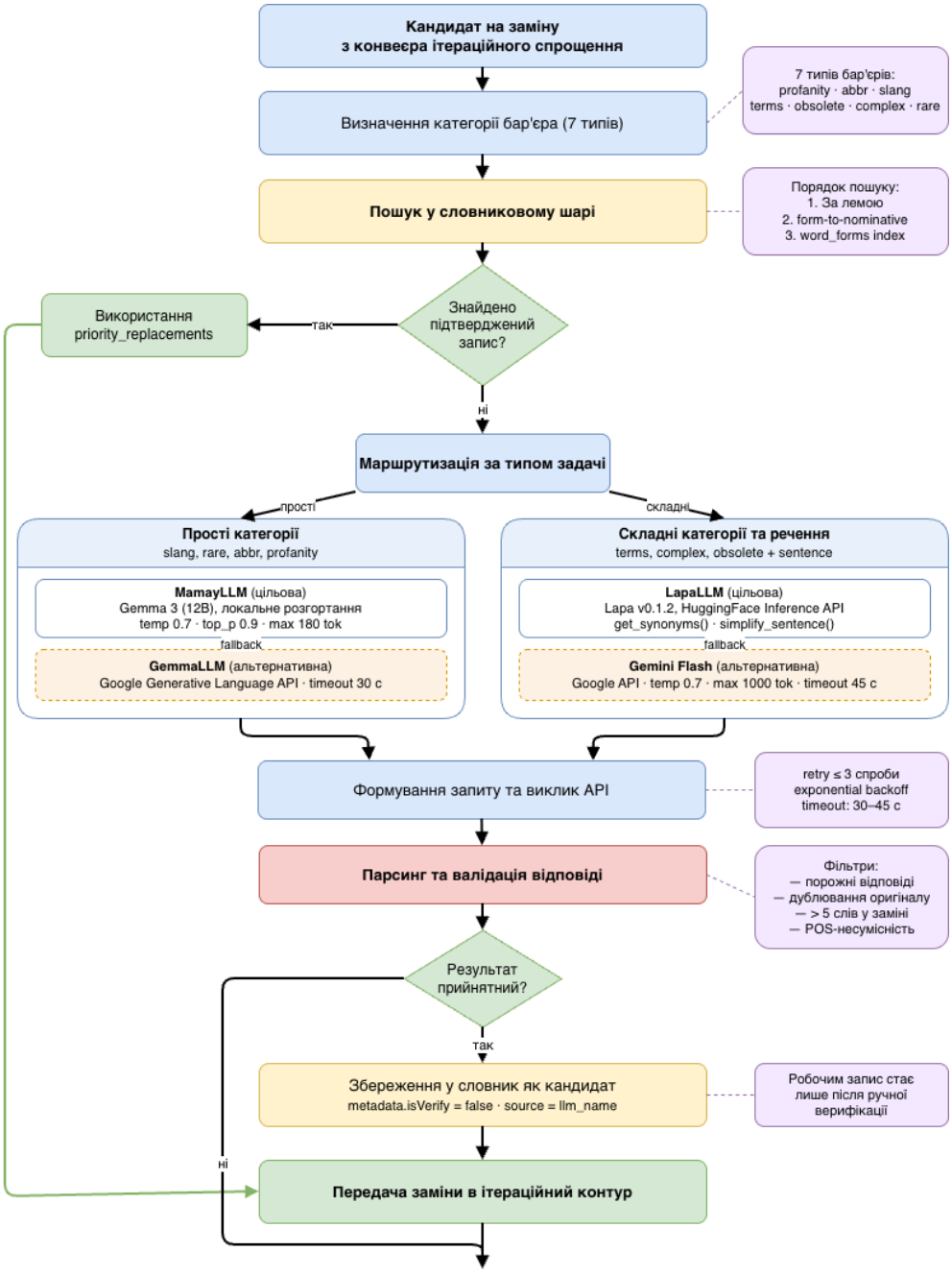


Рис. 3.5. Архітектура інтеграції LLM у контур генерації.

Архітектурно LLM підключається через клієнтський модуль для

формування запиту, передачі параметрів та обробки відповіді з централізованим контролем параметрів генерації та фіксацією версій моделей для відтворюваності. Підхід до інтеграції LLM у комплексі орієнтований на стабільність через найвищий потенціал варіативності генеративного рівня.

Розподіл задач між моделями виконується за складністю лінгвістичного випадку. Прості лексичні заміни використовують моделі зі швидким виконанням та стабільністю коротких відповідей. Складні випадки контекстного переформулювання використовують моделі з більшим контекстним охопленням. Такий розподіл забезпечує баланс між якістю та продуктивністю генерації.

Поділ на дві конфігурації, зафіксований у таблиці 3.4, виконує роль механізму контролю якості. Локальне розгортання цільової конфігурації забезпечує автономність системи зі зниженням залежності від зовнішніх сервісів. Альтернативна конфігурація надає базу для оцінювання чутливості результатів до вибору моделі та збереження працездатності системи при зміні інфраструктури.

Таблиця 3.4

Конфігурації LLM, призначення та роль у гібридній генерації замін

Конфігурація	**Прості категорії**	**Складні категорії**	**Призначення в системі**
Цільова	MamayLLM	LapaLLM	Стабільна якість та контрольований стиль у локальному розгортанні
Альтернативна	Gemma 3 12B	Gemini 1.5 Flash	Порівняння, резервний режим та тимчасова заміна цільових моделей

Клієнтські реалізації зовнішніх моделей використовують явні обмеження часу виконання (30 секунд для Gemma, 45 секунд для Gemini Flash) з механізмом повторів до трьох спроб та експоненційною затримкою для стійкості за мережевих збоїв. У разі вичерпання спроб або повернення моделлю порожнього чи непридатного результату підбір заміни завершується без пропозиції з продовженням ітераційного циклу для інших варіантів.

Після генерації LLM-пропозиція проходить фільтрацію та зберігається як словниковий запис для повторного використання у наступних ітераціях без

повторного виклику моделі. Контроль якості LLM-результатів охоплює відкидання порожніх відповідей, дублювання оригіналу, надмірно довгих варіантів, перевірку сумісності з морфологічними ознаками та правила семантичної адекватності. Критерії відбору та валідації LLM-результатів описані в таблиці 3.5

Таблиця 3.5

Критерії відбору та валідації LLM-результатів

Перевірка	**Опис**	**Ризик**
Дублювання	Відсікання оригіналу	Низька користь
Довжина	Обмеження довжини відповіді	Надмірне розширення
Морфологія	Узгодження форми	Граматична помилка
Мінімальна кількість варіантів	Приймаються лише відповіді з щонайменше двома різними пропозиціями	Випадкова або порожня генерація
Нормалізація кандидатів	Видалення повторів після приведення до нижнього регістру і очищення	Псевдоваріативність
Обмеження за кількістю слів	Відсікання надмірно довгих фраз у синонімах	Втрата лаконічності
Фільтрація за частиною мови	За увімкненого strict_pos відсікаються несумісні заміни	Семантичний зсув
Відмова від небезпечних евристик	Пошук за підрядком вимкнено за замовчуванням	Хибні збіги та заміни не того слова Хибні збіги та заміни не того слова
Фіксація параметрів генерації	Сталі значення temperature та top_p у клієнтах	Непорівнянність експериментів

Емпірична база двох конфігурацій великих мовних моделей описана у попередній публікації автора, де подано гібридну архітектуру з поєднанням словника та LLM з порівнянням моделей за якістю замін, швидкодією, покриттям та ресурсною вартістю [6].

Метод визначення стабільної емоції реалізовано як окремий мікросервіс, що формує стабілізований керуючий параметр для адаптації інтенсивності спрощення.

Потік даних у сервісі розпочинається зі зняття відео з камери та виділення

обличчя з подальшим обчисленням первинних емоційних ймовірностей зі стохастичною природою результату. Стабілізація сигналу реалізована через два паралельні канали спостереження (короткостроковий та довгостроковий) з різними коефіцієнтами затухання та вимогами до кількості кадрів для одночасної фіксації коротких змін та запобігання різким коливанням у підсумковому керуючому параметрі.

Вибір DeepFace як бібліотеки аналізу виразів обличчя обумовлений уніфікованим інтерфейсом до кількох попередньо навчених моделей розпізнавання облич. Серед доступних бекендів обрано VGG-Face – архітектуру глибокої згорткової нейромережі, навченої Oxford Visual Geometry Group на корпусі зображень облич, зі стабільною якістю класифікації мімічних виразів порівняно з FaceNet, OpenFace та ArcFace. Конфігурація бекенду передається через змінну середовища DEEPFACE_MODEL для заміни моделі без зміни коду та відтворюваності конкретної конфігурації у наукових експериментах.

Відеоконвеєр реалізовано в окремому daemon-потоці з запуском під час старту сервісу та безперервним виконанням незалежно від клієнтських запитів. OpenCV відкриває пристрій камери через cv2.VideoCapture(0) з циклічним зчитуванням одного кадру за ітерацію. Кадр перед передачею у детектор обличчя конвертується у відтінки сірого для прискорення роботи каскадного класифікатора Хаара. Виявлені прямокутники облич вирізаються з RGB-версії кадру через очікування DeepFace.analyze() тривимірного RGB-тензора. Параметри детектора (scaleFactor 1.1, minNeighbors 5, minSize 30 на 30 пікселів) обрані як компроміс між швидкістю та точністю. За відсутності виявлених облич у кадрі або непрохідної перевірки площі агрегатори переходять у режим затухання без оновлення.

Проміжною ланкою між виходом DeepFace та підсистемою адаптації виступає матриця відображення семи базових емоцій у простір трьох когнітивних станів – спокійного, дезорієнтованого та перевантаженого. Вибір трьох станів обумовлений безпосереднім зв'язком з інтенсивністю когнітивного навантаження під час читання. Спокійний стан забезпечує активніше спрощення,

дезорієнтований та перевантажений стани активують режим обережності. Ваги перетворення відображають психологічні зв'язки між емоціями та когнітивним навантаженням з систематизацією у додатку А (табл. 4). Вектор ймовірностей DeepFace трансформується у вектор внесків у три когнітивні стани з масштабуванням на нормалізовану впевненість розпізнавання.

Стабілізацію сигналу реалізує клас WeightedLabelAggregator як структура експоненційного ковзного усереднення з порогом активації. Кожне оновлення агрегатора починається з застосування затухання до всіх накопичених оцінок з подальшим видаленням записів з оцінкою нижче порогу для запобігання накопиченню числових залишків. Новий внесок додається за класичною схемою експоненційного ковзного усереднення, де коефіцієнт затухання визначає ефективну довжину пам'яті агрегатора. Стабільна мітка визначається за умови перевищення кількістю спостережень мінімального порогу та перевищення оцінкою найкращої мітки порогу активації. За відсутності обличчя у кадрі застосовується механізм затухання без нових внесків для поступового зниження сигналу без різкого скидання стану агрегатора.

Сервіс підтримує чотири незалежні агрегатори, об'єднані у дві пари – агрегатори емоцій та агрегатори станів. Агрегатори емоцій обробляють первинний вектор виходу DeepFace зі стабілізацією мітки домінантної емоції, агрегатори станів обробляють вектор когнітивних внесків від матриці відображення зі стабілізацією кінцевого когнітивного стану. У кожній парі короткостроковий канал реагує оперативніше з вищим порогом активації, довгостроковий канал стійкіший до коротких шумових сплесків із забезпеченням більшого покриття. Значення параметрів кожного каналу (коефіцієнт затухання α, поріг активації θ та мінімальна кількість спостережень n_{min}) отримано за протоколом програмної калібрування на базі телеметричного корпусу з трьох серій (додаток А, табл. 4). Параметри задаються через змінні середовища без потреби у перекомпіляції для зміни поведінки системи.

Фінальний когнітивний стан формується функцією consensus з об'єднанням виходів короткострокового та довгострокового агрегаторів станів за

правилом негативного консенсусу. Стан вважається негативним (дезорієнтований або перевантажений) за виконання однієї з двох умов: одночасне перебування обох каналів у негативному стані або перебування короткострокового каналу у негативному стані за нейтрального чи невизначеного довгострокового стану. Друга умова реалізує асиметрію реагування для швидшого виявлення початку негативного тренду без очікування підтвердження від довгострокового каналу. Результуючий стан для API визначається як довгостроковий стан або короткостроковий стан за відсутності довгострокового. Індикатор негативного консенсусу використовується підсистемою адаптації для переходу у консервативний режим обережності з обмеженням кількості трансформацій за одну ітерацію.

Сервіс реалізує три режими роботи, що задаються через змінну середовища EMOTION_MODE. Робочий режим запускає фоновий daemon-потік відеоконвеєра. Тестовий режим зчитує заздалегідь підготовлений JSON-файл із записаними послідовностями емоцій з відтворенням за фіксованим інтервалом для відтворюваних тестових запусків без фізичної камери. Вимкнений режим повністю деактивує фоновий потік. Потокобезпечний доступ до глобального стану забезпечується механізмом блокування threading.Lock зі захистом зчитування у відповідях API та запису з відеоконвеєра. Деградація за недоступності камери реалізована пасивно через завершення потоку у разі помилки зчитування кадру з поверненням невизначеного стану через API до перезапуску сервісу. Телеметрія фіксується у кільцевому буфері на 120 записів.

Стабілізація виконує роль фільтрації потоку кадрів та перетворення сигналу в узгоджений стан. Стабілізація знижує шум первинних емоційних ймовірностей та формує основу для дискретних станів використання у політиці обережності. Стабілізований керуючий параметр виступає індикатором для адаптації методу без статусу абсолютної істини зі зниженням ризику неправильного впливу на процес спрощення.

Вибір конкретних значень параметрів реалізовано як керовану процедуру калібрування з використанням телеметрії та серії повторюваних запусків. Сервіс

накопичує часовий ряд спостережень з підтримкою порівняння конфігурацій без змін у коді через параметризацію через змінні середовища. Результат калібрування забезпечує прийнятний рівень стабільності за помірної затримки реакції на зміни емоційного стану користувача.

У межах калібрування виконувалися серії запусків з різними користувачами та у різних умовах зйомки для відображення практичної неоднорідності відеопотоку у вебсередовищі. Для кожної конфігурації змінних середовища формувався журнал телеметрії з аналізом показників, зокрема кількості перемикань стану на умовно сталих фрагментах взаємодії, затримки підтвердження стійкої зміни, частки часу без визначеного стану та узгодженості короткострокового і довгострокового каналів. Процедура виявила дві категорії небажаної поведінки сервісу, серед яких надмірна чутливість з ефектом миготіння та надмірна інерційність із запізнілою адаптацією інтерфейсу (табл. 3.6).

Таблиця 3.6

Критерії калібрування параметрів стабілізації та обґрунтування використання

Критерій	**Інтерпретація у системі**	**Ризик за низької якості**	**Роль параметрів затухання, порога та мінімуму кадрів**
Частота перемикань станів	Кількість змін стану за сталого фрагмента взаємодії	Миготіння інтерфейсу та втрата довіри користувача	Збільшення інерції та порога зменшує перемикання
Затримка реакції на стійку зміну	Час до фіксації нового стану за трендової зміни	Відсутність корисної адаптації	Зменшення інерції або мінімуму кадрів підвищує чутливість
Частота відсутності керуючого параметра	Частка часу з порожнім станом агрегатора	Втрата адаптивності	Зниження порога та підбір мінімуму кадрів підвищує повноту покриття
Стійкість до пропусків обличчя	Поведінка за коротких пауз, поворотів голови, оклюзій	Хибні переходи у нейтральний стан	Затухання визначає швидкість забування та стабілізує канал
Узгодженість двох каналів	Збіг короткострокового та довгострокового каналів на довгих фрагментах	Помилковий консенсус	Різні параметри каналів створюють захист від одиничних сплесків

Клієнт опитує емоційний сервіс кожні 5 секунд без блокування інтерфейсу, зберігаючи останнє відоме значення за збою. Локальна стабілізація визнає значення сталим після 1 секунди. Основний механізм спрощення не залежить від доступності емоційного сервісу.

Телеметрія фіксує часовий ряд станів з підтримкою аналізу динаміки емоцій під час роботи з текстом. Дані телеметрії підтримують експериментальну оцінку впливу змін тексту на стан користувача та виступають доказовим матеріалом для аналізу ефективності адаптивних політик. Структура агрегованого запису телеметрії на рівні події подана у додатку А разом з прикладами значень для кожного поля.

Параметри функціонування закріплені у конфігураціях для відтворюваності експериментів. Модулі перетворення мовлення на текст та інтерактивного аватару не входять до поточної реалізації. Контроль виконання у межах закріплених параметрів реалізується через два принципи оцінювання продуктивності комплексу. Перший принцип полягає у поверненні вимірювань часу у відповіді сервера для побудови протоколів експериментів без додаткового інструментування клієнтського коду (додаток А, табл. 5). Другий принцип полягає в обмеженні навантаження за інтерактивного використання через обмеження області обробки, повторне використання результатів пакетного аналізу та режимні параметри керування інтенсивністю.

Додатковим механізмом оптимізації продуктивності виступає дворівневе кешування результатів аналізу. Серверний рівень реалізує LRU-кеш на 64 записи з ключем кешу як комбінацією меж області, порога впевненості та хешу тексту для уникнення повторного виконання детекторів за незмінного тексту. Клієнтський рівень зберігає результати запитів з часом життя 5 хвилин з механізмом дедуплікації для запобігання надсиланню ідентичних запитів у процесі виконання. Поточна реалізація комплексу має визначені межі застосування за доменом текстів та технічними ресурсами. Алгоритм оптимізований для українських текстів вебсередовища з потребою адаптації та додаткової валідації для інших мов. Окремі категорії лексики мають неповне

охоплення через специфіку домену з обмеженням універсальності системи.

Технічні обмеження включають залежність частини компонентів від локальних моделей та ресурсів з впливом на швидкодію та потребою у контролі середовища виконання. Робота LLM у зовнішніх інфраструктурах допускає варіативність часу відповіді з впливом на аналіз ефективності системи. Фіксація цих обмежень забезпечує коректну інтерпретацію експериментальних результатів.

Програмна реалізація алгоритму ітераційного спрощення тексту побудована як цикл з фіксованою послідовністю операцій формування списку кандидатів, ранжування за пріоритетом семи категорій бар'єрів та позицією у тексті, перевірки правил без перекривання меж та застосування трансформацій у порядку справа-наліво для збереження координат решти кандидатів. Алгоритм підтримує три режими обробки з координатною прив'язкою через абсолютну індексацію у вихідному рядку, формальними критеріями зупинки циклу та повним записом запуску з параметрами, журналом трансформацій та часовими характеристиками для відтворюваності експериментальних протоколів та порівняння конфігурацій між запусками.

3.2. Реалізація адаптивного інтерфейсу та допоміжних підсистем програмного комплексу

Комплекс містить набір допоміжних підсистем для зручності взаємодії, мультимодального доступу до тексту, налаштування середовища під індивідуальні потреби користувача та стійкості системи у разі помилок. Функціональна модель додаткового функціоналу охоплює модуль завантаження та попередньої обробки тексту, візуальний інтерфейс редактора, панель метрик доступності, засоби налаштування параметрів, підсистему синтезу мовлення, режим експертної валідації, механізми обробки помилок та зворотного зв'язку від користувачів.

Загальний вигляд застосунку сформований з урахуванням принципів когнітивної доступності (рис. 3.6). Нейтральний світлий фон з мінімальним

контрастом між функціональними зонами знижує зорове навантаження.



Рис. 3.6. Інтерфейс програмної реалізації алгоритму.

Просторова передбачуваність забезпечується лівою панеллю керування,

центральною областю редактора та правою панеллю інформаційних блоків з відповідністю читацькій траєкторії зліва направо.

Верхня панель застосунку містить назву системи та дві функціональні групи елементів (рис. 3.6 (1)). Перемикач мови підтримує роботу у двох режимах зі зміною текстів інтерфейсу та налаштувань алгоритму аналізу відповідно до обраної мови. Інтернаціоналізація реалізована засобами бібліотеки i18next з окремими файлами перекладів для кожної мови та динамічним перемиканням атрибуту lang кореневого HTML-елементу. Блок відображення поточної емоції виступає елементом діагностичного режиму з показом результату роботи підсистеми емоційного розпізнавання у реальному часі та прихованим станом у штатному режимі роботи.

Завантаження та попередня обробка тексту реалізована через спеціалізований віджет у верхній частині робочої області (рис. 3.6 (8)). Віджет приймає прямий ввід тексту та завантаження файлів поширених форматів. Підтримуються формати PDF, DOC, DOCX та TXT з обмеженням максимального розміру файлу 50 МБ.

Успішне завантаження переводить інтерфейс у стан готовності з відображенням опції заміни файлу та окремого інформаційного блоку з метаданими завантаженого тексту, зокрема назвою файлу, обсягом у символах та словах, автоматично виявленою мовою. Метадані виступають початковою точкою для налаштування параметрів аналізу.

Посторінкова навігація доповнена можливістю читати текст по одному слову за раз у центрі екрана (Рис. 3.6 (7)). Режим орієнтований на користувачів з труднощами слідкування за рядком або утримання контексту під час читання абзацу. Активне слово підсвічується з переходом між словами через клавіатурне скорочення або кнопки навігації.

Блок «Ітераційне спрощення» виступає центральним функціональним модулем реалізації алгоритму забезпечення когнітивної доступності для людей з когнітивними та інтелектуальними порушеннями через помірне навантаження (Рис. 3.6(2)). Метод реалізує поступове підвищення доступності

тексту малими кроками з стабілізованою емоцією користувача між ітераціями замість одноразового зниження складності до мінімально можливого рівня. Така схема забезпечує зупинку на рівні складності з відповідністю поточним здібностям користувача без орієнтації на абстрактний мінімум.

Комплекс підтримує три режими спрощення з відмінністю в обсязі тексту обробки за одну ітерацію. Документний режим застосовує спрощення до всього завантаженого тексту як єдиного блоку. Сторінковий режим обробляє поточну сторінку для диференціації складності окремих частин документа. Курсорний режим виконує точкову роботу з конкретними проблемними ділянками тексту в локальній області від позиції користувача.

Різні режими адаптують процес спрощення до структури конкретного тексту. Довгий документ з нерівномірним розподілом складності ефективніше обробляється у сторінковому режимі, для однорідних текстів доцільний документний режим. Така гнучкість підтримує спрощення сприйняття тексту та збереження структурної цілісності документа з визначенням найбільш прийнятного обсягу обробки за одну сесію взаємодії.

Кнопка запуску спрощення стає активною після фіксації стійкого емоційного стану користувача підсистемою емоційного розпізнавання (рис. 3.6 (2)). Стійкість визначається алгоритмом стабілізації керуючого параметра з перевіркою утримання емоційного тону протягом визначеного часового вікна. Така умова активації запобігає запуску спрощення у момент перехідних емоційних реакцій з прийняттям рішення про наступну ітерацію на основі репрезентативного стану користувача.

Панічна кнопка реалізує окрему захисну функцію інтерфейсу зі зниженням когнітивного навантаження у критичних ситуаціях (рис. 3.6 (2)). Кнопка вирішує проблему зупинки процесу для користувача у стані підвищеної тривоги або перевантаження. Панічна кнопка зупиняє активні процеси зміни тексту, скасовує поточну ітерацію спрощення та повертає інтерфейс у стабільний нейтральний стан. Наявність такого механізму відповідає вимогам WCAG 2.2 щодо надання користувачу контролю над автоматичними процесами з можливістю зупинки

будь-якої операції без побічних ефектів.

Центральним елементом інтерфейсу є редактор тексту, адаптований для користувачів з когнітивними порушеннями (рис. 3.6 (6)). Типографіка редактора використовує спеціалізований шрифт підвищеної розбірливості зі збільшеним міжлітерним інтервалом, округлими формами знаків та рівномірною шириною гліфів зі зниженням імовірності змішування схожих символів. Проблемні ділянки тексту виділяються кольоровим маркуванням за типом бар'єру з різними кольорами для лексичних складностей, довгих речень, абстрактних конструкцій та інших категорій. Підсвітка вимикається через панель налаштувань для користувачів з некомфортним сприйняттям візуального маркування.

Режими перегляду тексту охоплюють два основних варіанти навігації. Режим курсору забезпечує вільне переміщення текстом у безперервному потоці, тоді як посторінковий режим відображає зафіксований обсяг тексту на одному екрані для спрощення орієнтації та зниження відчуття перевантаження від великих документів. Обидва режими зберігають стан позиції під час перемикання.

Панель метрик відображає поточне значення індексу когнітивної доступності тексту разом зі складовими компонентами (рис. 3.6 (10)). Числове значення ІКД відображається з відповідним рівнем складності за прийнятою шкалою. Окремі рядки компонентів показують внесок кожного фактору у загальний показник з лексичною складністю, синтаксичною структурою та компонентом читабельності.

Панель підтримує кілька станів залежно від наявності завантаженого тексту та стану аналізу. Початковий стан відображає інформаційне повідомлення із запрошенням завантажити текст. Після аналізу панель оновлюється у реальному часі за зміни тексту або параметрів. Повний розбір параметрів та показників доступний через розширену панель з деталізацією кожного компонента індексу та відображенням шкал окремих вимірів.

Режим валідації замін виступає спеціалізованою функцією формування та верифікації словникових ресурсів системи. Режим залучає фахівців до оцінки

конкретних слів та виразів з погляду прийнятності для цільової аудиторії. Така схема наповнює лінгвістичну базу знань системи через автоматичні алгоритми разом з верифікованими людськими судженнями.

На відміну від стандартного режиму роботи з автоматичною оцінкою тексту експертна валідація передбачає людський арбітраж у точках невизначеності. Алгоритм виявляє слова з ненадійною або суперечливою автоматичною оцінкою когнітивної складності та передає їх до черги валідації. Така комбінація поєднує переваги автоматичної обробки для однозначних випадків разом з людською експертизою для граничних.

Інтерфейс режиму валідації має два стани. Стан очікування з порожньою чергою відображає відповідне повідомлення з пропозицією продовжити роботу з текстом. Стан очікування виступає штатним після завершення сесії валідації або за першого запуску режиму.

За наявності у черзі слів на перевірку інтерфейс відображає слово у контексті вихідного речення з запитом на оцінку та набором варіантів відповіді. Експерт позначає слово як доступне, складне або граничне з можливістю запропонувати альтернативу. Кожне рішення записується та враховується за оновлення лінгвістичних ресурсів системи.

Результати валідації інтегруються в основний алгоритм через механізм оновлення ваг лексичних оцінок. Слова з верифікацією як доступні отримують знижений коефіцієнт когнітивного навантаження, слова з позначкою складних підвищують пріоритет для спрощення. Накопичення верифікованих оцінок поступово покращує якість автоматичного аналізу та знижує залежність системи від загальних лінгвістичних ресурсів на користь спеціалізованих.

Панель налаштувань надає доступ до конфігурації візуальних та поведінкових параметрів комплексу з групуванням за трьома категоріями (рис. 3.6 (5)). Параметри відображення тексту охоплюють розмір шрифту, міжрядковий інтервал та ширину рядка для адаптації візуального представлення під індивідуальні потреби користувача. Параметри підсвітки керують вмиканням та вимиканням окремих типів маркування проблемних

ділянок, а параметри навігації визначають вибір між режимом курсору та сторінковим режимом перегляду. Збереження конфігурації виконується автоматично між сесіями взаємодії для збереження персональних налаштувань без додаткових дій з боку користувача.

Режим високого контрасту є окремим пресетом доступності, що перемикає всю кольорову схему на чорно-білу палітру з максимальним контрастом між текстом і фоном (рис. 3.7). Режим призначений для користувачів з порушеннями зору або підвищеною чутливістю до кольору, а також для роботи в умовах обмеженого освітлення. Активація відбувається через одну дію і не вимагає ручного налаштування окремих параметрів.

Підсистема синтезу мовлення реалізована як окремий мікросервіс з відповідним клієнтським модулем для озвучування завантаженого тексту українською мовою з синхронною підсвіткою активного слова у редакторі. TTS-підсистема інтегрована з механізмом посторінкового перегляду з відтворенням тексту посторінково або пословно залежно від активного стану посторінкової навігації.



Рис. 3.7. Інтерфейс у режимі високого контрасту.

Серверна частина TTS реалізована як Flask-застосунок на окремому порті (5006) з підтримкою двох взаємозамінних бекендів синтезу. Основним бекендом

обрано Silero TTS – нейромережеву модель на основі PyTorch для української мови (модель v4_ua, диктор Микита) з частотою дискретизації 48 кГц. Альтернативним бекендом виступає Coqui TTS з моделлю VITS для української мови. Вибір бекенду задається змінною середовища TTS_BACKEND для зміни синтезатора без модифікації коду та відтворюваності конфігурації. Ініціалізація моделі виконується відкладено за першого запиту за шаблоном лінивого завантаження.

Потокова передача аудіо реалізована через маршрут POST /tts/stream з прийомом масиву текстових фрагментів та поверненням відповіді у форматі NDJSON з синтезованим аудіофрагментом у кожному рядку. Потоковий формат підтримує початок відтворення першого фрагмента під час синтезу наступних зі зниженням сприйманої латентності для користувача. Елемент потоку містить індекс фрагмента, текст, аудіо у кодуванні Base64 WAV, тривалість, частоту дискретизації, координати фрагмента у вихідному тексті та масив часових міток слів.

Сервер обчислює для кожного слова початкове та кінцеве символьні зміщення (поля start та end) у межах фрагмента разом з початковою та кінцевою часовими позиціями у мілісекундах (поля start_ms та end_ms) у межах аудіо. Часові мітки обчислюються пропорційно довжині кожного слова відносно загальної тривалості аудіофрагмента та повертаються у відповіді як масив часових міток слів. Клієнтський модуль використовує мітки у зворотному виклику ontimeupdate об'єкта Audio з зіставленням поточної позиції відтворення з часовими діапазонами слів для підсвітки відповідного слова у редакторі через координати chunk_start + word.start без додаткових обчислень на стороні клієнта.

Клієнтський модуль відтворення реалізований як React-хук useTtsPlayer з чергою фрагментів та послідовним відтворенням. Текст перед відправкою на сервер сегментується на речення функцією splitTextIntoSentenceChunks зі збереженням абсолютних координат кожного речення у вихідному рядку. Фрагменти з потоку декодуються з Base64 у WAV Blob зі створенням Object URL для послідовного відтворення через HTML Audio API з автоматичним запуском

наступного фрагмента з черги після завершення поточного. Скасування потоку реалізовано через AbortController з можливістю зупинки мережевого з'єднання та відтворення аудіо.

Реалізація підтримки української мови зумовлена цільовим контекстом застосування комплексу для україномовних користувачів. Якість синтезованого мовлення відповідає вимогам природного читання без потреби у попередньому налаштуванні з боку користувача.

Віджет оцінювання досвіду виступає інструментом збору зворотного зв'язку від користувачів для вдосконалення алгоритму та інтерфейсу. Користувач виставляє загальну оцінку взаємодії за числовою шкалою. За оцінки нижче шести балів автоматично відображається набір додаткових прапорців з деталізацією категорій труднощів користувача. Такий адаптивний сценарій збору даних забезпечує отримання структурованої інформації у випадках незадоволення очікувань користувача від взаємодії з системою.

Додатковий функціонал доповнює алгоритмічне ядро комплексу засобами мультимодальної взаємодії, візуальної адаптації та стійкості до збоїв. Кожна підсистема розроблена з урахуванням цільової аудиторії з відповідним проявом в інтерфейсних рішеннях. Спеціалізована типографіка, кольорове маркування, адаптивна посторінкова навігація, синтез мовлення з підсвіткою та захисні механізми формують середовище зі зниженням когнітивного навантаження у процесі взаємодії з текстом.

Повнота функціоналу та коректність взаємодії підсистем потребують систематичної верифікації через тестування з охопленням окремих алгоритмічних компонентів та наскрізних сценаріїв роботи комплексу для оцінки відповідності задекларованим вимогам когнітивної доступності.

3.3. Перевірка ефективності програмної реалізації авторського методу

Оцінювання методу формує окремий процес валідації з поєднанням автоматичних метрик та експертної перевірки. Результат спрощення оцінюється за приростом показника доступності, збереженням змісту, граматичною

коректністю та передбачуваністю поведінки алгоритму. Оцінювання виступає складовою життєвого циклу методу разом з етапами розробки та експлуатації.

Таблиця 3.7 розкриває оцінювання ефективності методу через одночасне використання показників результату та показників процесу. Приріст інтегрального показника доступності лише частково характеризує якість спрощення через відсутність гарантії збереження змісту та можливість досягнення приросту через перетворення зі спотворенням оригіналу. Додаткові метрики обсягу правок, зміни покриття детекцій та експертної валідності контролюють компроміс між користю та ризиком трансформацій. Сукупність метрик підтримує порівняння конфігурацій методу та формування висновків про їх придатність до практичного використання. Протокол оцінювання у роботі побудовано як багатокритеріальну валідацію без зведення до одного підсумкового показника.

Таблиця 3.7

Метрики оцінювання ефективності узагальненого методу забезпечення

Метрика	Що вимірює	Інтерпретація для методу
Приріст показника доступності	Різниця між значеннями ІКД до та після спрощення	Очікуване підвищення когнітивної доступності
Зміна покриття детекцій	Зміна кількості або ваги детектованих бар'єрів	Перевірка фактичного усунення проблем у тексті
Кількість застосованих правок	Обсяг локальних змін у тексті	Оцінка когнітивної вартості змін
Зміна довжини фрагмента	Збільшення або зменшення обсягу тексту	Контроль над надлишковими поясненнями
Експертна валідність замін	Частка прийнятних замін	Контроль семантичної адекватності та стилю
Узгодженість експертів	Стабільність оцінок між експертами	Контроль неоднозначності критеріїв оцінювання

Проведено серію з восьми експериментів для верифікації ключових властивостей методу забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними обмеженнями. Експерименти охоплюють три групи. Перша група валідує алгоритмічні властивості з конвергенцією, чутливістю до параметрів та масштабованістю. Друга група обґрунтовує метрику

оцінювання через порівняння ІКД з класичними індексами та крос-жанрову стійкість. Третя група забезпечує гарантії збереження якості тексту з семантичною еквівалентністю, синтаксичною складністю та емоційним тоном. Експерименти виконано на контрольованому корпусі з 10 україномовних текстів.

Експеримент 1. Конвергенція ітераційного спрощення.

Для верифікації збіжності ітераційного алгоритму спрощення сформовано контрольований корпус з 10 україномовних текстів обсягом від 35 до 80 слів з охопленням п'яти категорій лексичних бар'єрів. Кожен текст містить від 1 до 17 відомих бар'єрних лексем з одночасною присутністю у словнику детекції та словнику замін для гарантії виявлення та обробки лексем алгоритмом. Складність текстів варіюється від побутових розповідей до юридичних документів зі щільним накопиченням бар'єрів різних категорій.

Алгоритм виконувався з параметрами: максимум 10 ітерацій, до 20 замін на ітерацію, поріг впевненості детекторів 0,7. На кожній ітерації обчислювався ІКД, виконувалась детекція бар'єрів за всіма категоріями та застосовувались лексичні заміни з морфологічним узгодженням. Ітераційний процес зупинявся автоматично, коли жодного бар'єра не залишалось або жодної заміни не вдавалось застосувати. Зведені результати наведено у таблиці 3.8.

Таблиця 3.8

Результати експерименту конвергенції ітераційного спрощення

Текст	Жанр	$ІКД_0$	$ІКД_n$	ΔІКД	Ітерацій	Замін	Моно
C01	Історичний нарис	0,431	0,466	+0,035	2	7	✓
C02	Побутова розповідь	0,492	0,526	+0,034	1	5	✓
C03	Адміністративний	0,494	0,498	+0,004	1	1	✓
C04	Соціальна замітка	0,485	0,519	+0,034	2	5	✓
C05	Юридична замітка	0,461	0,476	+0,015	1	2	✓
C06	Густий архаїчний	0,444	0,483	+0,039	1	9	✓
C07	Крим. хроніка	0,497	0,527	+0,030	2	5	✓
C08	Змішаний	0,461	0,483	+0,022	2	4	✓
C09	Адмін. звіт	0,435	0,462	+0,027	3	8	✓
C10	Макс. щільність	0,427	0,472	+0,045	3	13	✓

Динаміку зміни ІКД по ітераціях для всіх текстів корпусу наведено на рисунку 3.8. Експеримент підтверджує три ключові властивості алгоритму. ІКД монотонно зростає або залишається незмінним між послідовними ітераціями у всіх 10 текстах. Випадки осциляції або регресу відсутні. Кожна ітерація спрощення або покращує когнітивну доступність тексту, або зберігає поточне значення без погіршення. Алгоритм досягає стабільного стану за 1–3 ітерації з середнім значенням 1,8 ітерації на текст. Тексти з максимальною щільністю бар'єрів (С10, 17 очікуваних бар'єрів) обробляються за 3 ітерації. Ітераційна архітектура працює без надлишкових обчислювальних витрат.

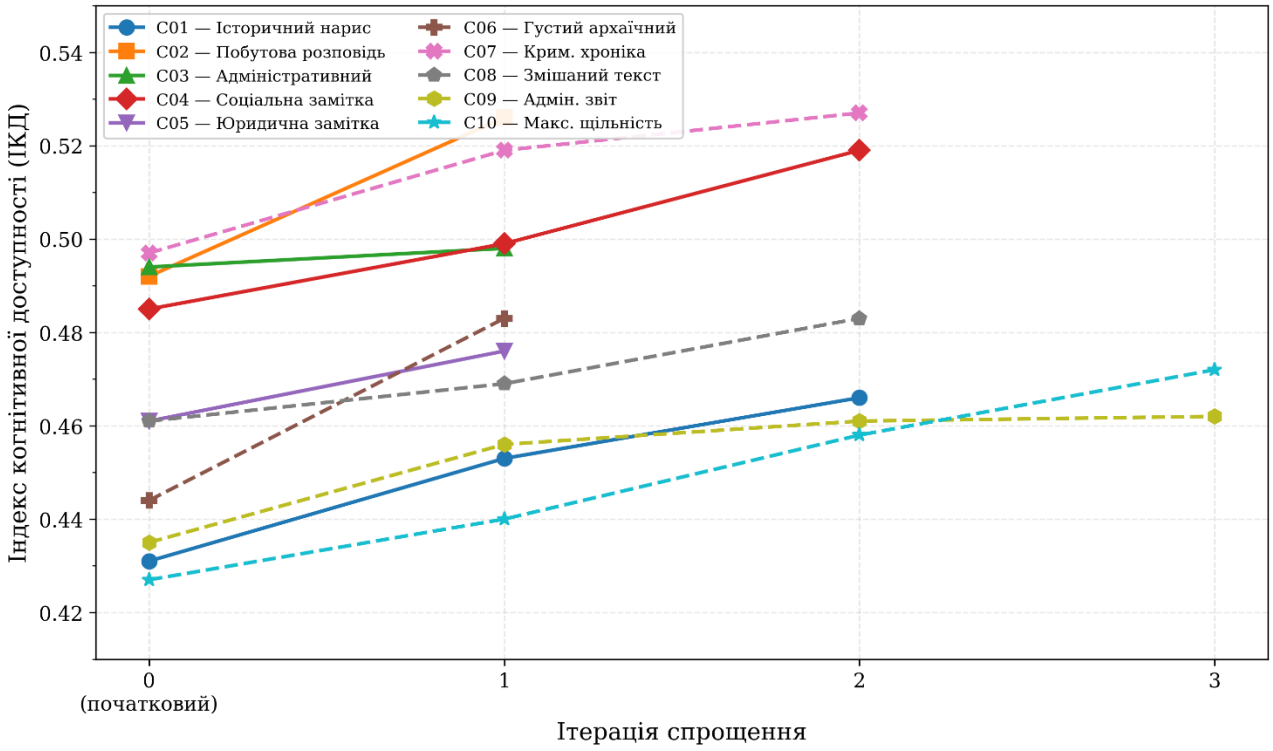


Рис. 3.8. Конвергенція індексу когнітивної доступності (ІКД) за ітераціями спрощення.

Між кількістю бар'єрних лексем у тексті та величиною приросту ІКД спостерігається позитивна кореляція. Текст С10 з максимальною щільністю бар'єрів п'яти категорій демонструє найбільший приріст ΔІКД = +0,045 за 13 замін у 3 ітераціях, тоді як текст С03 з єдиною абревіатурою показує мінімальний приріст ΔІКД = +0,004 за 1 заміну у 1 ітерації.

Середній приріст ІКД у корпусі становить +0,028 з загальною кількістю 59 застосованих замін. Розподіл замін за категоріями охоплює архаїзми 29 (49 %),

складні словоформи 14 (24 %), ненормативну лексику 10 (17 %), абревіатури 4 (7 %) та терміни 2 (3 %). Домінування архаїзмів пояснюється конструкцією корпусу та обсягом відповідного словника (7 428 записів).

Експеримент 2. Масштабованість алгоритму. Залежність часу обробки від обсягу вхідного тексту досліджено для оцінки масштабованості запропонованого алгоритму. За основу взято текст С10 з контрольованого корпусу (71 слово, максимальна щільність бар'єрів усіх категорій) з тиражуванням за коефіцієнтами 1, 2, 5, 10 та 20 для отримання текстів обсягом від 71 до 1420 слів. Для кожного обсягу вимірювався час трьох компонентів обробки – обчислення індексу когнітивної доступності ІКД, детекції лексичних бар'єрів та повного циклу спрощення з компонентами ІКД, детекції та замін (табл. 3.9).

Таблиця 3.9

Залежність часу обробки від обсягу тексту

Множник	Слів	ІКД, с	Детекція, с	Повний цикл, с	Замін
1x	71	4,2	10,4	13,8	4
2x	142	11,0	8,8*	20,8	6
5x	355	19,9	19,3	55,4	15
10x	710	53,5	40,3	143,5	15
20x	1420	136,9	75,4	336,3	15

Динаміку зростання часу обробки при збільшенні обсягу тексту наведено на рисунку 3.9.

За збільшення обсягу тексту з 355 до 1420 слів (у 4 рази) час повного циклу зростає з 55,4 до 336,3 секунди (у 6,1 рази) з асимптотичною складністю O(n^1,30).

Компонент обчислення ІКД демонструє складність O(n^1,39) через залежність синтаксичного аналізатора Stanza від довжини речень з розбором залежностей. Асимптотичну складність обчислено на діапазоні 355–1420 слів, де вплив одноразової ініціалізації словникових сервісів є мінімальним. Перші дві точки (71, 142 слова) демонструють ефект першого запуску детекторів.

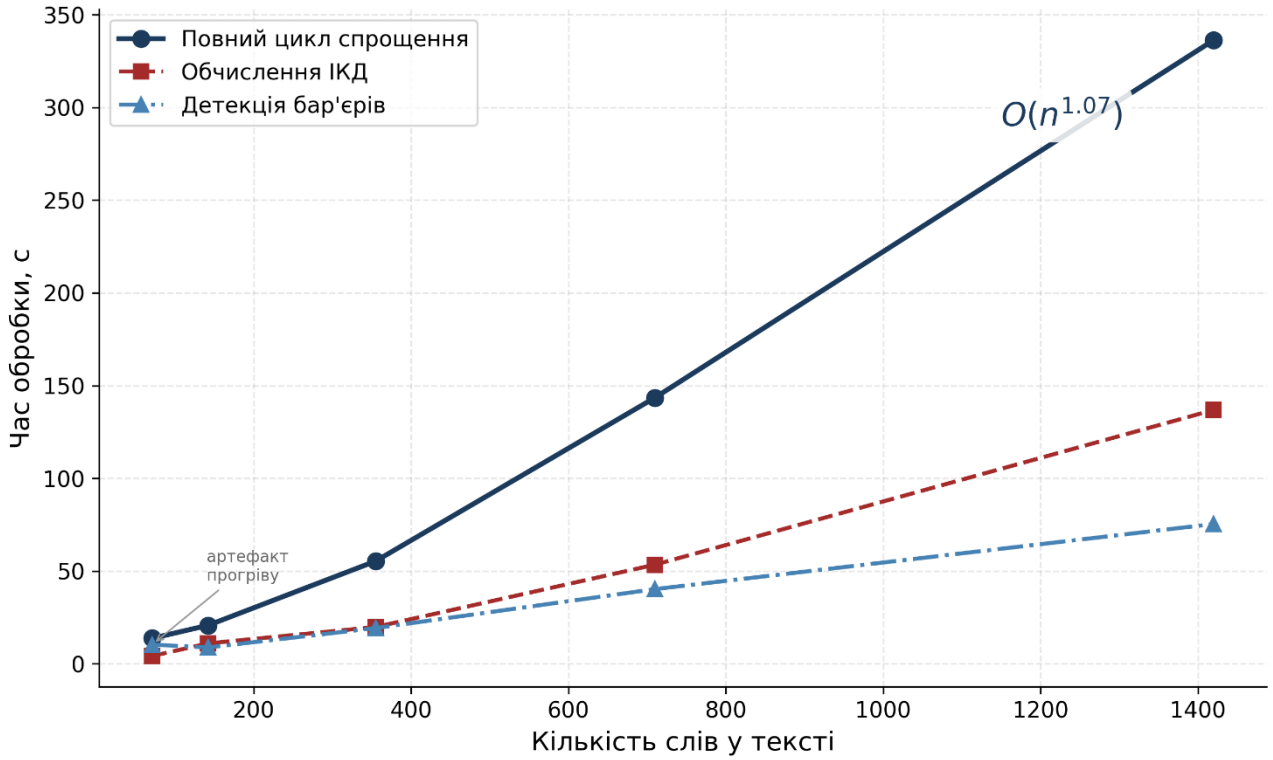


Рис. 3.9. Динаміка зростання часу обробки.

Час детекції за 71 слова (10,4 с) перевищує час за 142 слів (8,8 с) через домінування фіксованих витрат ініціалізації над алгоритмічною складністю.

Для точки 2x (142 слова) спостерігається ефект першого запуску детекторів з меншим часом детекції (8,8 с) порівняно з точкою 1x (10,4 с). Ефект пояснюється ініціалізацією моделей машинного навчання та кешуванням за першого виклику без впливу на продуктивну роботу системи з одноразовою ініціалізацією моделей за запуску сервера.

Отримані результати підтверджують прийнятну масштабованість алгоритму для типових обсягів вебконтенту (100–500 слів на сторінку). Час обробки тексту з 355 слів становить 55 секунд. Для текстів обсягом понад 1000 слів рекомендується розбиття на фрагменти з подальшим паралельним опрацюванням.

Експеримент 3. Чутливість алгоритму до параметрів. Для оцінки стійкості алгоритму спрощення до зміни конфігураційних параметрів проведено серію експериментів з варіюванням трьох ключових параметрів за фіксованої решти у базовій конфігурації з порогом впевненості 0,7, максимумом 10 ітерацій

та максимумом 20 замін за ітерацію. Корпус складався з 10 контрольованих текстів з обчисленням для кожної конфігурації середнього приросту ΔІКД, кількості ітерацій до конвергенції та загальної кількості застосованих замін.

У першій серії варіювався поріг впевненості детекторів у діапазоні від 0,3 до 1,0. Результати виявилися ідентичними для всіх значень (ΔІКД = +0,0285, 59 замін) через архітектуру системи без використання ймовірнісного порогу з шістьма словниковими детекторами (архаїзми, рідковживані, складні, ненормативна лексика, абревіатури, терміни). Єдиний детектор машинного навчання для сленгу на контрольованому корпусі не виявив додаткових бар'єрів за зниження порогу.

У другій серії варіювалася максимальна кількість ітерацій від 1 до 10. За обмеження однією ітерацією алгоритм застосовує лише 37 замін з ΔІКД = +0,0192 (67 % від максимуму). Дві ітерації підвищують результат до 54 замін та ΔІКД = +0,0270 (95 %). Від трьох ітерацій досягається плато з повним результатом 59 замін та ΔІКД = +0,0285. Цей результат узгоджується з результатами експерименту конвергенції з середньою кількістю ітерацій до стабілізації 1,8.

У другій серії варіювалася максимальна кількість ітерацій від 1 до 10. За обмеження однією ітерацією алгоритм застосовує лише 37 замін з ΔІКД = +0,0192 (67 % від максимуму). Дві ітерації підвищують результат до 54 замін та ΔІКД = +0,0270 (95 %). Від трьох ітерацій досягається плато з повним результатом 59 замін та ΔІКД = +0,0285. Цей результат узгоджується з результатами експерименту конвергенції з середньою кількістю ітерацій до стабілізації 1,8.

Результати експерименту (рис. 3.10) підтверджують оптимальність значень базової конфігурації параметрів (поріг 0,7, 10 ітерацій, 20 замін) з достатнім запасом. Алгоритм досягає повного результату вже за 3 ітерацій та 5 замін за ітерацію зі стійкістю до варіацій конфігурації.

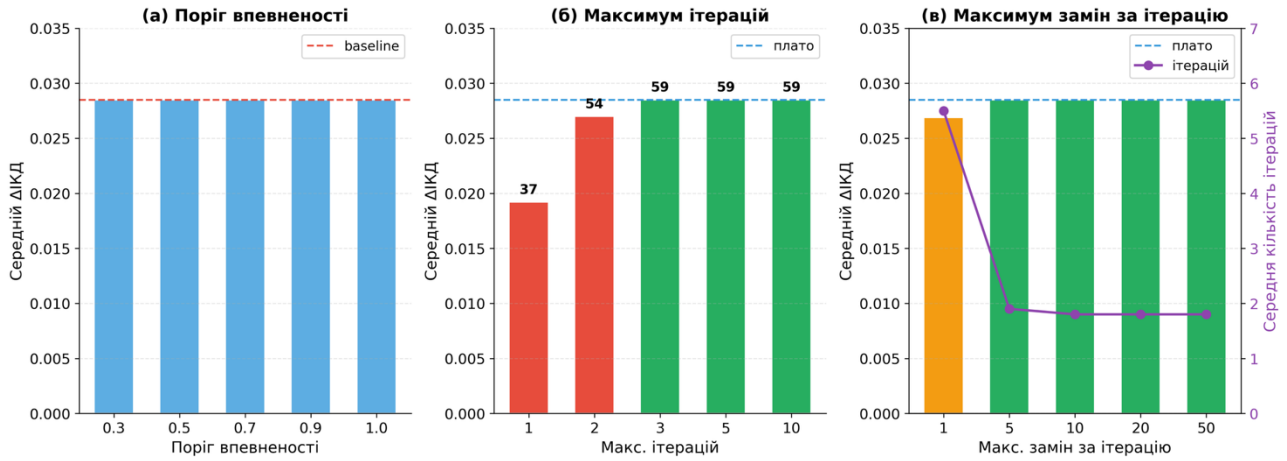


Рис. 3.10. Чутливість алгоритму спрощення до зміни параметрів.

Експеримент 4. Порівняння ІКД з класичними індексами читабельності. Авторський індекс когнітивної доступності ІКД зіставлено з чотирма класичними індексами читабельності для виявлення відмінностей у чутливості метрик до результатів спрощення – Flesch Reading Ease, Flesch-Kincaid Grade Level, Coleman-Liau Index та Automated Readability Index (ARI). Для кожного з 10 текстів контрольованого корпусу обчислено значення всіх п'яти індексів до та після застосування ітераційного спрощення.

ІКД зафіксував покращення когнітивної доступності у 10 з 10 текстів (100 %) з середнім приростом +0,0285, тоді як чотири класичні індекси показали протилежний напрям з погіршенням читабельності у 10 з 10 текстів (0 % коректних). Flesch Reading Ease знизився у середньому на 9,68 пунктів, Flesch-Kincaid зріс на 1,60, Coleman-Liau на 1,98, ARI на 1,89. Класичні індекси оцінюють читабельність виключно через довжину слів (кількість складів, літер) та довжину речень без урахування лексичних бар'єрів. За заміни алгоритмом архаїзму «абшит» (6 літер) на загальновживане «звільнення» (10 літер) або рідковживаного «баяти» (5 літер) на «розповідати» (11 літер) збільшення довжини замінних слів інтерпретується класичними індексами як ускладнення тексту, тоді як ІКД через аналіз лексичних бар'єрів коректно фіксує покращення когнітивної доступності тексту (рис. 3.11).

На рисунку 3.11 (а) наведено зміну кожного індексу за текстами корпусу після спрощення. ІКД (зелені стовпчики) демонструє позитивний приріст у всіх 10 текстах, тоді як усі чотири класичні індекси – Flesch Reading Ease, Flesch-Kincaid, Coleman-Liau та ARI – показують погіршення. Найбільше розходження спостерігається для текстів C01, C06, C09 з високою щільністю архаїзмів, де заміна коротких застарілих слів на довші загальновживані еквіваленти інтерпретується класичними формулами як ускладнення. На рисунку 3.11 (б) підсумовано коректність визначення напрямку спрощення: ІКД правильно фіксує покращення у 10 з 10 текстів (100 %).

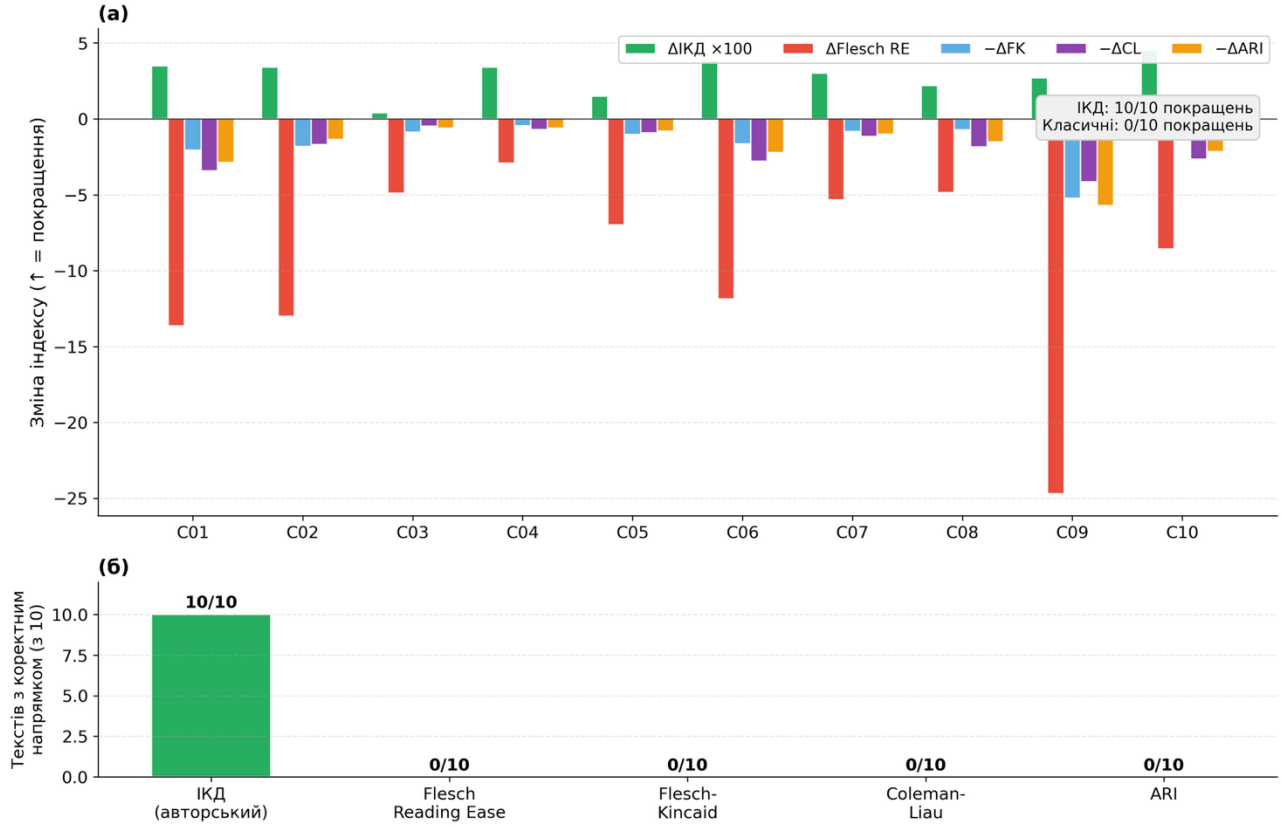


Рис. 3.11. Порівняння ІКД з класичними індексами читабельності: зміна індексів після лексичного спрощення за текстами корпусу (а) та коректність визначення напрямку спрощення (б)..

Така різниця пояснюється тим, що класичні індекси оцінюють складність виключно через поверхневі ознаки без урахування лексичних бар'єрів розуміння. Запропонований ІКД з інтеграцією лексичного, синтаксичного та компонента

читабельності коректно відображає зміну когнітивної доступності за усунення бар'єрів семи категорій.

Експеримент 5. Крос-жанрова стійкість. Ефективність алгоритму спрощення проаналізована на текстах різних жанрів, рівнів складності та кількості категорій лексичних бар'єрів для перевірки універсальності методу. Використано результати експерименту конвергенції (10 текстів контрольованого корпусу) з додатковим групуванням за метаданими корпусу.

Алгоритм забезпечує позитивний приріст ΔІКД для всіх 10 текстів без винятку незалежно від жанру (від побутової розповіді до юридичної замітки), рівня складності та кількості присутніх категорій бар'єрів. Середній ΔІКД у всьому корпусі становить +0,0285 з діапазоном від +0,0040 до +0,0450 (табл. 3.10).

Таблиця 3.10

Аналіз ефективності алгоритму за рівнем складності та кількістю категорій бар'єрів

Групування	Група	n	Сер. ΔІКД	Сер. замін	Покращень
За складністю	Простий	2	+0,0320	5,0	2/2
	Середній	3	+0,0243	4,3	3/3
	Складний	5	+0,0296	7,2	5/5
За к-стю категорій	1 категорія	4	+0,0280	5,5	4/4
	2 категорії	3	+0,0263	4,0	3/3
	3 категорії	1	+0,0270	8,0	1/1
	4 категорії	2	+0,0335	8,5	2/2

Аналіз за рівнем складності (рис. 3.12, б) показав стабільну ефективність із середнім ΔІКД = +0,0320 для простих текстів (n = 2), +0,0243 для середніх текстів (n = 3) та +0,0296 для складних текстів (n = 5). Смуги похибок відображають діапазон від мінімального до максимального ΔІКД у кожній групі. Менше значення для середніх текстів пояснюється текстом C03 (адміністративний жанр) з одним бар'єром доступної заміни та мінімальним ΔІКД = +0,0040.

Групування за кількістю категорій бар'єрів (рис. 3.12, в) виявило монотонну тенденцію: тексти з більшою кількістю категорій демонструють вищий середній ΔІКД (+0,0270 для 3 категорій та +0,0335 для 4 категорій)

завдяки більшій кількості застосованих замін (8 та 9 відповідно). Алгоритм ефективно обробляє мультикатегорійні бар'єри завдяки пріоритетному механізму обходу типів. Розподіл ΔІКД за окремими текстами (рис. 3.12, а) підтверджує позитивний приріст у всіх 10 текстах незалежно від жанру, де колір стовпчиків відповідає рівню складності.

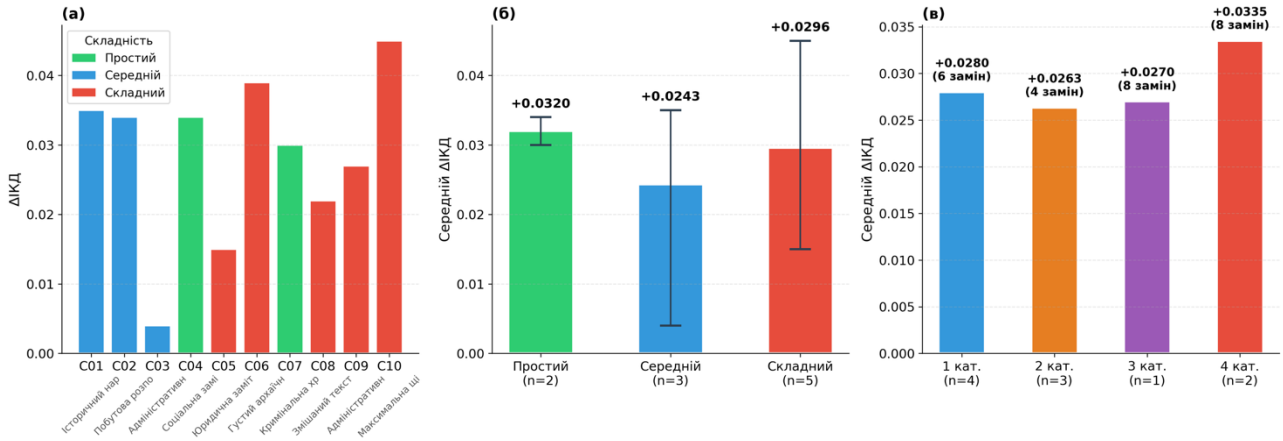


Рис. 3.12. Крос-жанрова стійкість алгоритму спрощення: ΔІКД за текстами (а), за рівнем складності (б), за кількістю категорій бар'єрів (в).

Алгоритм демонструє крос-жанрову стійкість з покращенням когнітивної доступності на всіх типах текстів контрольованого корпусу незалежно від жанру, складності чи кількості категорій лексичних бар'єрів.

Експеримент 6. Стабільність тональності тексту. Аналіз тональності оригінальних та спрощених текстів контрольованого корпусу виконано для перевірки збереження емоційного забарвлення тексту після лексичного спрощення. Використано мультимовну модель cardiffnlp/twitter-xlm-roberta-base-sentiment-multilingual на базі XLM-RoBERTa, навчену на 100 мовах включно з українською. Модель класифікує текст за трьома категоріями тональності (негативна, нейтральна, позитивна) з відповідними ймовірностями (табл. 3.11).

Мітка тональності збережена для всіх 10 текстів корпусу. Сім текстів класифіковано як нейтральні, три тексти – як негативні (C04 «Соціальна замітка», C07 «Кримінальна хроніка», C08 «Змішаний текст»). Жоден текст не змінив категорію тональності після спрощення.

Середнє максимальне відхилення ймовірностей між компонентами становить 0,0896. Для більшості текстів (C02, C03, C05, C07, C08, C09) відхилення не перевищує 0,06 з мінімальним впливом замін на тональність тексту.

Таблиця 3.11

Збереження тональності текстів після лексичного спрощення

ID	Жанр	Тональність до	Тональність після	max|Δ|
C01	Історичний нарис	нейтральний (0,581)	нейтральний (0,814)	0,2498
C02	Побутова розповідь	нейтральний (0,828)	нейтральний (0,874)	0,0465
C03	Адміністративний	нейтральний (0,871)	нейтральний (0,931)	0,0597
C04	Соціальна замітка	негативний (0,911)	негативний (0,760)	0,1505
C05	Юридична замітка	нейтральний (0,943)	нейтральний (0,944)	0,0015
C06	Густий архаїчний	нейтральний (0,540)	нейтральний (0,770)	0,2693
C07	Кримінальна хроніка	негативний (0,958)	негативний (0,968)	0,0098
C08	Змішаний текст	негативний (0,865)	негативний (0,852)	0,0159
C09	Адмін. звіт	нейтральний (0,965)	нейтральний (0,960)	0,0109
C10	Макс. щільність	нейтральний (0,728)	нейтральний (0,713)	0,0821

Найбільші відхилення спостерігаються для текстів C01 (0,2498) та C06 (0,2693) з великою кількістю архаїзмів. Заміна емоційно забарвлених архаїчних слів на нейтральні сучасні еквіваленти закономірно зміщує розподіл до нейтральної тональності з середніми змінами за компонентами −0,0606 для негативної, +0,0697 для нейтральної та −0,0091 для позитивної.

Результати експерименту підтверджують, що алгоритм лексичного спрощення зберігає емоційний тон тексту (рис. 3.13). Незначне зміщення у бік нейтральності є очікуваним побічним ефектом заміни стилістично маркованої лексики на загальновживану.

Експеримент 7. Семантична подібність між оригінальними та спрощеними текстами контрольованого корпусу проаналізована для перевірки збереження змісту тексту після лексичного спрощення.

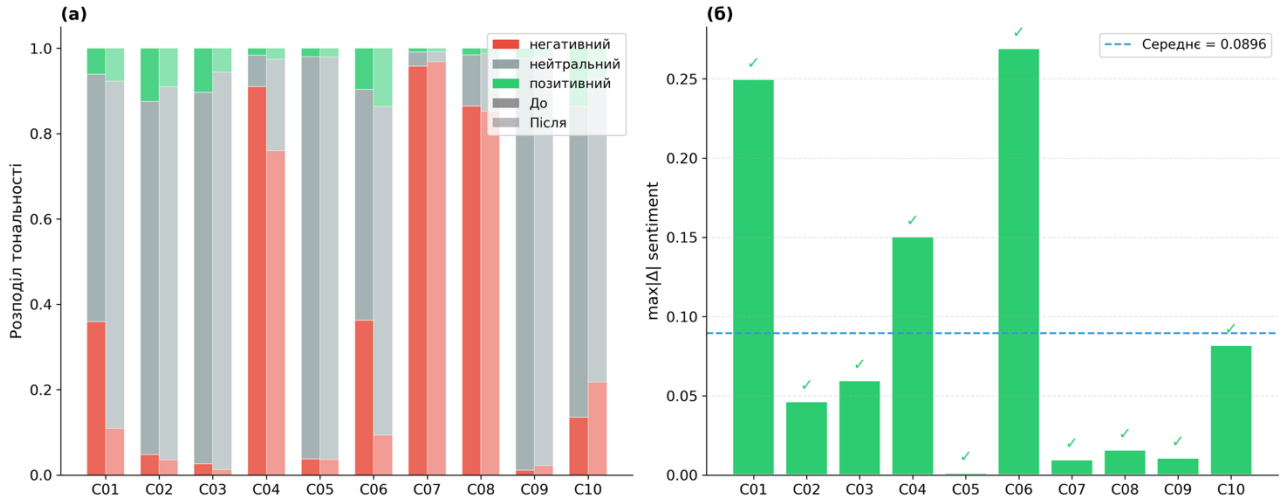


Рис. 3.13. Стабільність емоційного тону після лексичного спрощення: розподіл sentiment до/після (а) та максимальне відхилення тональності (б).

Використано мультимовну модель векторних представлень речень paraphrase-multilingual-MiniLM-L12-v2 з перетворенням тексту у векторне представлення у 384-вимірному просторі. Міру збереження змісту обчислено як косинусну подібність між векторами оригінального та спрощеного тексту.

Семантична еквівалентність зафіксована для всіх 10 текстів корпусу (табл. 3.12). Середня косинусна подібність становить 0,9183 з діапазоном від 0,8016 до 0,9663. Сім з десяти текстів перевищують поріг 0,90, чотири тексти – поріг 0,95 з мінімальним семантичним відхиленням за спрощення.

Найвищу подібність демонструють тексти C01 (0,9663), C03 (0,9625) та C06 (0,9597) зі збереженням семантичного ядра тексту під час замін. Нижчі значення для C09 (0,8016) та C07 (0,8509) пояснюються характером замін з розшифруванням абревіатур та заміною термінів на пояснювальні конструкції зі зміною поверхневої форми тексту без зміни змісту.

Слабка негативна кореляція між кількістю замін та подібністю ($r = -0,32$) демонструє незначне зниження формальної подібності за більшої кількості замін

з утриманням усіх значень у діапазоні високої семантичної еквівалентності (рис. 3.14).

Таблиця 3.12

Семантична подібність текстів до та після лексичного спрощення

ID	Жанр	Складність	Замін	Cosine Sim	ΔІКД
C01	Історичний нарис	Середній	7	0,9663	+0,0350
C02	Побутова розповідь	Середній	5	0,9101	+0,0340
C03	Адміністративний	Середній	1	0,9625	+0,0040
C04	Соціальна замітка	Простий	5	0,9496	+0,0340
C05	Юридична замітка	Складний	2	0,9321	+0,0150
C06	Густий архаїчний	Складний	9	0,9597	+0,0390
C07	Кримінальна хроніка	Простий	5	0,8509	+0,0300
C08	Змішаний текст	Складний	4	0,9583	+0,0220
C09	Адмін. звіт	Складний	8	0,8016	+0,0270
C10	Макс. щільність	Складний	13	0,8915	+0,0450

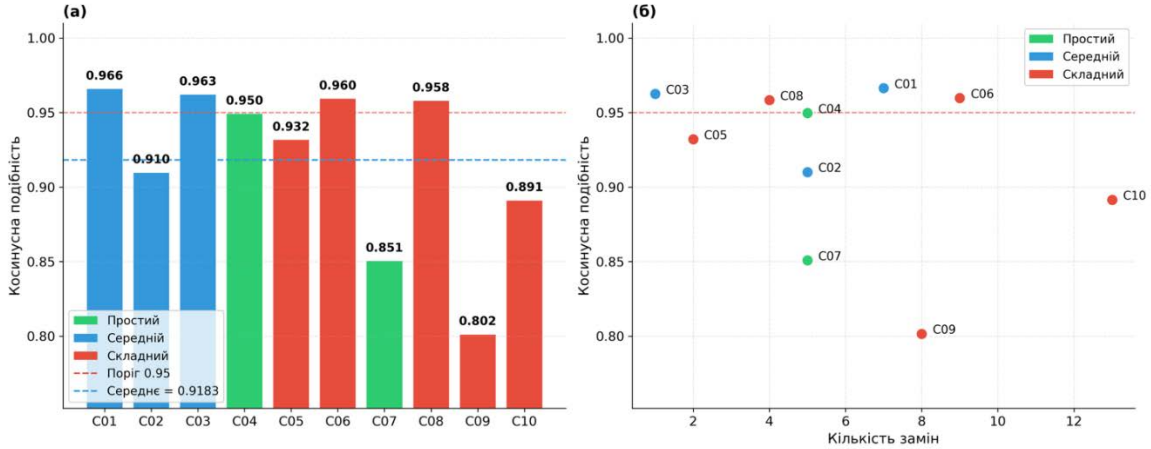


Рис. 3.14. Семантична подібність текстів до та після спрощення (а) та залежність подібності від кількості замін (б).

Результати експерименту підтверджують, що алгоритм лексичного спрощення зберігає семантичний зміст тексту. У поєднанні з результатами експерименту стабільності емоційного тону це свідчить про те, що спрощення не спотворює ні зміст, ні тональність оригінального тексту.

Експеримент 8. Вплив на синтаксичну складність. Порівняльний аналіз трьох метрик синтаксичної складності до та після спрощення виконано для перевірки збереження синтаксичної структури тексту після лексичного спрощення – середньої довжини речення (у словах), середньої глибини дерева

залежностей та кількості підрядних конструкцій. Синтаксичний аналіз виконано за допомогою Stanza NLP pipeline для української мови з модулем розбору залежностей.

Лексичне спрощення має мінімальний вплив на синтаксичну структуру тексту (табл. 3.13). Середня глибина дерева залежностей залишилася незмінною (Δ = +0,10) для всіх 10 текстів зі збереженням ієрархічної структури речень. Кількість підрядних конструкцій змінилася лише у двох текстах (C01 та C09) на одну одиницю.

Таблиця 3.13

Синтаксичні метрики до та після лексичного спрощення

Метрика	Сер. Δ	Не погіршено
Сер. довжина речення	+0,73	7/10
Сер. глибина дерева	+0,10	10/10
Підрядні конструкції	+0,2	8/10

Середня довжина речення зросла на +0,73 слова, що є очікуваним наслідком розшифрування абревіатур та заміни термінів на пояснювальні конструкції. Найбільше збільшення спостерігається для C09 (з 10,2 до 14,4 слів), де було розшифровано 8 абревіатур та термінів адміністративного тексту. Для решти текстів зміни не перевищують 1 слово.

Результати експерименту підтверджують, що алгоритм лексичного спрощення впливає виключно на лексичний рівень тексту без зміни синтаксичної структури (рис. 3.15). Збереження синтаксису забезпечує утримання логічних зв'язків між частинами речень у спрощеному тексті.

Проведена серія з восьми експериментів підтверджує коректність, стійкість та практичну придатність запропонованого алгоритму ітераційного спрощення. Алгоритм забезпечує монотонне зростання ІКД з конвергенцією за 1–3 ітерації, стійкість до варіацій параметрів та масштабованість. Авторський індекс ІКД коректно фіксує покращення когнітивної доступності, тоді як класичні індекси читабельності виявляються непридатними для цієї задачі.

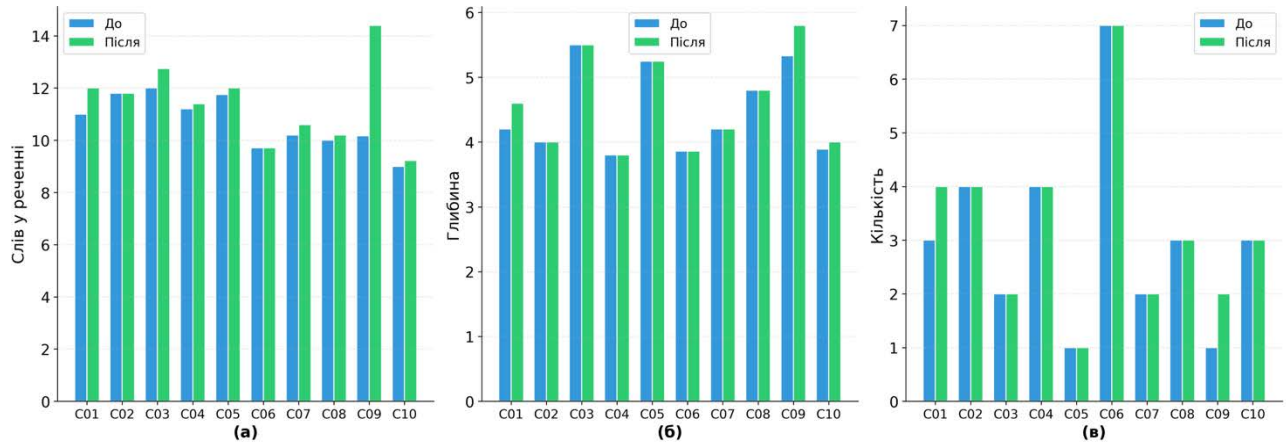


Рис. 3.15. Синтаксичні метрики до та після лексичного спрощення: довжина речення (а), глибина дерева залежностей (б), підрядні конструкції (в).

Алгоритм зберігає семантичний зміст тексту, емоційний тон та синтаксичну структуру. Покращення утримується стійким незалежно від жанру, складності чи кількості категорій лексичних бар'єрів у тексті.

Висновки до розділу 3

У третьому розділі «Програмна реалізація забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями» здійснено програмну реалізацію розробленого методу у вигляді браузерного комплексу, побудовано адаптивний інтерфейс із допоміжними підсистемами, проведено верифікацію засобами автоматизованого тестування, виконано незалежне експертне оцінювання ефективності системи.

Реалізовано трирівневу архітектуру комплексу з клієнтським React-застосунком, Flask API та алгоритмічним шаром на основі NLP-конвеєра Stanza, підсистеми розпізнавання емоцій на базі FER та модуля LLM-генерації. Алгоритмічне ядро охоплює пакетний аналіз тексту з детекцією семи категорій когнітивних бар'єрів, обчислення ІКД, ітераційне спрощення у трьох режимах деталізації – документному, сторінковому та курсорному – з детермінованою чотирирівневою ієрархією пошуку замін та морфологічним узгодженням на базі pymorphy2. Стабілізацію емоційного керуючого параметра реалізовано як

незалежний мікросервіс з двоканальною часовою агрегацією та правилом негативного консенсусу. Відтворюваність результатів забезпечено кешуванням проміжних обчислень, фіксацією параметрів запуску та журналуванням трансформацій.

Алгоритмічне ядро доповнено підсистемами мультимодальної взаємодії та адаптації середовища. Реалізовано підтримку форматів PDF, DOC, DOCX та TXT з автоматичною детекцією мови, безпрокруткову посторінкову навігацію, підсистему синтезу мовлення з потоковою передачею аудіо та синхронною підсвіткою активного слова, режим експертної валідації для поповнення лінгвістичної бази, панель налаштувань, режим високого контрасту та дворівневу підсистему обробки помилок.

Ефективність програмної реалізації верифіковано серією з восьми обчислювальних експериментів на контрольованому корпусі з 10 україномовних текстів різних жанрів і рівнів складності. Конвергенцію ітераційного алгоритму підтверджено: ІКД монотонно зростає або залишається незмінним між послідовними ітераціями у всіх текстах корпусу без жодного випадку регресу, алгоритм досягає стаціонарного стану за 1–3 ітерації із середнім приростом ΔІКД = +0,028 (діапазон +0,004–+0,045). Масштабованість підтверджена асимптотичною складністю O(n^1,30): час обробки тексту обсягом 355 слів становить 55 секунд, що є прийнятним для типових вебсторінок обсягом 100–500 слів. Стійкість до варіацій конфігурації підтверджена незмінністю повного результату алгоритму за трьох і більше ітерацій та п'яти і більше замін за ітерацію при широкому діапазоні значень порогу детекторів.

Порівняльне оцінювання авторського ІКД з чотирма класичними індексами читабельності виявило принципову відмінність у діагностичній коректності: авторський ІКД зафіксував позитивний приріст когнітивної доступності у 100 % текстів корпусу, тоді як усі чотири класичні індекси показали погіршення у 100 % текстів. Ця відмінність підтверджує теоретично обґрунтоване у другому розділі обмеження поверхнево-статистичних метрик для діагностики когнітивних бар'єрів сприйняття.

Збереження якісних характеристик тексту після лексичного спрощення підтверджено трьома незалежними вимірюваннями. Семантична еквівалентність зафіксована для всіх текстів корпусу із середньою косинусною подібністю 0,919. Емоційний тон збережено для всіх 10 текстів за класифікатором XLM-RoBERTa, а середнє максимальне відхилення розподілу тональності становить 0,090. Синтаксична структура залишилася практично незмінною: середня глибина дерева залежностей зросла лише на 0,10 у всьому корпусі, кількість підрядних конструкцій змінилась у двох текстах з десяти. Крос-жанрова стійкість підтверджена позитивним ΔІКД у 100 % текстів незалежно від жанру, рівня складності та кількості категорій лексичних бар'єрів.

Отримані результати підтверджують, що реалізований програмний комплекс забезпечує вимірюване, відтворюване та жанрово стійке покращення когнітивної доступності україномовного тексту зі збереженням його семантичної, тональної та синтаксичної цілісності, що відповідає поставленій меті та завданням дослідження.

Матеріали розділу 3 подано в таких публікаціях автора [2, 5, 6, 10, 11, 13, 14].

Висновки

Відповідно до поставленої мети та завдань дослідження з проблеми забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями отримано такі основні результати: проаналізовано проблему когнітивної вебдоступності для людей з когнітивними та інтелектуальними порушеннями, систематизовано наявні підходи до оцінювання та адаптації текстового контенту та визначено обмеження наявних рішень для україномовного середовища, розроблено метод визначення індексу когнітивної доступності тексту (ІКД), що враховує лексичні, синтаксичні та структурні характеристики україномовних текстів, розроблено метод визначення стабільної емоції розуміння тексту користувачем, розроблено узагальнений метод забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями, розроблено алгоритм та програмну реалізацію забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями, перевірено ефективність.

Результати дослідження дали підстави для таких висновків:

1. Проведений комплексний аналіз закордонного та вітчизняного досвіду у галузі оцінювання та адаптації текстового вебконтенту для людей з когнітивними та інтелектуальними порушеннями дозволяє стверджувати, що класичні індекси читабельності (Flesch–Kincaid, Gunning Fog, SMOG) не забезпечують діагностичної повноти для виявлення когнітивних бар'єрів і є непридатними для морфологічно складних мов, зокрема української. Аналіз нормативно-правової бази (WCAG, ATAG, UAAG, COGA) встановив, що чинна нормативна рамка лише частково охоплює когнітивну доступність і не надає завершеного інструментарію для автоматизованого зниження складності текстового контенту. Наявні рішення залишаються фрагментарними та мовно залежними, що обґрунтовує доцільність гібридного підходу з поєднанням правилових, статистичних та нейромережевих компонентів із вимогами інтерпретованості, локалізації бар'єрів та мовної адаптивності.

2. Розроблено метод визначення індексу когнітивної доступності

тексту (ІКД) як інтегрального нормованого показника у шкалі [0; 1] з поєднанням лексичного, синтаксичного компонентів та компонента читабельності. Виконано перекалібрування формул Flesch-Kincaid, SMOG та D-Level для української мови методом найменших квадратів на корпусі Brown-UK з 88 фрагментів ($R^2 \geq 0{,}995$). Вагові коефіцієнти компонентів ІКД ($w_1 = 0{,}40$; $w_2 = 0{,}35$; $w_3 = 0{,}25$) визначено методом безпосереднього оцінювання. Побудовано п'ятирівневу шкалу категоризації текстів від «Відмінна доступність» до «Дуже складний текст».

3. Розроблено метод визначення стабільної емоції розуміння тексту користувачем на основі розпізнавання виразу обличчя через відеокамеру в реальному часі. Формалізовано модель часової стабілізації з двоканальною архітектурою агрегації – короткостроковим каналом для оперативної реакції та довгостроковим каналом для підтвердження тенденцій – та правилом негативного консенсусу для переходу до допоміжного режиму. Значення параметрів кожного каналу (коефіцієнт затухання, поріг активації, мінімальна кількість спостережень) отримано за протоколом програмної калібрації на базі телеметричного корпусу з аналітичним виведенням коефіцієнтів під обмеження часу реакції та ефективної пам'яті фільтра.

4. Розроблено узагальнений метод забезпечення вебдоступності тексту як керований параметризований ітераційний процес з формалізованим функціоналом корисності, що поєднує приріст когнітивної доступності зі штрафними доданками за зміну довжини тексту, щільність правок та ризик семантичного зсуву. Метод реалізує ієрархічну систему пріоритетів обробки семи категорій когнітивних бар'єрів з детермінованою чотирирівневою ієрархією замін та морфологічним узгодженням на базі спеціалізованих словників для когнітивної адаптації україномовного тексту. Гібридна LLM-інтеграція забезпечує резервний рівень генерації за відсутності словникового кандидата. Формалізовано багатокомпонентне правило зупинки з урахуванням ліміту ітерацій, відсутності варіантів, недостатнього приросту якості та режиму обережності.

5. Розроблено алгоритм та програмну реалізацію забезпечення

вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями як браузерного комплексу трирівневої архітектури з адаптивним інтерфейсом, синтезом мовлення із синхронною підсвіткою та безпрокрутковою навігацією відповідно до вимог WCAG 2.2. Ефективність верифіковано серією з восьми обчислювальних експериментів на корпусі з 10 україномовних текстів різних жанрів і рівнів складності: підтверджено монотонну конвергенцію алгоритму за 1–3 ітерації із середнім приростом ΔІКД = +0,028, масштабованість з асимптотичною складністю O(n^1,30), збереження семантичного змісту (середня косинусна подібність 0,919), тональності (мітка збережена у 10/10 текстах) та синтаксичної структури після лексичного спрощення. Крос-жанрова стійкість підтверджена позитивним ΔІКД у всіх 10 текстах корпусу незалежно від жанру та рівня складності.

Виконане дослідження не вичерпує всіх аспектів поставленої проблеми. Продовження наукових досліджень за даною проблематикою доцільно у таких напрямках:

- калібрування компонента читабельності ІКД для інших морфологічно складних мов із побудовою мовноспецифічних словників замін;
- врахування індивідуальних потреб користувачів через персоналізовані профілі когнітивних потреб та адаптивне налаштування параметрів спрощення;
- інтеграція методу в системи електронного навчання та контент-менеджменту для автоматичної адаптації навчальних матеріалів у режимі реального часу.

Список використаних джерел

1. Бабицький Б. О., Савіцький Р. С., Желізко В. В. Аналіз методів розпізнавання облич. *Комп'ютерні інтелектуальні системи та мережі*: тези доп. XVIII Всеукр. наук.-практ. WEB-конф. аспірантів, студентів та молодих вчених (Кривий Ріг, 25–27 берез. 2025 р.). Кривий Ріг: КНУ, 2025. С. 175–178.
2. Волков О. М., Савіцький Р. С. Вплив Universal Sentence Encoder на розвиток обробки природної мови. *Інформаційні технології – 2025*: тези доп. XII Всеукр. наук.-практ. конф. молодих учених (Київ, 15 трав. 2025 р.). Київ: Київ. столич. ун-т ім. Б. Грінченка, 2025. С. 18.
3. Кабінет Міністрів України. Деякі питання доступності інформаційно-комунікаційних систем та документів в електронній формі : Постанова від 21.07.2023 № 757. URL: https://zakon.rada.gov.ua/go/757-2023-%D0%BF (дата звернення: 12.05.2026).
4. Кабінет Міністрів України. Про внесення змін до Національної стратегії із створення безбар'єрного простору в Україні на період до 2030 року : Розпорядження від 25.03.2025 № 294-р. URL: https://zakon.rada.gov.ua/go/294-2025-%D1%80 (дата звернення: 12.05.2026).
5. Карапет Л. В., Савіцький Р. С. Платформа для навчання з обмеженими можливостями. *Комп'ютерні технології: інновації, проблеми, рішення*: тези доп. VI Всеукр. наук.-техн. конф. (Житомир, 29–30 листоп. 2023 р.). Житомир: Житомирська політехніка, 2023. С. 213–214.
6. Савіцький Р. С., Вакалюк Т. А. Гібридна система генерації лексичних замін україномовного тексту для підвищення когнітивної доступності. *Вісник Хмельницького національного університету. Серія: Технічні науки*. 2026. № 2 (363). С. 182–188. DOI: 10.31891/2307–5732–2026–363–24.
7. Савіцький Р. С. Безбар'єрність вебпорталів освітніх навчальних закладів України. *Технічна інженерія*. 2023. № 1 (91). С. 172–177. DOI: 10.26642/ten-2023-1(91)-172-177
8. Савіцький Р. С. Гібридна система виявлення термінів в українськомовних текстах для забезпечення когнітивної доступності. *Computer Science and*

Applied Mathematics. 2025. № 2. С. 47–54. DOI: 10.26661/2786-6254-2025-2-06.

9. Савіцький Р. С., М’яновська А. Ю. Інтеграція бібліотек NLP та попередньо навчених моделей для автоматичної оцінки складності тексту. *Вісник Херсонського національного технічного університету*. 2025. № 2 (93). С. 319–326. DOI: 10.35546/kntu2078-4481.2025.2.2.39.
10. Савіцький Р. С. Стандарти доступності у питанні вивчення когнітивних та інтелектуальних порушень. *Інформаційно-комп’ютерні технології – 2024*: тези XIV Міжнар. наук.-техн. конф. (Житомир, 28–29 берез. 2024 р.). Житомир, 2024. С. 164.
11. Савіцький Р. С. Синхронізація українського законодавства з ЄС для покращення вебдоступності в освіті. *Автоматизація та комп’ютерно-інтегровані технології у виробництві та освіті: стан, досягнення, перспективи розвитку*: тези всеукр. наук.-практ. інтернет-конф. (Черкаси, 11–17 берез. 2024 р.). Черкаси: Черкас. нац. ун-т ім. Б. Хмельницького, 2024. С. 268.
12. Савіцький Р. С., Єфремов Ю. М. Огляд сучасного стану веб-доступності в Україні. *Міжнародна наукова інтернет-конференція* (Тернопіль, Україна – Переворськ, Польща, 3–4 квіт. 2023 р.). 2023. С. 39.
13. Савіцький Р. С. Використання random forest алгоритму для пошуку термінів в українській мові. *Практичні та теоретичні питання розвитку науки та освіти*: матеріали XV Міжнар. наук.-практ. конф. (Львів, 29–30 серп. 2025 р.). Львів: Львівський науковий форум, 2025. С. 109–110.
14. Савіцький Р. С., Бабицький Б. О. Аналіз архітектур нейронних мереж для розпізнавання облич. *Інформаційні технології і автоматизація – 2025*: тези доп. XVIII Міжнар. наук.-практ. конф. (Одеса, 30–31 жовт. 2025 р.). Одеса: ОНТУ, 2025. С. 1032.
15. Фоменко В. А., Савіцький Р. С. Використання Hugging Hub як сервісу бекенду для розгортання мовних моделей. *Development of Education, Science and Business: Results 2025*: proceedings of the International scientific and practical Internet conference (Dnipro, 18–19 Dec. 2025). Dnipro, 2026. 538 p.

16. Фоменко В. А., Савіцький Р. С. Автоматизований пошук помилок в українському тексті з використанням моделей штучного інтелекту. *Комп'ютерні технології: інновації, проблеми, рішення*: тези VIII Всеукр. наук.-практ. конф. (Житомир, 2–3 груд. 2025 р.). Житомир: Житомирська політехніка, 2025. С. 197.
17. AccessibleEU. EN 301549:2021. Accessibility requirements for ICT products and services. 2023. URL: https://accessible-eu-centre.ec.europa.eu/content-corner/digital-library/en-3015492021-accessibility-requirements-ict-products-and-services_en (дата звернення: 12.05.2026).
18. Alim S. Web Accessibility of the Top Research-Intensive Universities in the UK. *SAGE Open*. 2021. Vol. 11, № 4. P. 1–18. DOI: 10.1177/21582440211056614.
19. Allam H., Makubvure L., Gyamfi B., Graham K. N., Akinwolere K. Text classification: How machine learning is revolutionizing text categorization. *Information*, 16(2), 130. 2025. DOI: 10.3390/info16020130.
20. Alqahtani Y., McGuire M., Chakraborty J., Feng J. H. Understanding How ADHD Affects Visual Information Processing. *Universal Access in Human-Computer Interaction. Multimodality and Assistive Environments*. 13th International Conference, UAHCI 2019, Held as Part of the 21st HCI International Conference, HCII 2019, Orlando, FL, USA, July 26–31, 2019, Proceedings, Part II. Springer-Verlag, Berlin, Heidelberg, 23–31. DOI: 10.1007/978-3-030-23563-5_3.
21. Al-Thanyyan S. S., Azmi A. M. Automated Text Simplification: A Survey. *ACM Computing Surveys*. 2021. Vol. 54, № 2. Article 43. P. 1–36. DOI: 10.1145/3442695.
22. Alva-Manchego F., Scarton C., Specia L. Data-driven sentence simplification: Survey and benchmark. *Computational Linguistics*. 2020. Vol. 46, № 1. P. 135–187. DOI: 10.1162/coli_a_00370.
23. Ara J., Sik-Lanyi C. Automated evaluation of accessibility issues of webpage content: tool and evaluation. *Scientific Reports*. 2025. Vol. 15. P. 9516. DOI: 10.1038/s41598-025-92192-5
24. Ara J., Sik-Lanyi C., Kelemen A., Guzsvinecz T. An inclusive framework for

automated web content accessibility evaluation. *Universal Access in the Information Society,* 24, 1581–1607 2025. DOI: 10.1007/s10209-024-01164-5.

25. Avramovich A., Yeari M. The role of executive functions in reading comprehension by adolescents with ADHD. *Reading & Writing Quarterly*. 2023. Vol. 39, № 4. P. 277–299. DOI: 10.1080/10573569.2022.2103054.
26. Aydin N., Erdem O. A., Tekerek A. Comparative analysis of traditional machine learning and transformer-based deep learning models for text classification. *Journal of Polytechnic*. 2025. Vol. 28, № 2. P. 445–452. DOI: 10.2339/politeknik.1469530.
27. Ball M., Hughes A., McCormack W. Dyslexia: An Irish Perspective. 2nd ed. Blackrock : Blackhall Publishing, 2005.
28. Bojanowski P., Grave E., Joulin A., Mikolov T. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*. 2017. Vol. 5. P. 135–146. DOI: 10.1162/tacl_a_00051.
29. Braun M. C., Wölfel M. Usability of natural user interfaces for people with intellectual disabilities. *Pervasive Computing Technologies for Healthcare*. PH 2023. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 572. Springer, Cham. DOI: 10.1007/978-3-031-59717-6_24
30. Brown Corpus of Ukrainian. URL: https://github.com/brown-uk/corpus.
31. Calvo R. A., D'Mello S. Affect Detection: An Interdisciplinary Review of Models, Methods, and Their Applications. *IEEE Transactions on Affective Computing*. 2010. Vol. 1, № 1. P. 18–37. DOI: 10.1109/T-AFFC.2010.1.
32. Campoverde-Molina M., Luján-Mora S., Valverde L. Accessibility of university websites worldwide: a systematic literature review. *Universal Access in the Information Society*. 2023. Vol. 22. P. 133–168. DOI: 10.1007/s10209-021-00825-z.
33. Catlin D., Blamires M. Designing robots for special needs education. *Technology, Knowledge and Learning*. 2019. Vol. 24, № 2. P. 291–313. DOI: 10.1007/s10758-018-9378-8
34. Chadwick D., Wesson C., Fullwood C. Internet access by people with intellectual

disabilities: Inequalities and opportunities. *Future Internet*. 2013. Vol. 5, № 3. P. 376–397. DOI: 10.3390/fi5030376.

35. Chall J. S., Dale E. Readability Revisited: The New Dale-Chall Readability Formula. Cambridge, MA : Brookline Books, 1995. 159 p.
36. Chang T. A., Bergen B. K. Language model behavior: A comprehensive survey. *Computational Linguistics*. 2024. Vol. 50, № 1. P. 293–350. DOI: 10.1162/coli_a_00492.
37. Chupryna V. A., Savitskyi R. S. The use of eye-tracking technologies in educational systems. *Актуальні питання розвитку інформаційних технологій*: матеріали VII Міжнар. конф. (Дніпро, 19 листопада 2025). Дніпро: Приазовський державний технічний університет, 2025. P. 152–154.
38. Coelho R., Mattos P., Tannock R. Attention-Deficit Hyperactivity Disorder (ADHD) and narrative discourse in older adults. *Dementia & Neuropsychologia*. 2018. Vol. 12, № 4. P. 374–379. DOI: 10.1590/1980-57642018dn12-040006.
39. Cognitive Accessibility at W3C. URL: https://www.w3.org/WAI/cognitive/ (дата звернення: 04.05.2026).
40. Coleman M., Liau T. L. A Computer Readability Formula Designed for Machine Scoring. *Journal of Applied Psychology*. 1975. Vol. 60, № 2. P. 283–284. DOI: 10.1037/h0076540
41. Cornelissen P. L., Hansen P. C., Hutton J. L., Evangelinou V., Stein J. F. Magnocellular visual function and children's single word reading. *Vision Research*. 1998. Vol. 38, № 3. P. 471–482. DOI: 10.1016/S0042-6989(97)00199-5
42. Crossley S. A. Developing linguistic constructs of text readability using natural language processing. *Scientific Studies of Reading*. 2024. Vol. 29, № 2. P. 138–160. DOI: 10.1080/10888438.2024.2422365.
43. Cutts M. Oxford Guide to Plain English. 5th ed. Oxford : Oxford University Press, 2020. 384 p.
44. Devlin J., Chang M.-W., Lee K., Toutanova K. BERT: Pre-training of deep bidirectional transformers for language understanding. *NAACL-HLT 2019*. 2019. Vol. 1. P. 4171–4186. DOI: 10.18653/v1/N19-1423.

45. Diagnostic and Statistical Manual of Mental Disorders. 5th ed. Arlington : American Psychiatric Association, 2013. 991 p. DOI: 10.1176/appi.books.9780890425596
46. D'Mello S., Graesser A. Dynamics of Affective States during Complex Learning. *Learning and Instruction*. 2012. Vol. 22, № 2. P. 145–157.
47. Du Pre E. A., Gilroy D. G., Miles T. R. Dyslexia at College. 3rd ed. London : Routledge, 2007. DOI: 10.4324/9780203944790
48. DuBay W. H. The Principles of Readability. Costa Mesa, CA : Impact Information, 2004. 76 p. URL: https://files.eric.ed.gov/fulltext/ED490073.pdf.
49. Dunne C. Design strategies and dyslexia: Improving the accessibility of course material for third-level students with dyslexia in Ireland. 2024.
50. DuPaul G. J., Gormley M. J., Laracy S. D. Comorbidity of LD and ADHD: Implications of DSM-5 for assessment and treatment. *Journal of Learning Disabilities*. 2013. Vol. 46, № 1. P. 43–51. DOI: 10.1177/0022219412464351.
51. DuPaul G. J., Langberg J. M. Educational impairments in children with ADHD. *ADHD in Children and Adults: A Handbook for Diagnosis and Treatment* / Ed. R. A. Barkley. New York : Guilford Press, 2015. P. 169–190.
52. Enco-Jáuregui L., Meneses-Claudio B., Auccacusi-Kañahuire M. Web accessibility for people with dyslexia: A systematic literature review. *EAI Endorsed Transactions on Pervasive Health and Technology*. 2023. Vol. 9, № 1. DOI 10.4108/eetpht.9.4274.
53. Espinosa-Zaragoza I., Abreu-Salas J., Lloret E., Moreda P., Palomar M. A review of research-based automatic text simplification tools. *Proceedings of the International Conference Recent Advances in Natural Language Processing*. 2023. P. 321–330. URL: https://aclanthology.org/2023.ranlp-1.36/ (дата звернення: 12.05.2026).
54. European Union. Directive (EU) 2016/2102 of the European Parliament and of the Council of 26 October 2016 on the accessibility of the websites and mobile applications of public sector bodies. *EUR-Lex*. 2016. URL: https://eur-lex.europa.eu/eli/dir/2016/2102/oj/eng (дата звернення: 12.05.2026).

55. European Union. Directive (EU) 2019/882 of the European Parliament and of the Council of 17 April 2019 on the accessibility requirements for products and services. *EUR-Lex*. 2019. URL: https://eur-lex.europa.eu/eli/dir/2019/882/oj/eng (дата звернення: 12.05.2026).
56. Flesch R. A New Readability Yardstick. *Journal of Applied Psychology*. 1948. Vol. 32, № 3. P. 221–233. DOI: 10.1037/h0057532.
57. Freyer N., Kempt H., Klöser L. Easy-read and large language models: on the ethical dimensions of LLM-based text simplification. *Ethics and Information Technology*. 2024. Vol. 26, № 50. DOI: 10.1007/s10676-024-09792-4.
58. Friedman M. G., Bryen D. N. Web Accessibility Design Recommendations for People with Cognitive Disabilities. *Technology and Disability*. 2007. Vol. 19, № 4. P. 205–212. DOI: 10.3233/TAD-2007–19406.
59. Gartland S., Flynn P., Carneiro M. A., Holloway G., de Sousa Fialho J., Cullen J., Hamilton E., Harris A., Cullen C. The state of web accessibility for people with cognitive disabilities: A rapid evidence assessment. *Behavioral Sciences*. 2022. Vol. 12, № 2. P. 26. DOI: 10.3390/bs12020026.
60. Gibson E. Linguistic Complexity: Locality of Syntactic Dependencies. *Cognition*. 1998. Vol. 68, № 1. P. 1–76. DOI: 10.1016/S0010–0277(98)00034–1.
61. Gibson E. The Dependency Locality Theory: A Distance-Based Theory of Linguistic Complexity. *Image, Language, Brain: Papers from the First Mind Articulation Project Symposium* / eds.: A. Marantz, Y. Miyashita, W. O'Neil. Cambridge, MA : MIT Press, 2000. P. 95–126. URL https://tedlab.mit.edu/tedlab_website/researchpapers/Gibson_2000_DLT.pdf (дата звернення: 04.05.2026).
62. Gunning R. The Technique of Clear Writing. Rev. ed. New York : McGraw-Hill, 1968. 329 p.
63. Hejer B. B., Khribi K., Jemni M. Towards accessible open educational resources: Overview and challenges. *6th International Conference on Information and Communication Technology and Accessibility*. 2017. P. 1–6. DOI: 10.1109/ICTA.2017.8336068.

64. Hu X., Gao J. Facial Expression Recognition Reveals Students' Engagement in Online Class: Correlations with Six Engagement Measurements. *PLOS ONE*. 2025. Vol. 20, № 10. e0334232. DOI: 10.1371/journal.pone.0334232.
65. Huang Z.-Y., Chiang C.-C., Chen J.-H., Chen Y.-C., Chung H.-L., Cai Y.-P., Hsu H.-C. A Study on Computer Vision for Facial Emotion Recognition. *Scientific Reports*. 2023. Vol. 13. 8425. DOI: 10.1038/s41598–023–35446–4.
66. Hughes A., Ball M., Bissett R., McCormack W. Living with Dyslexia. 2nd ed. Dublin : Dyslexia Association of Ireland and Tower Press, 2011.
67. IAB Tech Lab Content Taxonomy : Webshrinker IAB Website Categories v3 / Webshrinker. URL: https://docs.webshrinker.com/v3/website-category-api.html (дата звернення: 04.05.2026).
68. Ilona A. Public sector website accessibility statements – what you need to know. 2018. URL: https://gds.blog.gov.uk/2018/11/21/public-sector-website-accessibility-statements-what-you-need-to-know/ (дата звернення: 12.05.2026).
69. Ingram D. R. The concept of accessibility: A search for an operational form. *Regional Studies*. 1971. Vol. 5, № 2. P. 101–107. DOI: 10.1080/09595237100185131
70. Iniesto F., McAndrew P., Minocha S., Coughlan T. What are the expectations of disabled learners when participating in a MOOC? *L@S '17 proceedings of the fourth (2017) ACM conference on learning @ scale*. 2017. Pp. 225–228. New York: ACM. 2017. DOI: 10.1145/3051457.3053991
71. Introduction to Web Accessibility URL: https://www.w3.org/WAI/fundamentals/accessibility-intro/ (date of access: 10.05.2026).
72. Iwarsson S., Ståhl A. Accessibility, usability and universal design – Positioning and definition of concepts describing person-environment relationships. *Disability and Rehabilitation*. 2003. Vol. 25, № 2. P. 57–66. DOI: 10.1080/0963828021000007969
73. James A. What Subjects Do Dyslexic Students Study at University? 2004. URL: http://www.dyslexic.com/ (дата звернення: 10.01.2013).

74. Jayaswal R., Ansari M.A., Dixit M., Singh D., Ahmad S.. Advances in Facial Expression Recognition Technologies for Emotion Analysis. *Discover Computing* 28, 203. 2025. DOI: 10.1007/s10791–025–09699–8.
75. Joulin A., Grave E., Bojanowski P., Mikolov T. Bag of tricks for efficient text classification. *EACL 2017*. 2017. DOI: 10.18653/v1/E17-2068.
76. Kamal Taha, Paul D. Yoo, Chan Yeun, Dirar Homouz, Aya Taha. A comprehensive survey of text classification techniques and their research applications: Observational and experimental insights. *Engineering Applications of Artificial Intelligence*. 2024. Vol. 138. P. 109501. DOI; 10.1016/j.cosrev.2024.100664
77. Karreman J., van der Geest T., Buursink E. Accessible website content guidelines for users with intellectual disabilities. *Journal of Applied Research in Intellectual Disabilities*. 2007. Vol. 20, № 6. P. 510–518. DOI: 10.1111/j.1468-3148.2006.00353.x
78. Khanna M. S., Kendall P. C. Bringing technology to training: web-based therapist training to promote the development of competent cognitive-behavioral therapists. *Cognitive and Behavioral Practice*. 2015. Vol. 22, № 3. P. 291–301. DOI: 10.1016/j.cbpra.2015.02.002.
79. Kincaid J. P., Fishburne R. P., Rogers R. L., Chissom B. S. Derivation of New Readability Formulas (Automated Readability Index, Fog Count and Flesch Reading Ease Formula) for Navy Enlisted Personnel : Research Branch Report 8–75. Millington, TN : Naval Technical Training, U. S. Naval Air Station Memphis, 1975.
80. Kuriyan A. B., Pelham W. E., Molina B. S. G., Waschbusch D. A., Gnagy E. M., Sibley M. H., Babinski D. E., Walther C., Cheong J., Yu J., Kent K. M. Young adult educational and vocational outcomes of children diagnosed with ADHD. *Journal of Abnormal Child Psychology*. 2013. Vol. 41. P. 27–41. DOI: 10.1007/s10802-012-9658-z.
81. Laasonen M., Lehtinen M., Leppämäki S., Tani P., Hokkanen L. Project DyAdd: Phonological Processing, Reading, Spelling, and Arithmetic in Adults With Dyslexia or ADHD. *Journal of Learning Disabilities*. 2010. Vol. 43, № 1. P. 3–14.

DOI: 10.1177/0022219409335216.

82. Lagacé-Leblanc J., Massé L., Rousseau N. Academic Impairments Faced by College Students with Attention-Deficit Hyperactivity Disorder: A Qualitative Study. *Journal of Postsecondary Education and Disability*. 2022. Vol. 35, № 2. P. 131–144. URL: https://files.eric.ed.gov/fulltext/EJ1364189.pdf (дата звернення: 04.05.2026).
83. Le Pochat V., Van Goethem T., Tajalizadehkhoob S., Korczyński M., Joosen W. Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. *Proceedings of the 26th Annual Network and Distributed System Security Symposium (NDSS 2019), San Diego, CA, USA, February 24–27, 2019*. Reston, VA : Internet Society, 2019. DOI: 10.14722/ndss.2019.23386.
84. Le Q. V., Mikolov T. Distributed Representations of Sentences and Documents. *Proceedings of the 31st International Conference on Machine Learning (ICML), Beijing, China, June 21–26, 2014*. PMLR. 2014. Vol. 32, № 2. P. 1188–1196. UR: https://proceedings.mlr.press/v32/le14.pdf (дата звернення: 04.05.2026).
85. Legendre A. M. Nouvelles méthodes pour la détermination des orbites des comètes. Paris : F. Didot, 1805. VIII, 80 p.
86. Li S., Deng W. Deep Facial Expression Recognition: A Survey. *IEEE Transactions on Affective Computing*. 2022. Vol. 13, № 3. P. 1195–1215. DOI: 10.1109/TAFFC.2020.2981446.
87. Liu F., Jin T., Lee J. S. Y. Automatic readability assessment for sentences: neural, hybrid and large language models. *Language Resources and Evaluation*. 2025. Vol. 59. P. 2265–2296. DOI: 10.1007/s10579-024-09800-5.
88. Liu T., De Costa F., Yasin M. A. I. The Impact of Dyslexia on the Effectiveness of Online Learning: A Systematic Literature Review. *Studies in Media and Communication*. 2023. Vol. 11, № 6. DOI: 10.11114/SMC.V11I6.5960.
89. Maia N., Nabais Sá M. J., Melo-Pires M. et al. Intellectual disability genomics: current state, pitfalls and future challenges. *BMC Genomics*. 2021. Vol. 22. P. 909. DOI: 10.1186/s12864-021-08227-4.
90. Martínez P., Ramos A., Moreno L. Exploring large language models to generate

Easy to Read content. *Frontiers in Computer Science*. 2024. Vol. 6. P. 1394705. DOI: 10.3389/fcomp.2024.1394705.

91. McCarthy J. E., Swierenga S. J. What we know about dyslexia and web accessibility: A research review. *Universal Access in the Information Society*. 2010. Vol. 9, № 2. P. 147–152. DOI: 10.1007/s10209-009-0160-5.
92. McLaughlin G. H. SMOG Grading – a New Readability Formula. *Journal of Reading*. 1969. Vol. 12, № 8. P. 639–646. URL: https://ogg.osu.edu/media/documents/health_lit/WRRSMOG_Readability_Formula_G._Harry_McLaughlin__1969_.pdf (дата звернення: 12.05.2026).
93. MDN Web Docs. Cognitive accessibility. 2025. URL: https://developer.mozilla.org/en-US/docs/Web/Accessibility/Guides/Cognitive_accessibility (дата звернення: 12.05.2026).
94. Mikk J. Sentence Length for Revealing the Cognitive Load Reversal Effect in Text Comprehension. *Educational Studies*. 2008. Vol. 34, № 2. P. 119–127. DOI: 10.1080/03055690701811164.
95. Mikolov T., Chen K., Corrado G., Dean J. Efficient Estimation of Word Representations in Vector Space. *Proceedings of the Workshop at International Conference on Learning Representations (ICLR 2013), Scottsdale, Arizona, USA, May 2–4, 2013*. P. 1–12. URL: https://www.scirp.org/reference/referencespapers?referenceid=3018666 (дата звернення: 04.05.2026).
96. Miranda A., Mercader J., Fernández M. I., Colomer C. Reading performance of young adults with ADHD diagnosed in childhood. *Journal of Attention Disorders*. 2017. Vol. 21, № 4. P. 294–304. DOI: 10.1177/1087054713507977
97. Moreno L., Petrie H., Martínez P., Alarcon R. Designing User Interfaces for Content Simplification Aimed at People with Cognitive Impairments. *Universal Access in the Information Society*. 2024. Vol. 23, № 1. P. 99–117. DOI: 10.1007/s10209–023–00986-z.
98. North K., Ranasinghe T., Shardlow M. et al. Deep learning approaches to lexical

simplification: A survey. *Journal of Intelligent Information Systems*. 2025. Vol. 63, № 1. P. 111–134. DOI: 10.1007/s10844-024-00882-9.

99. North K., Ranasinghe T., Shardlow M., Zampieri M. MultiLS: An end-to-end lexical simplification framework. *Proceedings of the Third Workshop on Text Simplification, Accessibility, and Readability*. 2024. P. 1–11. DOI: 10.18653/v1/2024.tsar-1.1

100. Ortiz C., Vidair H. B., Acri M., Chacko A., Kobak K. Pilot study of an online parent-training course for disruptive behavior with live remote coaching for practitioners. *Professional Psychology: Research and Practice*. 2020. Vol. 51, № 2. P. 125–133. DOI: 10.1037/pro0000286.

101. Paetzold G. H., Specia L. A Survey on Lexical Simplification. *Journal of Artificial Intelligence Research*. 2017. Vol. 60. P. 549–593. DOI: 10.1613/jair.5526.

102. Pan W., Li X., Chen X., Xu R. Textual form features for text readability assessment. *Natural Language Processing*. 2025. Vol. 31, № 3. P. 800–841. DOI: 10.1017/nlp.2024.50.

103. Pascual Almenara A., Sayago Barrantes S. Towards Designing a Set of Usability and Accessibility Heuristics Focused on Cognitive Diversity: An Exploratory Case Study with Generative Artificial Intelligence. *Information*. 2024. Vol. 15, № 12. Article 769. DOI: 10.3390/info15120769.

104. Pennington J., Socher R., Manning C. D. GloVe: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, October 25–29, 2014*. Stroudsburg, PA : Association for Computational Linguistics, 2014. P. 1532–1543. DOI: 10.3115/v1/D14-1162

105. Petrie H., Savva A., Power C. Towards a unified definition of web accessibility. *Proceedings of the 12th Web for All Conference*. 2015. Article 35, P. 1–13. DOI: 10.1145/2745555.2746653.

106. Pfiffner L. J., Dvorsky M. R., Friedman L. M. et al. Development of a Web-Based Training Platform for School Clinicians in Evidence-Based Practices for

ADHD. *School Mental Health*. 2023. Vol. 15. P. 49–66. DOI: 10.1007/s12310-022-09556-9.

107. Picard R. W. Affective Computing. Cambridge, MA : MIT Press, 1997. 306 p.

108. Poultourtzidis I., Katsouli A. M., Anastasiades S., Makroglou S., Sidiropoulos E., Bamidis P. D. Supporting digital inclusion and web accessibility for people with cognitive disabilities. *Studies in Health Technology and Informatics*. 2022. Vol. 294. P. 619–623. DOI: 10.3233/SHTI220543.

109. Preston C. C., Colman A. M. Optimal Number of Response Categories in Rating Scales: Reliability, Validity, Discriminating Power, and Respondent Preferences. *Acta Psychologica*. 2000. Vol. 104, № 1. P. 1–15. DOI: 10.1016/S0001–6918(99)00050–5.

110. Raymaker D. M., Kapp S. K., McDonald K. E., Weiner M., Ashkenazy E., Nicolaidis C. Development of the AASPIRE web accessibility guidelines for autistic web users. *Autism in Adulthood*. 2019. Vol. 1, № 2. P. 146–157. DOI: 10.1089/aut.2018.0020.

111. Report of the Task Force on Dyslexia / Department of Education and Science. Dublin : Stationery Office, 2001. 200 p.

112. Saadi I., Cunningham D. W., Taleb-Ahmed A., Hadid A., El Hillali Y. Driver's Facial Expression Recognition: A Comprehensive Survey. *Expert Systems with Applications*. 2024. Vol. 242. 122784. DOI: 10.1016/j.eswa.2023.122784.

113. Saman Jamshidi, Mahin Mohammadi, Saeed Bagheri, Hamid Esmaeili Najafabadi, Alireza Rezvanian, Mehdi Gheisari, Mustafa Ghaderzadeh, Amir Shahab Shahabi, Zongda Wu. Effective text classification using BERT, MTM LSTM, and DT. *Data & Knowledge Engineering*. 2024. Vol. 151. P. 102306. DOI: 10.1016/j.datak.2024.102306

114. Savchenko A. V. Facial Expression Recognition with Adaptive Frame Rate based on Multiple Testing Correction. *Proceedings of the 40th International Conference on Machine Learning (ICML 2023)*. PMLR, 2023. Vol. 202. P. 30119–30129. URL: https://proceedings.mlr.press/v202/savchenko23a.html. (дата звернення: 04.05.2026).

115. Savchenko A. V. Video-based Frame-level Facial Analysis of Affective Behavior on Mobile Devices using EfficientNets. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW 2022)*. 2022. P. 2358–2365. DOI: 10.1109/CVPRW56347.2022.00263.

116. Savchenko A. V., Savchenko L. V., Makarov I. Classifying Emotions and Engagement in Online Learning Based on a Single Facial Expression Recognition Neural Network. *IEEE Transactions on Affective Computing*. 2022. Vol. 13, № 4. P. 2132–2143. DOI: 10.1109/TAFFC.2022.3188390.

117. Scholze D.A., Gosdin M.M., Perez S.L., Schweitzer J.B.. Identifying the Information Needs and Format Preferences for Web-Based Content Among Adults With or Parents of Children With Attention-Deficit/Hyperactivity Disorder. *JMIR Formative Research*. 2023. Vol. 7. e47409. DOI: 10.2196/47409.

118. Section 508 of the Rehabilitation Act, as amended. URL: https://www.section508.gov/manage/laws-and-policies/section-508-law/ (дата звернення: 12.05.2026).

119. Serengil S. I., Ozpinar A. A Benchmark of Facial Recognition Pipelines and Co-Usability Performances of Modules. *Journal of Information Technologies / Bilişim Teknolojileri Dergisi*. 2024. Vol. 17, № 2. P. 95–107. DOI: 10.17671/gazibtd.1399077.

120. Sigurd B., Eeg-Olofsson M., van de Weijer J. Word Length, Sentence Length and Frequency – Zipf Revisited. *Studia Linguistica*. 2004. Vol. 58, № 1. P. 37–52. DOI: 10.1111/j.0039–3193.2004.00109.x.

121. Singer J. Neurodiversity : The Birth of an Idea. Lexington, Kentucky, 2017. 82 p. URL: http://dickyricky.com/books/psych/NeuroDiversity%20-%20The%20Birth%20of%20an%20Idea%20-%20Judy%20Singer.pdf (дата звернення: 04.05.2026).

122. Skodzik T., Holling H., Pedersen A. Long-term memory performance in adult ADHD: A meta-analysis. *Journal of Attention Disorders*. 2017. Vol. 21, № 4. P. 267–283. DOI: 10.1177/1087054713510561.

123. Smith E. A., Senter R. J. Automated Readability Index : AMRL-TR-66–220.

Wright-Patterson AFB, OH : Aerospace Medical Research Laboratories, 1967. 14 p.

124. Soleymani M., Lichtenauer J., Pun T., Pantic M. A Multimodal Database for Affect Recognition and Implicit Tagging. *IEEE Transactions on Affective Computing*. 2012. Vol. 3, № 1. P. 42–55. DOI: 10.1109/T-AFFC.2011.25.

125. Spaniol M., Danielsson H. A meta-analysis of the executive function components inhibition, shifting, and attention in intellectual disabilities. *Journal of Intellectual Disability Research*. 2022. Vol. 66, № 1. P. 9–31. DOI: 10.1111/jir.12878.

126. Spearman C. The Proof and Measurement of Association between Two Things. *International Journal of Epidemiology*. 2010. Vol. 39, № 5. P. 1137-1150. DOI: 10.1093/ije/dyq191

127. Stenner A. J., Burdick H., Sanford E. E., Burdick D. S. The Lexile Framework for Reading Technical Report. Durham, NC : MetaMetrics, Inc., 2007. URL: https://metametricsinc.com/wp-content/uploads/2017/07/Stenner_Burdick_Sanford__Burdick-_The_LFR_Technical_Report.pdf. (дата звернення: 04.05.2026).

128. The WebAIM Million: The 2026 Report on the Accessibility of the Top 1,000,000 Home Pages / WebAIM. Logan, UT : Utah State University, 2026. URL: https://webaim.org/projects/million/ (дата звернення: 04.05.2026).

129. Tlili A., Denden M., Essalmi F., Jemni M., Chang M., Kinshuk, Chen N. S. Automatic modeling learner's personality using learning analytics approach in an intelligent Moodle learning platform. *Interactive Learning Environments*. 2019. Vol. 31, № 5. P. 2529–2543. DOI: 10.1080/10494820.2019.1636084.

130. Tucudean G., Bucos M., Dragulescu B., Caleanu C. D. Natural language processing with transformers: a review. *PeerJ Computer Science*. 2024. Vol. 10. e2222. DOI: 10.7717/peerj-cs.2222.

131. Vajjala S. Trends, Limitations and Open Challenges in Automatic Readability Assessment Research. *Proceedings of the Thirteenth Language Resources and Evaluation Conference (LREC 2022)*. Marseille : ELRA, 2022. P. 5366–5377.

URL: https://aclanthology.org/2022.lrec-1.574/. (дата звернення: 04.05.2026).

132. Viola P., Jones M. Rapid Object Detection Using a Boosted Cascade of Simple Features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001), Kauai, HI, USA, December 8–14, 2001*. Los Alamitos, CA : IEEE Computer Society, 2001. Vol. 1. DOI: 10.1109/CVPR.2001.990517.

133. Virnes M. Robotics in special needs education. *Proceedings of the 7th International Conference on Interaction Design and Children*. Chicago : ACM, 2008. P. 29–32. DOI: 10.1145/1463689.1463710

134. W3C. Authoring Tool Accessibility Guidelines (ATAG) 2.0. 2015. URL: https://www.w3.org/TR/ATAG20/ (дата звернення: 12.05.2026).

135. W3C. Essential Components of Web Accessibility. URL: https://www.w3.org/WAI/fundamentals/components/ (дата звернення: 12.05.2026).

136. W3C. Making content usable for people with cognitive and learning disabilities. URL: https://www.w3.org/TR/coga-usable/ (дата звернення: 12.05.2026).

137. W3C. User Agent Accessibility Guidelines (UAAG) Overview. URL: https://www.w3.org/WAI/standards-guidelines/uaag/ (дата звернення: 12.05.2026).

138. Web Content Accessibility Guidelines (WCAG) 2.2 : W3C Recommendation, 12 December 2024 / eds.: M. Cooper, A. Kirkpatrick, A. Campbell, R. Bradley Montgomery, C. Adams ; World Wide Web Consortium. URL: https://www.w3.org/TR/WCAG22/ (дата звернення: 04.05.2026).

139. What is Neurodiversity? / Genius Within C.I.C. URL: https://geniuswithin.org/what-is-neurodiversity/ (дата звернення: 04.05.2026).

140. What is neurodiversity? / Harvard Health Publishing. 2021. URL: https://www.health.harvard.edu/blog/what-is-neurodiversity-202111232645 (дата звернення: 12.05.2026).

141. Whitney C., Cornelissen P. Letter-position encoding and dyslexia. *Journal of Research in Reading*. 2005. Vol. 28, № 3. P. 274–301. DOI: 10.1111/j.1467-

9817.2005.00270.x.

142. WHO. Disability and health. 2023. URL: https://www.who.int/en/news-room/fact-sheets/detail/disability-and-health (дата звернення: 12.05.2026).

143. Woollams A. M. Connectionist neuropsychology: uncovering ultimate causes of acquired dyslexia. *Philosophical Transactions of The Royal Society*. 2014. Vol. 369, № 1634. DOI: 10.1098/rstb.2012.0398.

144. Xie T. Comparing KNN, Logistic Regression, Random Forest and BERT fine-tuning for scam message detection. *Finance & Economics*. 2025. Vol. 1, № 5. DOI: 10.61173/e19gh209

145. Xiong Y., Chen G., Cao J. Research on public service request text classification based on BERT-BiLSTM-CNN feature fusion. *Applied Sciences*. 2024. Vol. 14, № 14., 6282. DOI: 10.3390/app14146282.

146. Yujia Wu, Jun Wan. A survey of text classification based on pre-trained language model. *Neurocomputing*. 2025. Vol. 616. P. 128921. DOI: 10.1016/j.neucom.2024.128921.

147. Zervas P., Kardaras V., Sampson D. G. An online educational portal for supporting open access to teaching and learning of people with disabilities. *2014 IEEE 14th International Conference on Advanced Learning Technologies*, Athens, Greece, 2014. Pp. 564-565. DOI: 10.1109/ICALT.2014.165.

148. Zhang H., Shafiq M. O. Survey of transformers and towards ensemble learning using transformers for natural language processing. *Journal of Big Data*, 11, 25. 2024. DOI: 10.1186/s40537-023-00842-0.

149. Zhang X., Tlili A., Nascimbeni F., Burgos D., Huang R., Chang T., Jemni M., Khribi M.. Accessibility within open educational resources and practices for disabled learners: a systematic literature review. *Smart Learning Environments*. 2020. Vol. 7. P. 1. DOI: 10.1186/s40561-019-0113-2.

150. Zhao Y, Yin M, Zhu C, Tan C, Hu S, Liu D. Can situations awaken emotions? The compilation and evaluation of the Emotional Situation Sentence System (ESSS). *PLOS ONE*. 2024. Vol. 16, № 7. e0305649. DOI: 10.1371/journal.pone.0305649.

151. Zorzi M., Barbiero C., Facoetti A., Lonciari I., Carrozzi M., Montico M., Bravar L., George F., Pech-Georgel C., Ziegler J. C. Extra-large letter spacing improves reading in dyslexia. *Proceedings of the National Academy of Sciences*. 2012. Vol. 109, № 28. P. 11455–11459. DOI: 10.1073/pnas.1205566109.

Додатки

Додаток А

Таблиця А.1

Склад словників та їх категоризація

Група ресурсу	**Файли у alhoritm/data/replacements**	**Обсяг**	**Роль у конвеєрі**
profanity_group	profanity.json vulg.json obscene.json	7276 записів після об’єднання	Нормалізація ненормативної лексики та образ
obsolete_group	obsolete.json arch.json	7623 записи після об’єднання	Пояснення або заміна застарілої лексики
complex_group	complex.json	674 записи	Спрощення складної лексики
slang	slang.json	3370 записів	Нормалізація розмовної лексики
abbr	abbr.json	1251 запис	Розкриття скорочень
rare	rare.json	3 записи	Точкове покриття рідкісних випадків
term_definitions	term_definitions.json	224 визначення	Детерміновані пояснення термінів

Таблиця А.2

Приклад агрегованого запису телеметрії на рівні події

Поле	**Приклад значення**
timestamp	2025-01-01T000000Z
rawEmotion	нейтральна
confidence	0.93
stableEmotionFast	нейтральна
stableEmotionSlow	нейтральна
stateFast	спокійний
stateSlow	спокійний
consensusState	`M`
negativeConsensus	false
emotionScores	карта ваг короткострокового та довгострокового каналів
stateScores	карта ваг короткострокового та довгострокового каналів

Таблиця А.3

Технологічний стек реалізації алгоритму за архітектурними шарами

Архітектурний шар	Технологія	Версія	Роль у комплексі
Клієнтський рівень	React	18.3.x	Побудова інтерактивного SPA-інтерфейсу з компонентною моделлю взаємодії
Клієнтський рівень	TypeScript	5.7.x	Типізація стану, API-контрактів та клієнтської координації
Клієнтський рівень	Vite	5.4.x	Збірка та локальна розробка клієнтського застосунку
Клієнтський рівень	Axios	1.7.x	HTTP-клієнт для виклику текстового API та emotion-api
Клієнтський рівень	i18next / react-i18next	23.x / 15.x	Інтернаціоналізація інтерфейсу та підтримка багатомовного клієнта
Сервіс аналізу та спрощення	Python	3.11	Основна мова реалізації серверного алгоритмічного контуру
Сервіс аналізу та спрощення	Flask	3.x	Вебфреймворк для реалізації API аналізу та спрощення
Рівень мовного аналізу	Stanza	>= 1.7	Токенізація, лематизація, POS-розмітка та синтаксичний розбір українського тексту
Рівень мовного аналізу	transformers	>= 4.30	Запуск моделей для термінологічної та контекстної NLP-обробки
Рівень мовного аналізу	pymorphy2	0.9.x	Морфологічний аналіз та узгодження словоформ під час замін
Сервіс емоційного контексту	DeepFace	залежить від середовища	Оцінювання емоцій за відеокадрами обличчя
Сервіс емоційного контексту	OpenCV	>= 4.5	Захоплення відеопотоку та детекція обличчя
Генеративний резервний рівень	LLM-клієнти	конфігуровано	Резервна генерація замін та перефразувань за контрольованими правилами
Тестування і спостережуваність	Playwright	1.57.x	E2E-перевірка клієнтських сценаріїв, у тому числі негативних

Таблиця А.4

Результати програмної калібрування параметрів стабілізації

Канал	θ	θ/c̄, %	α	n_min	T_react, кадрів	C	R, хв$^{-1}$	L, с
Емоції, КК	0,55	71	0,78	6	5	0,461	34,2	1,52
Емоції, ДК	0,50	65	0,78	4	5	0,548	39,2	1,62
Стан, КК	0,50	65	0,80	4	5	0,526	27,2	2,33
Стан, ДК	0,45	58	0,78	6	4	0,611	28,8	2,79

Таблиця А.5

Метрики часу виконання, що формуються у відповіді ітераційного спрощення

Поле метрики	**Од.**	**Зміст**
timings_ms.total	мс	Загальний час виконання запиту
timings_ms.analysis	мс	Час повторного аналізу після спрощення за увімкненого include_analysis
timings_ms.syntax	мс	Час синтаксичного етапу за увімкненого синтаксичного етапу
timings_ms.iterations[i]	мс	Час виконання і тої ітерації
detector_timings_ms.total.total	мс	Сума часу детекторів у межах усіх повторних детекцій
detector_timings_ms.total.profanity	мс	Сумарний час детектора ненормативної лексики
detector_timings_ms.total.abbr	мс	Сумарний час детектора абревіатур
detector_timings_ms.total.slang	мс	Сумарний час детектора сленгу
detector_timings_ms.total.terms	мс	Сумарний час детектора термінів
detector_timings_ms.total.obsolete	мс	Сумарний час детектора застарілої лексики
detector_timings_ms.total.complex	мс	Сумарний час детектора складної лексики
detector_timings_ms.total.rare	мс	Сумарний час детектора рідковживаної лексики

Таблиця 6

Ваги відображення емоцій DeepFace у простір когнітивних станів

Емоція	**Спокійний**	**Дезорієнтований**	**Перевантажений**	**Психологічна інтерпретація**
Нейтральна	1.0	0.0	0.0	Стан рівноваги без активації допоміжного режиму
Радість	1.0	0.0	0.0	Позитивний емоційний фон без когнітивного перевантаження
Гнів	0.0	0.0	1.0	Маркер фрустрації з високим когнітивним навантаженням
Відраза	0.0	0.0	0.9	Маркер фрустрації з високим когнітивним навантаженням
Страх	0.0	0.3	0.7	Реакція на когнітивне перевантаження зі складністю орієнтації
Сум	0.0	0.7	0.3	Маркер зниженого залучення з утрудненою орієнтацією
Здивування	0.2	0.8	0.0	Сигнал несподіваної зміни контексту

Таблиця 7

Параметри OLS-регресії для каліброваних метрик читабельності

Формула	Предиктор	β	SE(β)	t	R^2	RMSE	n
FK_ukr (2.2)	ASL	0,0650	0,0020	33,110	0,9950	0,0427	88
	ASW	0,8533	0,0262	32,619			
	β_0	−1,1972	0,0436	−27,491			
SMOG_ukr (2.3)	√(30P/S)	0,1530	0,0008	198,293	0,9978	0,0282	88
	β_0	−0,2137	0,0098	−21,754			
DL_ukr (2.4)	ASL	0,0669	0,0015	44,833	0,9968	0,0315	88
	PDW	0,0230	0,0007	34,636			
	β_0	0,1744	0,0114	15,244			

Таблиця 8

Валідація каліброваних метрик на незалежній підвибірці Brown-UK

Жанр	Очік. ранг	FK_ukr	SMOG_ukr	DL_ukr
Дитяча проза	1	0,9462	0,7255	0,7781
Побутова проза	2	1,4159	1,3365	1,1746
Публіцистика	3	1,9418	1,7027	1,7079
Науково-популярні	4	2,2335	2,0973	1,8684
Офіційно-ділові	5	2,6756	2,7427	2,4485
Наукові	6	2,7947	2,7958	2,6105
Спірмен ρ		**1,0**	**1,0**	**1,0**

Додаток Б

Список публікацій здобувача за темою дисертації
Савіцького Романа Святославовича
"Методи і програмні засоби аналізу вебдоступності в освітньому процесі для людей з когнітивними та інтелектуальними порушеннями"
зі спеціальності 121 "Інженерія програмного забезпечення"

Наукові праці в фахових виданнях України, в яких опубліковані основні наукові результати дисертації

1. Савіцький Р. С. Безбар'єрність вебпорталів освітніх навчальних закладів України. *Технічна інженерія.* 2023. № 1 (91). С. 172–177. DOI: 10.26642/ten-2023-1(91)-172-177
2. Савіцький Р. С. Гібридна система виявлення термінів в українськомовних текстах для забезпечення когнітивної доступності. *Computer Science and Applied Mathematics.* 2025. № 2. С. 47–54. DOI: 10.26661/2786-6254-2025-2-06.
3. Савіцький Р. С., М'яновська А. Ю. Інтеграція бібліотек NLP та попередньо навчених моделей для автоматичної оцінки складності тексту. *Вісник Херсонського національного технічного університету.* 2025. № 2 (93). С. 319–326. DOI: 10.35546/kntu2078-4481.2025.2.2.39.
4. Савіцький Р., Вакалюк Т. Гібридна система генерації лексичних замін україномовного тексту для підвищення когнітивної доступності. *Вісник Хмельницького національного університету. Серія: Технічні науки.* 2026. № 2 (363). С. 182–188. DOI: 10.31891/2307-5732-2026-363-24.

Наукові праці, що засвідчують апробацію матеріалів дисертації

5. Савіцький Р. С. Стандарти доступності у питанні вивчення когнітивних та інтелектуальних порушень. *Інформаційно-комп'ютерні технології – 2024*: *тези XIV Міжнар. наук.-техн. конф. (Житомир, 28–29 берез. 2024 р.).* Житомир, 2024. С. 164.

6. Савіцький Р. С. Синхронізація українського законодавства з ЄС для покращення вебдоступності в освіті. *Автоматизація та комп'ютерно-інтегровані технології у виробництві та освіті: стан, досягнення, перспективи розвитку: тези всеукр. наук.-практ. інтернет-конф. (Черкаси, 11–17 берез. 2024 р.).* Черкаси: Черкас. нац. ун-т ім. Б. Хмельницького, 2024. С. 268.
7. Бабицький Б. О., Савіцький Р. С., Желізко В. В. Аналіз методів розпізнавання облич. *Комп'ютерні інтелектуальні системи та мережі: тези доп. XVIII Всеукр. наук.-практ. WEB-конф. аспірантів, студентів та молодих вчених (Кривий Ріг, 25–27 берез. 2025 р.).* Кривий Ріг: КНУ, 2025. С. 175–178.
8. Фоменко В. А., Савіцький Р. С. Використання Hugging Hub як сервісу бекенду для розгортання мовних моделей. *Development of Education, Science and Business: Results 2025: proceedings of the International scientific and practical Internet conference (Dnipro, 18–19 Dec. 2025).* Dnipro, 2026. 538 p.
9. Савіцький Р. С., Єфремов Ю. М. Огляд сучасного стану веб-доступності в Україні. *Міжнародна наукова інтернет-конференція (Тернопіль, Україна – Переворськ, Польща, 3–4 квіт. 2023 р.).* 2023. С. 39.
10. Савіцький Р. С. Використання random forest алгоритму для пошуку термінів в українській мові. *Практичні та теоретичні питання розвитку науки та освіти: матеріали XV Міжнар. наук.-практ. конф. (Львів, 29–30 серп. 2025 р.).* Львів: Львівський науковий форум, 2025. С. 109–110.
11. Савіцький Р. С., Бабицький Б. О. Аналіз архітектур нейронних мереж для розпізнавання облич. *Інформаційні технології і автоматизація – 2025: тези доп. XVIII Міжнар. наук.-практ. конф. (Одеса, 30–31 жовт. 2025 р.).* Одеса: ОНТУ, 2025. С. 1032.

Наукові праці, які додатково відображають наукові результати дисертації:

12. Карапет Л. В., Савіцький Р. С. Платформа для навчання з обмеженими можливостями. *Комп'ютерні технології: інновації, проблеми, рішення: тези доп. VI Всеукр. наук.-техн. конф. (Житомир, 29–30 листоп. 2023 р.).* Житомир: Житомирська політехніка, 2023. С. 213–214.
13. Волков О. М., Савіцький Р. С. Вплив Universal Sentence Encoder на розвиток обробки природної мови. *Інформаційні технології – 2025: тези доп. XII Всеукр. наук.-практ. конф. молодих учених (Київ, 15 трав. 2025 р.).* Київ: Київ. столич. ун-т ім. Б. Грінченка, 2025. С. 18.
14. Фоменко В. А., Савіцький Р. С. Автоматизований пошук помилок в українському тексті з використанням моделей штучного інтелекту. *Комп'ютерні технології: інновації, проблеми, рішення: тези VIII Всеукр. наук.-практ. конф. (Житомир, 2–3 груд. 2025 р.).* Житомир: Житомирська політехніка, 2025. С. 197.
15. Chupryna V. A., Savitskyi R. S. The use of eye-tracking technologies in educational systems. *Актуальні питання розвитку інформаційних технологій: матеріали VII Міжнар. конф.* (Дніпро, 19 листопада 2025). Дніпро: Приазовський державний технічний університет, 2025. С. 152–154.

Додаток В

Відомості про апробацію результатів дисертації
Савіцького Романа Святославовича
"Методи і алгоритми забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями"
зі спеціальності 121 "Інженерія програмного забезпечення"

Масові науково-практичні заходи міжнародного рівня:

1. Міжнародна наукова інтернет-конференція (Тернопіль, Україна – Переворськ, Польща, 3–4 квітня 2023 р.) . Форма участі – публікація матеріалів на тему «*Огляд сучасного стану веб-доступності в Україні»*.
2. XIV Міжнародна науково-технічна конференція «Інформаційно-комп'ютерні технології – 2024» (Житомир, 28–29 березня 2024 р.) . Форма участі – публікація матеріалів на тему *«Стандарти доступності у питанні вивчення когнітивних та інтелектуальних порушень»*.
3. XV Міжнародна науково-практична конференція «Практичні та теоретичні питання розвитку науки та освіти» (Львів, 29–30 серпня 2025 р.) . Форма участі – публікація матеріалів на тему *«Використання random forest алгоритму для пошуку термінів в українській мові».*
4. XVIII Міжнародна науково-практична конференція «Інформаційні технології і автоматизація – 2025» (Одеса, 30–31 жовтня 2025 р.) . Форма участі – публікація матеріалів на тему *«Аналіз архітектур нейронних мереж для розпізнавання облич».*
5. VII Міжнародна конференція «Актуальні питання розвитку інформаційних технологій» (Дніпро, 19 листопада 2025 р.) . Форма участі – публікація матеріалів на тему *«The use of eye-tracking technologies in educational systems»*.
6. Міжнародна науково-практична інтернет-конференція «Development of Education, Science and Business: Results 2025» (Дніпро, 18–19 грудня 2025

р.) . Форма участі – публікація матеріалів на тему *«Використання Hugging Hub як сервісу бекенду для розгортання мовних моделей».*

Масові науково-практичні заходи всеукраїнського рівня:

1. VI Всеукраїнська науково-технічна конференція «Комп'ютерні технології: інновації, проблеми, рішення» (Житомир, 29–30 листопада 2023 р.) . Форма участі – публікація матеріалів на тему *«Платформа для навчання з обмеженими можливостями».*
2. Всеукраїнська науково-практична інтернет-конференція «Автоматизація та комп'ютерно-інтегровані технології у виробництві та освіті: стан, досягнення, перспективи розвитку» (Черкаси, 11–17 березня 2024 р.) . Форма участі – публікація матеріалів на тему *«Синхронізація українського законодавства з ЄС для покращення вебдоступності в освіті».*
3. XVIII Всеукраїнська науково-практична WEB-конференція аспірантів, студентів та молодих вчених «Комп'ютерні інтелектуальні системи та мережі» (Кривий Ріг, 25–27 березня 2025 р.) . Форма участі – публікація матеріалів на тему *«Аналіз методів розпізнавання облич».*
4. XII Всеукраїнська науково-практична конференція молодих учених «Інформаційні технології – 2025» (Київ, 15 травня 2025 р.) . Форма участі – публікація матеріалів на тему *«Вплив Universal Sentence Encoder на розвиток обробки природної мови».*
5. VIII Всеукраїнська науково-практична конференція «Комп'ютерні технології: інновації, проблеми, рішення» (Житомир, 2–3 грудня 2025 р.) . Форма участі – публікація матеріалів на тему *«Автоматизований пошук помилок в українському тексті з використанням моделей штучного інтелекту».*

Масові науково-практичні заходи регіонального рівня:

1. Засідання міжкафедрального науково-методологічного семінару для молодих учених Державного університету "Житомирська політехніка"

(Україна, м. Житомир, Державний університет "Житомирська політехніка", 21 січня 2026 р.). Форма участі – виступ з презентацією за результатами дисертаційного дослідження на тему: *"Методи і алгоритми забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями"*.

Додаток Г

Довідки про впровадження результатів дисертації

Савіцького Романа Святославовича

"Методи і алгоритми забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями"

зі спеціальності 121 "Інженерія програмного забезпечення"

(СКАНОВАНІ КОПІЇ)

VMKGROUP

ТОВАРИСТВО З ОБМЕЖЕНОЮ ВІДПОВІДАЛЬНІСТЮ "ВМК ГРУП ЮА"

Житомир, вул. Пушкінська, будинок 54, квартира 2, 10002
ЄДРПОУ 44862102, тел. +38 (097) 284 -05 -84
IBAN: UA42 320478000002600692492691O в АБ "УКРГАЗБАНК"

ДОВІДКА
про впровадження результатів наукового дослідження
на тему «Методи і алгоритми забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями»
САВІЦЬКОГО Романа Святославовича
на здобуття наукового ступеня доктора філософії
за спеціальністю 121 Інженерія програмного забезпечення
у галузі знань 12 Інформаційні технології

Дана довідка підтверджує, що результати дисертаційного дослідження САВІЦЬКОГО Романа Святославовича на тему «Методи і алгоритми забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями» проходили апробацію в діяльності **ТОВ «ВМК ГРУП ЮА»**.

В рамках апробації на підприємстві було проведено експериментальне тестування розробленого методу забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями та програмного забезпечення, що забезпечує його практичну реалізацію. Тестування здійснювалося з використанням фактичних даних підприємства, отриманих із робочих інформаційних систем, які відповідають необхідній категорії.

За результатами проведеної апробації встановлено, що запропонований метод і програмне забезпечення є працездатними та придатними до практичного застосування.

Практичне значення отриманих результатів дисертаційного дослідження САВІЦЬКОГО Р.С. обумовлене актуальними потребами з виявлення та забезпечення доступності, що викликані взаємодією користувачів з когнітивними обмеженнями з матеріалами компанії. Розроблені здобувачем моделі, методи, алгоритми та технології мають прикладне значення, відповідають сучасним потребам і становлять інтерес для подальшого практичного використання.

Директор
ТОВ «ВМК ГРУП ЮА» **Максим ГОРОБЧУК**

Товариство з обмеженою відповідальністю «ВМК ГРУП ЮА» Ідентифікаційний код 44862102 Україна м.Житомир

ТОВ «ТІЄТОЕВРІ КРІЕЙТ УКРАЇНА»
Ідентифікаційний код 30522902
04071, м. Київ, вул. Верхній Вал , 28/12
Тел.: +380 44 585 25 00
E-mail: DLoffice.ukraine@tietoevry.com

ДОВІДКА
про впровадження результатів наукового дослідження
на тему «Методи і алгоритми забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями»
САВІЦЬКОГО Романа Святославовича
на здобуття наукового ступеня доктора філософії
за спеціальністю 121 Інженерія програмного забезпечення
у галузі знань 12 Інформаційні технології

«28» квітня 2026 р.

Дана довідка підтверджує, що результати дисертаційного дослідження САВІЦЬКОГО Романа Святославовича на тему «Методи і алгоритми забезпечення вебдоступності тексту для людей з когнітивними та інтелектуальними порушеннями» проходили апробацію в діяльності Товариства з обмеженою відповідальністю "ТІЕТОЕВРІ КРІЕЙТ УКРАЇНА" (TietoEvry Create Ukraine LLC; ЄДРПОУ 30522902; місцезнаходження: 04071, м. Київ, вул. Верхній Вал, буд. 28/12) (далі — "Підприємство") у дослідницьких цілях.

В рамках апробації проведено тестування методу оцінювання когнітивної доступності тексту на основі індексу CAI, алгоритму ітеративного спрощення вебконтенту засобами словників синонімічних замін, а також підсистеми визначення емоційного стану користувача. Тестування здійснювалося на знеособлених, загальнодоступних матеріалах інформаційного характеру, які відповідають категорії текстів із підвищеною когнітивною складністю.

За результатами апробації встановлено, що у межах протестованих сценаріїв запропонований метод коректно визначає рівень когнітивної доступності тексту, забезпечує локалізацію лексичних бар'єрів та формує обґрунтовані рекомендації щодо їх усунення. У межах проведеної апробації програмне забезпечення продемонструвало працездатність та потенціал для подальшого наукового дослідження.

Практичне значення отриманих результатів дисертаційного дослідження САВІЦЬКОГО Р.С. обумовлене загальною актуальністю проблематики у підвищенні доступності текстових матеріалів для осіб з когнітивними особливостями. Розроблені методи, алгоритми та програмний

tieto

комплекс мають прикладне значення, відповідають актуальним вимогам вебдоступності та становлять інтерес у відповідній галузі досліджень.

Цю довідку видано виключно для подання до спеціалізованої вченої ради з метою академічного захисту дисертації САВІЦЬКОГО Р.С. Видача цієї довідки не є комерційним впровадженням результатів дослідження, не створює жодних договірних, фінансових чи інших юридичних зобов'язань та не розглядається як визнання Підприємством придатності розробок до самостійного промислового або комерційного використання.

Генеральний директор _____ **Олександр КОЛОТИНЕЦЬ**

Додаток Д

Вихідний код програмного комплексу

alhoritm/app.py

```
from flask import Flask, render_template, jsonify, Response, g, request
import logging
from flask_cors import CORS
from typing import Optional, Dict, Any
from config.app_config import BaseConfig
from config.settings import DEFAULT_DEV_CORS_ORIGINS, get_settings
from services.registry import get_registry
from utils.replacement_service import ReplacementService
from utils.errors import ApiError, new_trace_id
from utils.logging_setup import configure_logging
from routers.slang import slang
from routers.profanity import profanity
from routers.abbr import abbr
from routers.term import term
from routers.rare import rare
from routers.obsolete import obsolete
from routers.complex import complex
from routers.parseFile import parseFile
from routers.syntax import syntax
from routers.cognitive_accessibility import cognitive_accessibility
from routers.replacements import replacements
from routers.survey import survey
from routers.problems import problems
from routers.analyze import analyze
from routers.iterative_simplify import iterative_simplify

def create_app(config: Optional[Dict[str, Any] | BaseConfig] = None) -> Flask:
    app = Flask(__name__)
    if config is None:
        settings = get_settings()
        app.config.update(settings.as_flask_config())
    elif isinstance(config, dict):
        app.config.update(config)
    else:
        app.config.from_object(config)

    configure_logging(logging.INFO)

    registry = get_registry()
    registry.register_instance("settings", settings if config is None else
get_settings())
    registry.register_factory(
        "replacement_service",
        lambda:
ReplacementService(enable_substring_heuristic=get_settings().ENABLE_SUBSTRING_HE
URISTIC_DEFAULT),
    )

    CORS(
        app,
        origins=app.config.get("CORS_ORIGINS", DEFAULT_DEV_CORS_ORIGINS),
        methods=app.config.get("CORS_METHODS", ["GET", "POST", "OPTIONS"]),
        allow_headers=app.config.get("CORS_HEADERS", ["Content-Type",
"Authorization"]),
    )
```

```
@app.before_request
def attach_trace_id():
    g.trace_id = request.headers.get("X-Trace-Id") or new_trace_id()

@app.after_request
def add_trace_id_header(response: Response):
    trace_id = getattr(g, "trace_id", None)
    if trace_id:
        response.headers["X-Trace-Id"] = trace_id
    return response

@app.after_request
def log_request(response: Response):
    logger = logging.getLogger("http")
    trace_id = getattr(g, "trace_id", None)
    logger.info(
        "%s %s %s",
        request.method,
        request.path,
        response.status_code,
        extra={"trace_id": trace_id},
    )
    return response

@app.errorhandler(ApiError)
def handle_api_error(err: ApiError):
    trace_id = getattr(g, "trace_id", None)
    payload = err.to_dict()
    payload["trace_id"] = trace_id
    return jsonify(payload), err.status_code

@app.errorhandler(Exception)
def handle_unexpected_error(err: Exception):
    trace_id = getattr(g, "trace_id", None)
    payload = {
        "error": "Internal server error",
        "code": "internal_error",
        "details": {},
        "trace_id": trace_id,
    }
    return jsonify(payload), 500

# Реєстрація маршрутів
app.register_blueprint(slang)
app.register_blueprint(profanity)
app.register_blueprint(term)
app.register_blueprint(abbr)
app.register_blueprint(rare)
app.register_blueprint(obsolete)
app.register_blueprint(complex)
app.register_blueprint(parseFile)
app.register_blueprint(syntax)
app.register_blueprint(cognitive_accessibility)
app.register_blueprint(replacements)
app.register_blueprint(survey)
app.register_blueprint(problems)
app.register_blueprint(analyze)
app.register_blueprint(iterative_simplify)
```

```
    # Маршрут для головної сторінки
    @app.route("/")
    def index():
        return render_template("index.html")

    @app.route("/openapi.json", methods=["GET"])
    def openapi_spec():
        try:
            with open("openapi.json", "r", encoding="utf-8") as f:
                return Response(f.read(), mimetype="application/json")
        except Exception as exc:
            return jsonify({"error": f"Failed to load OpenAPI spec: {exc}"}),
500

    @app.route("/docs", methods=["GET"])
    def swagger_ui():
        html = """
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>API Docs</title>
    <link rel="stylesheet" href="https://unpkg.com/swagger-ui-dist/swagger-
ui.css" />
  </head>
  <body>
    <div id="swagger-ui"></div>
    <script src="https://unpkg.com/swagger-ui-dist/swagger-ui-
bundle.js"></script>
    <script>
      SwaggerUIBundle({
        url: '/openapi.json',
        dom_id: '#swagger-ui'
      });
    </script>
  </body>
</html>
"""
        return Response(html, mimetype="text/html")

    return app


app = create_app()

if __name__ == "__main__":
    app.run(debug=True, port=5002)
```

alhoritm/config/app_config.py

```
from __future__ import annotations

from dataclasses import dataclass
from typing import List

from config.settings import DEFAULT_DEV_CORS_ORIGINS, get_settings


@dataclass
class BaseConfig:
```

```
    DEBUG: bool = False
    TESTING: bool = False
    ENV_NAME: str = "base"
    CORS_ORIGINS: List[str] = None
    CORS_METHODS: List[str] = None
    CORS_HEADERS: List[str] = None

    def __post_init__(self) -> None:
        if self.CORS_ORIGINS is None:
            self.CORS_ORIGINS = DEFAULT_DEV_CORS_ORIGINS.copy()
        if self.CORS_METHODS is None:
            self.CORS_METHODS = ["GET", "POST", "OPTIONS"]
        if self.CORS_HEADERS is None:
            self.CORS_HEADERS = ["Content-Type", "Authorization"]


class DevConfig(BaseConfig):
    DEBUG = True
    ENV_NAME = "development"


class TestConfig(BaseConfig):
    TESTING = True
    ENV_NAME = "testing"


class ProdConfig(BaseConfig):
    ENV_NAME = "production"


def load_config_from_env() -> BaseConfig:
    settings = get_settings()
    env = settings.APP_ENV
    if env in {"prod", "production"}:
        config: BaseConfig = ProdConfig()
    elif env in {"test", "testing"}:
        config = TestConfig()
    else:
        config = DevConfig()

    config.DEBUG = settings.DEBUG
    config.TESTING = settings.TESTING
    config.CORS_ORIGINS = settings.CORS_ORIGINS
    config.CORS_METHODS = settings.CORS_METHODS
    config.CORS_HEADERS = settings.CORS_HEADERS
    return config
```

alhoritm/config/experiment.py

```
"""
Centralised experiment/runtime configuration.

This module groups parameters that affect performance/behaviour so experiments
can
be reproduced and tuned in one place.
"""
from __future__ import annotations

from config.settings import get_settings
```

```
_settings = get_settings()

# ----------------------------
# Batch analysis caching
# ----------------------------

ANALYSIS_CACHE_MAX = _settings.ANALYSIS_CACHE_MAX

# Default confidence threshold used by probabilistic detectors (e.g. slang).
DEFAULT_CONFIDENCE_THRESHOLD = _settings.DEFAULT_CONFIDENCE_THRESHOLD


# ----------------------------
# Iterative simplify behaviour
# ----------------------------

ITERATIVE_SIMPLIFY_STRICT_POS_DEFAULT =
_settings.ITERATIVE_SIMPLIFY_STRICT_POS_DEFAULT

ITERATIVE_SIMPLIFY_MODE_DEFAULTS = _settings.ITERATIVE_SIMPLIFY_MODE_DEFAULTS

ITERATIVE_SIMPLIFY_MODE_LIMITS = _settings.ITERATIVE_SIMPLIFY_MODE_LIMITS

ITERATIVE_SIMPLIFY_REDETECT_EVERY_BATCHES =
_settings.ITERATIVE_SIMPLIFY_REDETECT_EVERY_BATCHES

# Cursor mode: keep expensive operations local to the cursor position.
CURSOR_LEXICAL_WINDOW_MAX_CHARS = _settings.CURSOR_LEXICAL_WINDOW_MAX_CHARS
CURSOR_SYNTAX_WINDOW_MAX_CHARS = _settings.CURSOR_SYNTAX_WINDOW_MAX_CHARS

# Syntax improvement defaults (rule-based / stanza-backed).
SYNTAX_MAX_ITERATIONS_DEFAULT = _settings.SYNTAX_MAX_ITERATIONS_DEFAULT
SYNTAX_MIN_IMPROVEMENT_DEFAULT = _settings.SYNTAX_MIN_IMPROVEMENT_DEFAULT


# ----------------------------
# Replacement dictionary lookup
# ----------------------------

# Disabled by default for accuracy and reproducibility.
ENABLE_SUBSTRING_HEURISTIC_DEFAULT =
_settings.ENABLE_SUBSTRING_HEURISTIC_DEFAULT

# We do NOT query any online LLM "by key" (per-word) during dictionary lookup.
ENABLE_LLM_BY_KEY_DEFAULT = _settings.ENABLE_LLM_BY_KEY_DEFAULT
```

alhoritm/config/settings.py

```
from __future__ import annotations

import os
from dataclasses import dataclass, field
from functools import lru_cache
from typing import List, Dict


DEFAULT_DEV_CORS_ORIGINS = ["http://localhost:5173", "http://127.0.0.1:5173"]


def _parse_bool(value: str | None, default: bool) -> bool:
```

```
    if value is None:
        return default
    return value.strip().lower() in {"1", "true", "yes", "on"}


def _parse_int(name: str, value: str | None, default: int, *, min_value: int |
None = None) -> int:
    if value is None or value == "":
        return default
    try:
        parsed = int(value)
    except ValueError as exc:
        raise ValueError(f"{name} must be an integer.") from exc
    if min_value is not None and parsed < min_value:
        raise ValueError(f"{name} must be >= {min_value}.")
    return parsed


def _parse_float(name: str, value: str | None, default: float, *, min_value:
float | None = None, max_value: float | None = None) -> float:
    if value is None or value == "":
        return default
    try:
        parsed = float(value)
    except ValueError as exc:
        raise ValueError(f"{name} must be a number.") from exc
    if min_value is not None and parsed < min_value:
        raise ValueError(f"{name} must be >= {min_value}.")
    if max_value is not None and parsed > max_value:
        raise ValueError(f"{name} must be <= {max_value}.")
    return parsed


def _parse_list(value: str | None, default: List[str]) -> List[str]:
    if not value:
        return default
    items = [item.strip() for item in value.split(",") if item.strip()]
    return items or default


@dataclass(frozen=True)
class Settings:
    # App environment
    APP_ENV: str = "development"
    DEBUG: bool = True
    TESTING: bool = False
    CORS_ORIGINS: List[str] = field(default_factory=lambda:
DEFAULT_DEV_CORS_ORIGINS.copy())
    CORS_METHODS: List[str] = field(default_factory=lambda: ["GET", "POST",
"OPTIONS"])
    CORS_HEADERS: List[str] = field(default_factory=lambda: ["Content-Type",
"Authorization"])

    # Experiment/runtime config
    ANALYSIS_CACHE_MAX: int = 64
    DEFAULT_CONFIDENCE_THRESHOLD: float = 0.7
    ITERATIVE_SIMPLIFY_STRICT_POS_DEFAULT: bool = True
    ITERATIVE_SIMPLIFY_MODE_DEFAULTS: Dict[str, Dict[str, int]] =
field(default_factory=lambda: {
        "document": {"max_changes": 20, "steps": 5},
```

```
        "page": {"max_changes": 10, "steps": 5},
        "cursor": {"max_changes": 10, "steps": 5},
        "custom": {"max_changes": 5, "steps": 1},
    })
    ITERATIVE_SIMPLIFY_MODE_LIMITS: Dict[str, Dict[str, int]] =
field(default_factory=lambda: {
        "document": {"max_changes": 50, "steps": 20},
        "page": {"max_changes": 25, "steps": 20},
        "cursor": {"max_changes": 25, "steps": 20},
        "custom": {"max_changes": 50, "steps": 50},
    })
    ITERATIVE_SIMPLIFY_REDETECT_EVERY_BATCHES: Dict[str, int] =
field(default_factory=lambda: {
        "document": 3,
        "page": 3,
        "cursor": 1,
        "custom": 1,
    })
    CURSOR_LEXICAL_WINDOW_MAX_CHARS: int = 1200
    CURSOR_SYNTAX_WINDOW_MAX_CHARS: int = 1200
    SYNTAX_MAX_ITERATIONS_DEFAULT: int = 1
    SYNTAX_MIN_IMPROVEMENT_DEFAULT: float = 0.01
    ENABLE_SUBSTRING_HEURISTIC_DEFAULT: bool = False
    ENABLE_LLM_BY_KEY_DEFAULT: bool = False

    def as_flask_config(self) -> dict:
        return {
            "DEBUG": self.DEBUG,
            "TESTING": self.TESTING,
            "CORS_ORIGINS": self.CORS_ORIGINS,
            "CORS_METHODS": self.CORS_METHODS,
            "CORS_HEADERS": self.CORS_HEADERS,
        }


@lru_cache(maxsize=1)
def get_settings() -> Settings:
    env = (os.getenv("APP_ENV") or os.getenv("FLASK_ENV") or
"development").lower()

    debug_default = True if env not in {"prod", "production"} else False
    testing_default = True if env in {"test", "testing"} else False

    return Settings(
        APP_ENV=env,
        DEBUG=_parse_bool(os.getenv("APP_DEBUG"), debug_default),
        TESTING=_parse_bool(os.getenv("APP_TESTING"), testing_default),
        CORS_ORIGINS=_parse_list(os.getenv("APP_CORS_ORIGINS"),
DEFAULT_DEV_CORS_ORIGINS.copy()),
        CORS_METHODS=_parse_list(os.getenv("APP_CORS_METHODS"), ["GET", "POST",
"OPTIONS"]),
        CORS_HEADERS=_parse_list(os.getenv("APP_CORS_HEADERS"), ["Content-Type",
"Authorization"]),
        ANALYSIS_CACHE_MAX=_parse_int("ANALYSIS_CACHE_MAX",
os.getenv("ANALYSIS_CACHE_MAX"), 64, min_value=1),
        DEFAULT_CONFIDENCE_THRESHOLD=_parse_float(
            "DEFAULT_CONFIDENCE_THRESHOLD",
            os.getenv("DEFAULT_CONFIDENCE_THRESHOLD"),
            0.7,
            min_value=0.0,
```

```
            max_value=1.0,
        ),
        ITERATIVE_SIMPLIFY_STRICT_POS_DEFAULT=_parse_bool(
            os.getenv("ITERATIVE_SIMPLIFY_STRICT_POS_DEFAULT"), True
        ),
        CURSOR_LEXICAL_WINDOW_MAX_CHARS=_parse_int(
            "CURSOR_LEXICAL_WINDOW_MAX_CHARS",
os.getenv("CURSOR_LEXICAL_WINDOW_MAX_CHARS"), 1200, min_value=1
        ),
        CURSOR_SYNTAX_WINDOW_MAX_CHARS=_parse_int(
            "CURSOR_SYNTAX_WINDOW_MAX_CHARS",
os.getenv("CURSOR_SYNTAX_WINDOW_MAX_CHARS"), 1200, min_value=1
        ),
        SYNTAX_MAX_ITERATIONS_DEFAULT=_parse_int(
            "SYNTAX_MAX_ITERATIONS_DEFAULT",
os.getenv("SYNTAX_MAX_ITERATIONS_DEFAULT"), 1, min_value=1
        ),
        SYNTAX_MIN_IMPROVEMENT_DEFAULT=_parse_float(
            "SYNTAX_MIN_IMPROVEMENT_DEFAULT",
os.getenv("SYNTAX_MIN_IMPROVEMENT_DEFAULT"), 0.01, min_value=0.0
        ),
        ENABLE_SUBSTRING_HEURISTIC_DEFAULT=_parse_bool(
            os.getenv("ENABLE_SUBSTRING_HEURISTIC_DEFAULT"), False
        ),
        ENABLE_LLM_BY_KEY_DEFAULT=_parse_bool(
            os.getenv("ENABLE_LLM_BY_KEY_DEFAULT"), False
        ),
    )
```

alhoritm/algorithms/base.py

```
from abc import ABC, abstractmethod
from typing import Dict, Any, Optional
import stanza

class SyntaxAlgorithm(ABC):
    """
    Базовий абстрактний клас для алгоритмів синтаксичного аналізу складності
тексту
    """

    def __init__(self, nlp_pipeline: Optional[stanza.Pipeline] = None):
        """
        Ініціалізація алгоритму

        Args:
            nlp_pipeline: Stanza NLP pipeline для аналізу тексту
        """
        self.nlp = nlp_pipeline
        self.name = self.__class__.__name__

    @abstractmethod
    def calculate(self, text: str) -> Dict[str, Any]:
        """
        Основний метод розрахунку метрики складності

        Args:
            text: Текст для аналізу

        Returns:
```

```
        Словник з результатами аналізу
    """
    pass

@abstractmethod
def interpret_score(self, score: float) -> str:
    """
    Інтерпретація числового результату

    Args:
        score: Числовий результат алгоритму

    Returns:
        Текстова інтерпретація результату
    """
    pass

def validate_text(self, text: str) -> bool:
    """
    Валідація вхідного тексту

    Args:
        text: Текст для перевірки

    Returns:
        True якщо текст валідний, False інакше
    """
    return bool(text and text.strip())

def preprocess_text(self, text: str) -> str:
    """
    Попередня обробка тексту

    Args:
        text: Вхідний текст

    Returns:
        Оброблений текст
    """
    return text.strip()

def get_basic_stats(self, text: str) -> Dict[str, int]:
    """
    Отримання базових статистик тексту

    Args:
        text: Текст для аналізу

    Returns:
        Словник з базовими статистиками
    """
    if not self.nlp:
        raise ValueError("NLP pipeline не ініціалізований")

    doc = self.nlp(text)

    total_sentences = len(doc.sentences)
    total_words = 0
    total_characters = len(text)
    total_characters_no_spaces = len(text.replace(' ', ''))
```

```
        for sentence in doc.sentences:
            for token in sentence.tokens:
                if token.text.isalpha():
                    total_words += 1

        return {
            'sentences': total_sentences,
            'words': total_words,
            'characters': total_characters,
            'characters_no_spaces': total_characters_no_spaces
        }

    def format_result(self, calculation_result: Dict[str, Any], interpretation:
str) -> Dict[str, Any]:
        """
        Форматування фінального результату

        Args:
            calculation_result: Результат розрахунків
            interpretation: Текстова інтерпретація

        Returns:
            Форматований результат
        """
        return {
            **calculation_result,
            'algorithm': self.name,
            'interpretation': interpretation
        }

    def analyze(self, text: str) -> Dict[str, Any]:
        """
        Повний аналіз тексту з валідацією та інтерпретацією

        Args:
            text: Текст для аналізу

        Returns:
            Повний результат аналізу
        """
        if not self.validate_text(text):
            return {
                'error': 'Порожній або невалідний текст',
                'algorithm': self.name
            }

        processed_text = self.preprocess_text(text)
        calculation_result = self.calculate(processed_text)

        # Отримуємо основну метрику для інтерпретації
        main_score = calculation_result.get('score', 0)
        interpretation = self.interpret_score(main_score)

        return self.format_result(calculation_result, interpretation)
```

alhoritm/algorithms/d_level.py

```
import math
import json
import os
```

```
from .base import SyntaxAlgorithm

class DLevelUkrainian(SyntaxAlgorithm):
    """
    D-Level (Dale-Chall) для української мови.

    Використовується авторська пілотна калібровка:
    DL_ukr = 0.0670 × ASL + 0.0231 × PDW + 0.1675
    """

    def __init__(self, nlp):
        super().__init__(nlp)
        self.name = "D-Level (Dale-Chall) Ukrainian"
        self.description = "Оцінка складності тексту на основі відсотка складних
слів та довжини речень"

        # Load common (easy) words dictionary
        self.common_words = self._load_common_words()

        # Load difficult words from existing dictionaries
        self.difficult_words = self._load_difficult_words()

        # Пілотні коефіцієнти, калібровані на українській вибірці текстів.
        self.sentence_coefficient = 0.0670
        self.difficult_word_coefficient = 0.0231
        self.base_constant = 0.1675

    def _load_common_words(self):
        """Load common Ukrainian words from multiple sources"""
        common_words = set()

        # Common words from abbr.py blacklist
        abbr_common_words = {
            # Прийменники та сполучники
            'до', 'на', 'за', 'під', 'над', 'при', 'від', 'з', 'у', 'в', 'о',
'об', 'про', 'по', 'через', 'між',
            'що', 'як', 'де', 'коли', 'чому', 'куди', 'звідки', 'або', 'та',
'і', 'й', 'але', 'проте', 'однак',

            # Займенники
            'я', 'ти', 'він', 'вона', 'воно', 'ми', 'ви', 'вони', 'той', 'та',
'те', 'ті', 'цей', 'ця', 'це', 'ці',
            'мій', 'твій', 'його', 'її', 'наш', 'ваш', 'їх', 'себе', 'свій',
'своя', 'своє', 'свої',

            # Дієслова-зв'язки та допоміжні
            'є', 'був', 'була', 'було', 'були', 'буду', 'будеш', 'буде',
'будемо', 'будете', 'будуть',
            'мати', 'має', 'маю', 'маєш', 'маємо', 'маєте', 'мають',

            # Часто вживані прикметники та прислівники
            'також', 'тому', 'тоді', 'тут', 'там', 'ось', 'ще', 'вже', 'завжди',
'ніколи', 'часто', 'рідко',
            'дуже', 'більше', 'менше', 'краще', 'гірше', 'добре', 'погано',
'швидко', 'повільно',

            # Кольори
            'білий', 'чорний', 'червоний', 'синій', 'зелений', 'жовтий',
'сірий', 'коричневий', 'рожевий', 'помаранчевий',
```

```
            # Числівники
            'один', 'два', 'три', 'чотири', 'п\'ять', 'шість', 'сім', 'вісім',
'дев\'ять', 'десять',
            'перший', 'другий', 'третій', 'останній', 'багато', 'мало',
'кілька', 'декілька',

            # Часто вживані іменники
            'час', 'рік', 'день', 'ніч', 'ранок', 'вечір', 'люди', 'людина',
'чоловік', 'жінка', 'дитина',
            'дім', 'будинок', 'місто', 'село', 'країна', 'світ', 'земля',
'небо', 'море', 'річка',
            'робота', 'справа', 'справи', 'діло', 'питання', 'проблема',
'рішення', 'відповідь',
            'період', 'момент', 'хвилина', 'година', 'тиждень', 'місяць',
'життя', 'смерть',
        }

        # Add additional common Ukrainian words
        additional_common_words = {
            'не', 'ні', 'так', 'ні', 'може', 'можна', 'треба', 'потрібно',
'хочу', 'хоче', 'хочеш',
            'знаю', 'знає', 'знаєш', 'розумію', 'розуміє', 'думаю', 'думає',
'кажу', 'каже',
            'роблю', 'робить', 'іду', 'йде', 'їм', 'їсть', 'п\'ю', 'п\'є',
'сплю', 'спить',
            'читаю', 'читає', 'пишу', 'пише', 'слухаю', 'слухає', 'дивлюся',
'дивиться',
            'новий', 'стара', 'великий', 'маленький', 'довгий', 'короткий',
'високий', 'низький',
            'гарний', 'поганий', 'легкий', 'важкий', 'простий', 'складний',
'цікавий', 'нудний',
            'сьогодні', 'вчора', 'завтра', 'зараз', 'потім', 'раніше',
'пізніше', 'скоро',
            'дома', 'вдома', 'школа', 'університет', 'робота', 'магазин',
'лікарня', 'церква',
            'мама', 'тато', 'син', 'дочка', 'брат', 'сестра', 'дід', 'баба',
'друг', 'подруга',
            'говорю', 'говорить', 'слухаю', 'слухає', 'бачу', 'бачить', 'чую',
'чує'
        }

        common_words.update(abbr_common_words)
        common_words.update(additional_common_words)

        return common_words

    def _load_difficult_words(self):
        """Load difficult words from existing dictionaries"""
        difficult_words = set()

        # Try to load from rare.json
        try:
            rare_path = os.path.join(os.path.dirname(__file__), '..', 'data',
'source', 'rare.json')
            if os.path.exists(rare_path):
                with open(rare_path, 'r', encoding='utf-8') as f:
                    rare_data = json.load(f)
                    for item in rare_data:
                        if isinstance(item, dict) and 'word' in item:
                            difficult_words.add(item['word'].lower())
```

```
                    elif isinstance(item, str):
                        difficult_words.add(item.lower())
        except Exception as e:
            print(f"Warning: Could not load rare.json: {e}")

        # Try to load from arch.json
        try:
            arch_path = os.path.join(os.path.dirname(__file__), '..', 'data',
'source', 'arch.json')
            if os.path.exists(arch_path):
                with open(arch_path, 'r', encoding='utf-8') as f:
                    arch_data = json.load(f)
                    for item in arch_data:
                        if isinstance(item, dict) and 'word' in item:
                            difficult_words.add(item['word'].lower())
                        elif isinstance(item, str):
                            difficult_words.add(item.lower())
        except Exception as e:
            print(f"Warning: Could not load arch.json: {e}")

        return difficult_words

    def _is_difficult_word(self, word):
        """Check if a word is considered difficult"""
        word_lower = word.lower()

        # If word is in common words, it's not difficult
        if word_lower in self.common_words:
            return False

        # If word is explicitly in difficult words, it's difficult
        if word_lower in self.difficult_words:
            return True

        # Additional heuristics for Ukrainian difficult words
        # Long words (>7 characters) are likely more difficult
        if len(word) > 7:
            return True

        # Words with specific endings that indicate complexity
        difficult_endings = [
            'ість', 'ність', 'ство', 'ння', 'ення', 'ання', 'ування', 'ізація',
            'ний', 'льний', 'ський', 'цький', 'тський', 'івський'
        ]

        for ending in difficult_endings:
            if word_lower.endswith(ending):
                return True

        # By default, assume unknown words are difficult
        return True

    def calculate(self, text):
        """Calculate D-Level score for Ukrainian text"""
        try:
            # Get basic statistics
            stats = self.get_basic_stats(text)

            if stats['sentences'] == 0 or stats['words'] == 0:
                return {
```

```
                'd_level_score': 0,
                'difficult_words': 0,
                'difficult_percentage': 0,
                'avg_sentence_length': 0,
                'total_words': 0,
                'total_sentences': 0,
                'difficult_examples': []
            }

        # Process text with Stanza
        doc = self.nlp(text)

        # Count difficult words
        difficult_words = []
        difficult_examples = []

        for sentence in doc.sentences:
            for word in sentence.words:
                word_text = word.text.lower()

                # Skip punctuation and short words
                if len(word_text) < 2 or not word_text.isalpha():
                    continue

                if self._is_difficult_word(word_text):
                    difficult_words.append(word_text)
                    if len(difficult_examples) < 10:  # Limit examples
                        difficult_examples.append({
                            'word': word.text,
                            'lemma': word.lemma,
                            'pos': word.upos
                        })

        # Calculate statistics
        total_words = stats['words']
        total_sentences = stats['sentences']
        difficult_count = len(difficult_words)
        difficult_percentage = (difficult_count / total_words) * 100 if
total_words > 0 else 0
        avg_sentence_length = total_words / total_sentences if
total_sentences > 0 else 0

        d_level_score = (
            self.sentence_coefficient * avg_sentence_length +
            self.difficult_word_coefficient * difficult_percentage +
            self.base_constant
        )

        return {
            'd_level_score': round(d_level_score, 2),
            'score': round(d_level_score, 2),  # Для сумісності з базовим
класом
            'difficult_words': difficult_count,
            'difficult_percentage': round(difficult_percentage, 2),
            'avg_sentence_length': round(avg_sentence_length, 2),
            'total_words': total_words,
            'total_sentences': total_sentences,
            'difficult_examples': difficult_examples[:10]  # First 10
examples
        }
```

```
        except Exception as e:
            print(f"Error calculating D-Level: {e}")
            return {
                'd_level_score': 0,
                'difficult_words': 0,
                'difficult_percentage': 0,
                'avg_sentence_length': 0,
                'total_words': 0,
                'total_sentences': 0,
                'difficult_examples': []
            }

    def interpret_score(self, score):
        """Інтерпретація каліброваної шкали складності."""
        if score < 1.5:
            return "Простий рівень"
        elif score < 2.5:
            return "Середній рівень"
        else:
            return "Складний рівень"

    def build_detailed_interpretation(self, calculation_result):
        """Формує розгорнуту інтерпретацію для відповіді API."""
        d_level = calculation_result.get('d_level_score', 0)
        difficult_percentage = calculation_result.get('difficult_percentage', 0)

        if d_level < 1.5:
            education_level = "Клас 1: простий текст"
            readability = "Простий рівень"
        elif d_level < 2.5:
            education_level = "Клас 2: середній текст"
            readability = "Середній рівень"
        else:
            education_level = "Клас 3: складний текст"
            readability = "Складний рівень"

        complexity = "Низька"
        if difficult_percentage > 30:
            complexity = "Висока"
        elif difficult_percentage > 20:
            complexity = "Помірна"

        if difficult_percentage > 15:
            complexity_analysis = (
                f"Високий відсоток складних слів ({difficult_percentage}%). "
                "Рекомендується спрощення лексики."
            )
        elif difficult_percentage > 10:
            complexity_analysis = (
                f"Помірний відсоток складних слів ({difficult_percentage}%). "
                "Текст може бути складним для частини читачів."
            )
        else:
            complexity_analysis = (
                f"Низький відсоток складних слів ({difficult_percentage}%). "
                "Лексичний бар'єр виражений слабо."
            )

        return {
```

```
            'education_level': education_level,
            'readability': readability,
            'complexity': complexity,
            'complexity_analysis': complexity_analysis,
            'score_explanation': f"D-Level: {d_level}, складні слова:
{difficult_percentage}%"
        }

    def analyze(self, text):
        """
        Перевизначений метод analyze для D-Level алгоритму
        """
        if not self.validate_text(text):
            return {
                'error': 'Порожній або невалідний текст',
                'algorithm': self.name
            }

        processed_text = self.preprocess_text(text)
        calculation_result = self.calculate(processed_text)

        detailed_interpretation =
self.build_detailed_interpretation(calculation_result)

        return {
            **calculation_result,
            'algorithm': self.name,
            'interpretation': detailed_interpretation.get('readability', ''),
            'detailed_interpretation': detailed_interpretation
        }
```

alhoritm/algorithms/flesch_kincaid.py

```
import re
from typing import Dict, Any
from .base import SyntaxAlgorithm

class FleschKincaidUkrainian(SyntaxAlgorithm):
    """
    Flesch-Kincaid для української мови.

    У реалізації збережено класичний Flesch Reading Ease як довідковий
    показник (`flesch_score`) для сумісності з наявним API, але основним
    каліброваним показником складності є `grade_level`.
    """

    def __init__(self, nlp_pipeline=None):
        super().__init__(nlp_pipeline)
        self.name = "Flesch-Kincaid Ukrainian"

    def count_syllables_ukrainian(self, word: str) -> int:
        """
        Підрахунок складів в українському слові
        Базується на голосних звуках української мови
        """
        # Українські голосні
        vowels = 'аеєиіїоуюя'
        word = word.lower()

        # Видаляємо розділові знаки
```

```
        word = re.sub(r'[^\u0400-\u04FF]', '', word)

        if not word:
            return 0

        syllable_count = 0
        prev_was_vowel = False

        for char in word:
            is_vowel = char in vowels
            if is_vowel and not prev_was_vowel:
                syllable_count += 1
            prev_was_vowel = is_vowel

        # Мінімум 1 склад для будь-якого слова
        return max(1, syllable_count)

    def calculate(self, text: str) -> Dict[str, Any]:
        """
        Розрахунок індексу Flesch-Kincaid для української мови
        """
        if not self.nlp:
            raise ValueError("NLP pipeline не ініціалізований")

        doc = self.nlp(text)

        total_sentences = len(doc.sentences)
        total_words = 0
        total_syllables = 0

        for sentence in doc.sentences:
            for token in sentence.tokens:
                # Рахуємо тільки слова (не розділові знаки)
                if re.match(r'[а-яіїєґ]', token.text.lower()):
                    total_words += 1
                    syllables = self.count_syllables_ukrainian(token.text)
                    total_syllables += syllables

        if total_sentences == 0 or total_words == 0:
            return {
                'score': 0,
                'flesch_score': 0,
                'grade_level': 0,
                'sentences': total_sentences,
                'words': total_words,
                'syllables': total_syllables,
                'avg_sentence_length': 0,
                'avg_syllables_per_word': 0
            }

        # Розрахунок середніх значень
        avg_sentence_length = total_words / total_sentences
        avg_syllables_per_word = total_syllables / total_words

        # Залишаємо класичний FRE як довідковий baseline.
        flesch_score = 206.835 - (1.015 * avg_sentence_length) - (84.6 *
avg_syllables_per_word)

        # Авторська пілотна калібровка для української мови:
        # FK_ukr = 0.0646 * ASL + 0.8555 * ASW - 1.1971
```

```
        grade_level = 0.0646 * avg_sentence_length + 0.8555 *
avg_syllables_per_word - 1.1971

        # Обмежуємо значення в розумних межах
        flesch_score = max(0, min(100, flesch_score))
        grade_level = max(0, grade_level)

        return {
            'score': round(grade_level, 2),  # Основний калібрований показник
складності
            'flesch_score': round(flesch_score, 2),
            'grade_level': round(grade_level, 2),
            'sentences': total_sentences,
            'words': total_words,
            'syllables': total_syllables,
            'avg_sentence_length': round(avg_sentence_length, 2),
            'avg_syllables_per_word': round(avg_syllables_per_word, 2)
        }

    def interpret_score(self, score: float) -> str:
        """
        Інтерпретація каліброваного показника складності.
        """
        if score < 1.5:
            return "Простий рівень"
        elif score < 2.5:
            return "Середній рівень"
        else:
            return "Складний рівень"

    def interpret_grade_level(self, grade: float) -> str:
        """
        Інтерпретація каліброваної шкали.
        """
        if grade < 1.5:
            return "Клас 1: простий текст"
        elif grade < 2.5:
            return "Клас 2: середній текст"
        else:
            return "Клас 3: складний текст"

    def format_result(self, calculation_result: Dict[str, Any], interpretation:
str) -> Dict[str, Any]:
        """
        Форматування результату з додатковою інтерпретацією grade level
        """
        result = super().format_result(calculation_result, interpretation)

        # Додаємо детальну інтерпретацію
        result['detailed_interpretation'] = {
            'readability': interpretation,
            'education_level':
self.interpret_grade_level(calculation_result.get('grade_level', 0))
        }

        return result
```

alhoritm/algorithms/smog.py

```
import re
import math
from typing import Dict, Any
from .base import SyntaxAlgorithm

class SMOGUkrainian(SyntaxAlgorithm):
    """
    SMOG для української мови.

    Використовується авторська пілотна калібровка:
    SMOG_ukr = 0.1528 × √(30P / S) − 0.2058
    """

    def __init__(self, nlp_pipeline=None):
        super().__init__(nlp_pipeline)
        self.name = "SMOG Ukrainian"
        # Пілотні коефіцієнти, калібровані на українській вибірці текстів.
        self.base_coefficient = 0.1528
        self.sentence_factor = 30
        self.base_grade = -0.2058

    def count_syllables_ukrainian(self, word: str) -> int:
        """
        Підрахунок складів в українському слові
        Базується на голосних звуках української мови
        """
        # Українські голосні
        vowels = 'аеєиіїоуюя'
        word = word.lower()

        # Видаляємо розділові знаки
        word = re.sub(r'[^\u0400-\u04FF]', '', word)

        if not word:
            return 0

        syllable_count = 0
        prev_was_vowel = False

        for char in word:
            is_vowel = char in vowels
            if is_vowel and not prev_was_vowel:
                syllable_count += 1
            prev_was_vowel = is_vowel

        # Мінімум 1 склад для будь-якого слова
        return max(1, syllable_count)

    def is_polysyllabic_ukrainian(self, word: str) -> bool:
        """
        Перевіряє, чи є слово полісилабічним (3+ складів) для української мови

        Args:
            word: Слово для перевірки

        Returns:
            True якщо слово має 3 або більше складів
        """
        # Видаляємо розділові знаки
```

```
        clean_word = re.sub(r'[^\u0400-\u04FFa-zA-Z]', '', word)

        if len(clean_word) < 3:  # Дуже короткі слова рідко полісилабічні
            return False

        syllable_count = self.count_syllables_ukrainian(word)
        return syllable_count >= 3

    def count_polysyllabic_words(self, text: str) -> Dict[str, Any]:
        """
        Підраховує полісилабічні слова в тексті

        Args:
            text: Текст для аналізу

        Returns:
            Словник з результатами підрахунку
        """
        if not self.nlp:
            raise ValueError("NLP pipeline не ініціалізований")

        doc = self.nlp(text)

        total_sentences = len(doc.sentences)
        total_words = 0
        polysyllabic_words = 0
        polysyllabic_examples = []
        word_syllable_stats = []

        for sentence in doc.sentences:
            for token in sentence.tokens:
                # Рахуємо тільки реальні слова
                if re.match(r'[а-яіїєґa-zA-Z]', token.text):
                    total_words += 1
                    syllables = self.count_syllables_ukrainian(token.text)
                    word_syllable_stats.append({
                        'word': token.text,
                        'syllables': syllables,
                        'is_polysyllabic': syllables >= 3
                    })

                    if self.is_polysyllabic_ukrainian(token.text):
                        polysyllabic_words += 1
                        if len(polysyllabic_examples) < 10:  # Зберігаємо
приклади
                            polysyllabic_examples.append({
                                'word': token.text,
                                'syllables': syllables
                            })

        return {
            'total_sentences': total_sentences,
            'total_words': total_words,
            'polysyllabic_words': polysyllabic_words,
            'polysyllabic_examples': polysyllabic_examples,
            'polysyllabic_percentage': (polysyllabic_words / max(1,
total_words)) * 100,
            'word_stats': word_syllable_stats[:20]  # Перші 20 слів для аналізу
        }
```

```
    def calculate(self, text: str) -> Dict[str, Any]:
        """
        Розрахунок SMOG індексу для української мови
        """
        if not self.nlp:
            raise ValueError("NLP pipeline не ініціалізований")

        # Підраховуємо полісилабічні слова
        poly_stats = self.count_polysyllabic_words(text)

        total_sentences = poly_stats['total_sentences']
        polysyllabic_words = poly_stats['polysyllabic_words']

        if total_sentences == 0:
            return {
                'score': 0,
                'smog_grade': 0,
                'sentences': 0,
                'total_words': 0,
                'polysyllabic_words': 0,
                'polysyllabic_percentage': 0,
                'polysyllabic_examples': [],
                'reading_level': 'Неможливо визначити'
            }

        polysyllabic_density = polysyllabic_words * self.sentence_factor /
total_sentences
        smog_grade = self.base_coefficient * math.sqrt(polysyllabic_density) +
self.base_grade
        smog_grade = max(0, smog_grade)

        return {
            'score': round(smog_grade, 2),  # Основний score для базового класу
            'smog_grade': round(smog_grade, 2),
            'sentences': total_sentences,
            'total_words': poly_stats['total_words'],
            'polysyllabic_words': polysyllabic_words,
            'polysyllabic_percentage':
round(poly_stats['polysyllabic_percentage'], 2),
            'polysyllabic_examples': poly_stats['polysyllabic_examples'],
            'polysyllabic_density': round(polysyllabic_density, 2),
            'word_stats': poly_stats['word_stats']
        }

    def interpret_score(self, score: float) -> str:
        """
        Інтерпретація каліброваної шкали складності.
        """
        if score < 1.5:
            return "Простий рівень"
        elif score < 2.5:
            return "Середній рівень"
        else:
            return "Складний рівень"

    def get_education_level(self, grade: float) -> str:
        """
        Визначення класу складності за каліброваною шкалою.
        """
        if grade < 1.5:
```

```
            return "Клас 1: простий текст"
        elif grade < 2.5:
            return "Клас 2: середній текст"
        else:
            return "Клас 3: складний текст"

    def format_result(self, calculation_result: Dict[str, Any], interpretation:
str) -> Dict[str, Any]:
        """
        Форматування результату з додатковою інтерпретацією
        """
        result = super().format_result(calculation_result, interpretation)

        # Додаємо детальну інтерпретацію
        smog_grade = calculation_result.get('smog_grade', 0)
        result['detailed_interpretation'] = {
            'readability': interpretation,
            'education_level': self.get_education_level(smog_grade),
            'complexity_analysis': self._analyze_complexity(calculation_result)
        }

        return result

    def _analyze_complexity(self, calc_result: Dict[str, Any]) -> str:
        """
        Аналіз складності на основі результатів SMOG
        """
        poly_percentage = calc_result.get('polysyllabic_percentage', 0)
        smog_grade = calc_result.get('smog_grade', 0)

        if poly_percentage < 10:
            complexity = "низька концентрація складних слів"
        elif poly_percentage < 20:
            complexity = "помірна концентрація складних слів"
        elif poly_percentage < 30:
            complexity = "висока концентрація складних слів"
        else:
            complexity = "дуже висока концентрація складних слів"

        if smog_grade < 1.5:
            overall = "Текст належить до простого класу"
        elif smog_grade < 2.5:
            overall = "Текст належить до середнього класу"
        else:
            overall = "Текст належить до складного класу"

        return f"{overall}. {complexity.capitalize()}."
```

alhoritm/nlp/cognitive_accessibility.py

```
import re
import statistics
from pathlib import Path

from utils.syntax_improvement_service import SyntaxImprovementService
from nlp.pipelines import get_stanza_pipeline
from repositories.base import JsonRepository

# Константи для розширеного синтаксичного аналізу
SUBORDINATE_DEP_LABELS = {
```

```
    'advcl', 'acl', 'acl:relcl', 'ccomp', 'xcomp', 'mark'
}
SIMPLE_SUBORDINATE_CONNECTORS = {
    'що', 'щоб', 'якщо', 'хоча', 'оскільки', 'коли', 'доки', 'допоки',
    'аби', 'наче', 'ніби', 'мов', 'немов', 'поки', 'бо', 'адже', 'тому',
    'попри'
}
PHRASE_SUBORDINATE_CONNECTORS = [
    'попри те що', 'тому що', 'після того як', 'перед тим як',
    'завдяки тому що'
]
TARGET_SENTENCE_LENGTH = 25
TARGET_SENTENCE_VARIATION = 12
MAX_TREE_DEPTH = 12

# Ініціалізація Stanza NLP pipeline з розширеним синтаксичним аналізом
_NLP_PROCESSORS = 'tokenize,mwt,pos,lemma,depparse'
nlp = None

SOURCE_DIR = Path(__file__).resolve().parents[1] / "data"
_repo = JsonRepository(SOURCE_DIR)


def load_profanity_group():
    files = ['profanity.json', 'vulg.json', 'obscene.json']
    combined_words = set()

    for filename in files:
        try:
            words = _repo.read_json(f"source/{filename}", default=[])
            for item in words:
                word = item.get('word')
                if word:
                    combined_words.add(word.lower())
        except Exception as e:
            print(f"Помилка завантаження {filename}: {e}")

    return combined_words


def load_obsolete_group():
    files = ['obsolete.json', 'arch.json']
    combined_words = set()

    for filename in files:
        try:
            words = _repo.read_json(f"source/{filename}", default=[])
            for item in words:
                word = item.get('word')
                if word:
                    combined_words.add(word.lower())
        except Exception as e:
            print(f"Помилка завантаження {filename}: {e}")

    return combined_words


def load_complex_group():
    files = ['complex.json']
    combined_words = set()
```

```
    for filename in files:
        try:
            words = _repo.read_json(f"source/{filename}", default=[])
            for item in words:
                word = item.get('word')
                if word:
                    combined_words.add(word.lower())
        except Exception as e:
            print(f"Помилка завантаження {filename}: {e}")

    return combined_words


def load_single_category(filename):
    words = set()
    try:
        data = _repo.read_json(f"source/{filename}", default=[])
        for item in data:
            word = item.get('word')
            if word:
                words.add(word.lower())
    except Exception as e:
        print(f"Помилка завантаження {filename}: {e}")

    return words


def load_dictionaries():
    dictionaries = {}

    try:
        dictionaries['profanity_group'] = load_profanity_group()
        dictionaries['obsolete_group'] = load_obsolete_group()
        dictionaries['complex_group'] = load_complex_group()

        dictionaries['rare'] = load_single_category('rare.json')
        dictionaries['slang'] = load_single_category('slang.json')
        dictionaries['abbr'] = load_single_category('abbr.json')
        dictionaries['terms'] = set()

    except Exception as e:
        print(f"Помилка завантаження словників: {e}")
        dictionaries = {
            'profanity_group': set(),
            'obsolete_group': set(),
            'complex_group': set(),
            'rare': set(),
            'slang': set(),
            'abbr': set(),
            'terms': set()
        }

    return dictionaries


DICTIONARIES = load_dictionaries()
syntax_improvement_service = None
```

```
def _get_nlp():
    global nlp
    if nlp is None:
        nlp = get_stanza_pipeline(_NLP_PROCESSORS)
    return nlp


def get_syntax_improvement_service() -> SyntaxImprovementService | None:
    global syntax_improvement_service
    if syntax_improvement_service is None:
        pipeline = _get_nlp()
        if pipeline is None:
            return None
        syntax_improvement_service = SyntaxImprovementService(pipeline)
    return syntax_improvement_service


def _safe_int(value):
    try:
        return int(str(value).split('-')[0])
    except (ValueError, TypeError, AttributeError):
        return None


def _dependency_tree_depth(sentence):
    children = {}
    for word in sentence.words:
        head = _safe_int(word.head)
        child = _safe_int(word.id)
        if head is None or child is None:
            continue
        children.setdefault(head, []).append(child)

    def _depth(node):
        node_children = children.get(node, [])
        if not node_children:
            return 1
        return 1 + max(_depth(child) for child in node_children)

    if 0 not in children:
        return 1
    return max(_depth(child) for child in children[0])


def _has_simple_svo(sentence):
    words = sentence.words
    verbs = [word for word in words if word.upos in {'VERB', 'AUX'}]
    subjects = [word for word in words if (word.deprel or
'').startswith('nsubj')]
    objects = [
        word for word in words
        if (word.deprel or '').startswith('obj') or (word.deprel or
'').startswith('iobj')
    ]
    for verb in verbs:
        verb_id = _safe_int(verb.id)
        if verb_id is None:
            continue
        subj_positions = [
            _safe_int(word.id)
```

```
            for word in subjects
            if _safe_int(word.head) == _safe_int(verb.id)
        ]
        obj_positions = [
            _safe_int(word.id)
            for word in objects
            if _safe_int(word.head) == _safe_int(verb.id)
        ]
        subj_positions = [pos for pos in subj_positions if pos is not None]
        obj_positions = [pos for pos in obj_positions if pos is not None]
        if not subj_positions or not obj_positions:
            continue
        if min(subj_positions) < verb_id < min(obj_positions):
            return True
    return False


def calculate_lexical_component(text: str, word_count: int) -> dict:
    pipeline = _get_nlp()
    if pipeline is None:
        return {
            'lexical_score': 1.0,
            'factors': {},
            'counts': {k: 0 for k in ['profanity_group', 'obsolete_group',
'complex_group', 'rare', 'slang', 'abbr', 'terms']}
        }
    doc = pipeline(text)

    counters = {
        'profanity_group': 0,
        'obsolete_group': 0,
        'complex_group': 0,
        'rare': 0,
        'slang': 0,
        'abbr': 0,
        'terms': 0
    }

    for sentence in doc.sentences:
        for token in sentence.tokens:
            word = token.text.lower()

            for word_obj in token.words:
                lemma = word_obj.lemma.lower()

                if lemma in DICTIONARIES['profanity_group'] or word in
DICTIONARIES['profanity_group']:
                    counters['profanity_group'] += 1
                elif lemma in DICTIONARIES['obsolete_group'] or word in
DICTIONARIES['obsolete_group']:
                    counters['obsolete_group'] += 1
                elif lemma in DICTIONARIES['complex_group'] or word in
DICTIONARIES['complex_group']:
                    counters['complex_group'] += 1
                elif lemma in DICTIONARIES['rare'] or word in
DICTIONARIES['rare']:
                    counters['rare'] += 1
                elif lemma in DICTIONARIES['slang'] or word in
DICTIONARIES['slang']:
                    counters['slang'] += 1
```

```
                elif lemma in DICTIONARIES['abbr'] or word in
DICTIONARIES['abbr']:
                    counters['abbr'] += 1
                elif (
                    len(word) > 8 and
                    (re.search(r'[a-zA-Z]', word) or
                     any(word.endswith(suffix) for suffix in ['ація', 'ування',
'ність', 'ство', 'ізація']))
                ):
                    counters['terms'] += 1

    normalized = {}
    for key, count in counters.items():
        normalized[key] = min(count / word_count, 1.0) if word_count > 0 else 0

    lexical = ((1 - normalized['profanity_group']) *
               (1 - normalized['obsolete_group']) *
               (1 - normalized['complex_group']) *
               (1 - normalized['rare']) *
               (1 - normalized['slang']) *
               (1 - normalized['abbr']) *
               (1 - normalized['terms']))

    return {
        'lexical_score': round(lexical, 3),
        'factors': {
            'profanity_group_ratio': round(normalized['profanity_group'], 3),
            'obsolete_group_ratio': round(normalized['obsolete_group'], 3),
            'complex_group_ratio': round(normalized['complex_group'], 3),
            'rare_words_ratio': round(normalized['rare'], 3),
            'slang_ratio': round(normalized['slang'], 3),
            'abbr_ratio': round(normalized['abbr'], 3),
            'terms_ratio': round(normalized['terms'], 3)
        },
        'counts': counters
    }


def calculate_syntactic_component(text: str) -> dict:
    pipeline = _get_nlp()
    if pipeline is None:
        return {'syntactic_score': 1.0, 'factors': {}}
    doc = pipeline(text)
    sentences = doc.sentences
    sentence_count = len(sentences)

    if sentence_count == 0:
        return {'syntactic_score': 1.0, 'factors': {}}

    sentence_lengths = [len(sentence.tokens) for sentence in sentences if
sentence.tokens]
    if not sentence_lengths:
        return {'syntactic_score': 1.0, 'factors': {}}

    avg_sentence_length = sum(sentence_lengths) / len(sentence_lengths)
    length_stdev = statistics.pstdev(sentence_lengths) if len(sentence_lengths)
> 1 else 0.0

    length_penalty = (
        0.7 * min(avg_sentence_length / TARGET_SENTENCE_LENGTH, 1.0) +
```

```
        0.3 * min(length_stdev / TARGET_SENTENCE_VARIATION, 1.0)
    )
    sentence_structure_score = max(0.0, 1 - length_penalty)

    subordinate_sentence_count = 0
    subordinate_markers = 0
    connector_hits = 0
    tree_depths = []
    svo_sentences = 0
    total_tokens = 0

    for sentence in sentences:
        words = sentence.words
        if not words:
            continue
        total_tokens += len(words)
        sentence_has_subordination = False

        for word in words:
            base_rel = (word.deprel or '').split(':')[0]
            if base_rel in SUBORDINATE_DEP_LABELS:
                subordinate_markers += 1
                sentence_has_subordination = True

        for token in sentence.tokens:
            token_text = token.text.lower()
            if token_text in SIMPLE_SUBORDINATE_CONNECTORS:
                connector_hits += 1
                sentence_has_subordination = True
        sentence_text_lower = sentence.text.lower()
        for phrase in PHRASE_SUBORDINATE_CONNECTORS:
            if phrase in sentence_text_lower:
                connector_hits += 1
                sentence_has_subordination = True

        if sentence_has_subordination:
            subordinate_sentence_count += 1

        tree_depths.append(_dependency_tree_depth(sentence))

        if _has_simple_svo(sentence):
            svo_sentences += 1

    clause_density = subordinate_sentence_count / sentence_count
    marker_density = (subordinate_markers + connector_hits) / max(total_tokens,
1)
    clause_complexity = min(0.6 * clause_density + 0.4 * marker_density, 1.0)
    clause_score = max(0.0, 1 - clause_complexity)

    avg_tree_depth = sum(tree_depths) / len(tree_depths) if tree_depths else 1.0
    tree_depth_score = max(0.0, 1 - min(avg_tree_depth / MAX_TREE_DEPTH, 1.0))

    svo_ratio = svo_sentences / sentence_count

    syntactic = (
        0.35 * sentence_structure_score +
        0.35 * clause_score +
        0.2 * tree_depth_score +
        0.1 * svo_ratio
    )
```

```
    return {
        'syntactic_score': round(syntactic, 3),
        'factors': {
            'avg_sentence_length': round(avg_sentence_length, 1),
            'length_variation': round(length_stdev, 1),
            'sentence_structure_score': round(sentence_structure_score, 3),
            'clause_density': round(clause_density, 3),
            'marker_density': round(marker_density, 3),
            'clause_score': round(clause_score, 3),
            'avg_tree_depth': round(avg_tree_depth, 2),
            'tree_depth_score': round(tree_depth_score, 3),
            'svo_ratio': round(svo_ratio, 3)
        }
    }


def calculate_readability_component(text: str) -> dict:
    pipeline = _get_nlp()
    if pipeline is None:
        return {'readability_score': 1.0, 'factors': {}}
    doc = pipeline(text)
    words = []
    sentences = doc.sentences

    for sentence in sentences:
        for token in sentence.tokens:
            if token.text.isalpha():
                words.append(token.text)

    if not words or not sentences:
        return {'readability_score': 1.0, 'factors': {}}

    avg_word_length = sum(len(word) for word in words) / len(words)
    avg_sentence_length = len(words) / len(sentences)

    word_complexity = min(avg_word_length / 8, 1.0)
    sentence_complexity = min(avg_sentence_length / 20, 1.0)

    readability = (1 - word_complexity) * (1 - sentence_complexity)

    return {
        'readability_score': round(readability, 3),
        'factors': {
            'avg_word_length': round(avg_word_length, 1),
            'avg_sentence_length': round(avg_sentence_length, 1),
            'word_complexity': round(word_complexity, 3),
            'sentence_complexity': round(sentence_complexity, 3)
        }
    }


def calculate_cognitive_accessibility_index(text: str) -> dict:
    pipeline = _get_nlp()
    if pipeline is None:
        return {
            'cai': 1.0,
            'category': 'Відмінна доступність',
            'components': {},
            'error': 'NLP pipeline unavailable'
```

```
        }
    doc = pipeline(text)
    word_count = sum(len(sentence.tokens) for sentence in doc.sentences)

    if word_count == 0:
        return {
            'cai': 1.0,
            'category': 'Відмінна доступність',
            'components': {},
            'error': 'Порожній текст'
        }

    lexical_data = calculate_lexical_component(text, word_count)
    syntactic_data = calculate_syntactic_component(text)
    readability_data = calculate_readability_component(text)

    w1, w2, w3 = 0.4, 0.35, 0.25

    cai = (w1 * lexical_data['lexical_score'] +
           w2 * syntactic_data['syntactic_score'] +
           w3 * readability_data['readability_score'])

    if cai >= 0.8:
        category = 'Відмінна доступність'
    elif cai >= 0.6:
        category = 'Хороша доступність'
    elif cai >= 0.4:
        category = 'Помірна доступність'
    elif cai >= 0.2:
        category = 'Низька доступність'
    else:
        category = 'Дуже складний текст'

    return {
        'cai': round(cai, 3),
        'category': category,
        'components': {
            'lexical': lexical_data,
            'syntactic': syntactic_data,
            'readability': readability_data
        },
        'weights': {
            'lexical_weight': w1,
            'syntactic_weight': w2,
            'readability_weight': w3
        },
        'text_stats': {
            'word_count': word_count,
            'sentence_count': len(doc.sentences)
        }
    }
```

alhoritm/nlp/pipelines.py

```
from __future__ import annotations

import threading
from dataclasses import dataclass, field
from typing import Any, Dict
```

```
import stanza


@dataclass
class _LazyPipelines:
    _cache: Dict[str, Any] = field(default_factory=dict)
    _lock: threading.Lock = field(default_factory=threading.Lock)

    def get(self, key: str, *, processors: str) -> Any:
        with self._lock:
            if key in self._cache:
                return self._cache[key]
        try:
            pipeline = stanza.Pipeline(
                lang="uk",
                processors=processors,
                tokenize_no_ssplit=True,
            )
        except Exception as exc:
            print(f"Failed to initialize stanza pipeline ({processors}): {exc}")
            return None
        if pipeline is None:
            return None
        with self._lock:
            self._cache[key] = pipeline
        return pipeline


_pipelines = _LazyPipelines()


def get_stanza_pipeline(processors: str) -> Any:
    return _pipelines.get(processors, processors=processors)
```

alhoritm/nlp/syntax_metrics.py

```
from nlp.pipelines import get_stanza_pipeline

from algorithms.flesch_kincaid import FleschKincaidUkrainian
from algorithms.smog import SMOGUkrainian
from algorithms.d_level import DLevelUkrainian


_NLP_PROCESSORS = 'tokenize,mwt,pos,lemma'
nlp = None

flesch_kincaid_algorithm = FleschKincaidUkrainian(nlp)
smog_algorithm = SMOGUkrainian(nlp)
d_level_algorithm = DLevelUkrainian(nlp)


def get_nlp():
    global nlp
    if nlp is None:
        nlp = get_stanza_pipeline(_NLP_PROCESSORS)
    return nlp


def _ensure_algorithms_nlp():
    pipeline = get_nlp()
```

```
    if pipeline is None:
        return None
    if flesch_kincaid_algorithm.nlp is None:
        flesch_kincaid_algorithm.nlp = pipeline
    if smog_algorithm.nlp is None:
        smog_algorithm.nlp = pipeline
    if d_level_algorithm.nlp is None:
        d_level_algorithm.nlp = pipeline
    return pipeline


def build_flesch_kincaid_response(text: str, algorithm: FleschKincaidUkrainian)
-> dict:
    _ensure_algorithms_nlp()
    result = algorithm.analyze(text)

    if 'error' in result:
        return {
            "input_text": text,
            "flesch_kincaid": {"error": result['error']}
        }

    flesch_result = {
        "flesch_score": result.get("flesch_score", 0),
        "grade_level": result.get("grade_level", 0),
        "sentences": result.get("sentences", 0),
        "words": result.get("words", 0),
        "syllables": result.get("syllables", 0),
        "avg_sentence_length": result.get("avg_sentence_length", 0),
        "avg_syllables_per_word": result.get("avg_syllables_per_word", 0),
        "interpretation": result.get("detailed_interpretation", {
            "readability": result.get("interpretation", ""),
            "education_level": ""
        })
    }

    return {
        "input_text": text,
        "flesch_kincaid": flesch_result
    }


def build_syntax_stats_response(text: str, algorithm: FleschKincaidUkrainian,
pipeline) -> dict:
    if not text.strip():
        return {
            "input_text": text,
            "stats": {
                "sentences": 0,
                "words": 0,
                "characters": 0,
                "characters_no_spaces": 0,
                "avg_sentence_length": 0,
                "avg_word_length": 0
            }
        }

    if pipeline is None:
        pipeline = _ensure_algorithms_nlp()
    if pipeline is None:
```

```
        return {
            "input_text": text,
            "stats": {
                "sentences": 0,
                "words": 0,
                "characters": 0,
                "characters_no_spaces": 0,
                "avg_sentence_length": 0,
                "avg_word_length": 0
            }
        }
    basic_stats = algorithm.get_basic_stats(text)

    doc = pipeline(text)
    word_lengths = []
    total_syllables = 0

    for sentence in doc.sentences:
        for token in sentence.tokens:
            if token.text.isalpha():
                word_lengths.append(len(token.text))
                total_syllables +=
algorithm.count_syllables_ukrainian(token.text)

    avg_sentence_length = basic_stats['words'] / basic_stats['sentences'] if
basic_stats['sentences'] > 0 else 0
    avg_word_length = sum(word_lengths) / len(word_lengths) if word_lengths else
0
    avg_syllables_per_word = total_syllables / basic_stats['words'] if
basic_stats['words'] > 0 else 0

    return {
        "input_text": text,
        "stats": {
            **basic_stats,
            "avg_sentence_length": round(avg_sentence_length, 2),
            "avg_word_length": round(avg_word_length, 2),
            "syllables_stats": {
                "total_syllables": total_syllables,
                "avg_syllables_per_word": round(avg_syllables_per_word, 2)
            }
        }
    }


def build_smog_response(text: str, algorithm: SMOGUkrainian) -> dict:
    _ensure_algorithms_nlp()
    result = algorithm.analyze(text)

    if 'error' in result:
        return {
            "input_text": text,
            "smog": {"error": result['error']}
        }

    smog_result = {
        "smog_grade": result.get("smog_grade", 0),
        "sentences": result.get("sentences", 0),
        "total_words": result.get("total_words", 0),
        "polysyllabic_words": result.get("polysyllabic_words", 0),
```

```
        "polysyllabic_percentage": result.get("polysyllabic_percentage", 0),
        "polysyllabic_examples": result.get("polysyllabic_examples", []),
        "interpretation": result.get("detailed_interpretation", {
            "readability": result.get("interpretation", ""),
            "education_level": "",
            "complexity_analysis": ""
        })
    }

    return {
        "input_text": text,
        "smog": smog_result
    }


def build_d_level_response(text: str, algorithm: DLevelUkrainian) -> dict:
    _ensure_algorithms_nlp()
    result = algorithm.analyze(text)

    if 'error' in result:
        return {
            "input_text": text,
            "d_level": {"error": result['error']}
        }

    d_level_result = {
        "d_level_score": result.get("d_level_score", 0),
        "difficult_words": result.get("difficult_words", 0),
        "difficult_percentage": result.get("difficult_percentage", 0),
        "avg_sentence_length": result.get("avg_sentence_length", 0),
        "total_words": result.get("total_words", 0),
        "total_sentences": result.get("total_sentences", 0),
        "difficult_examples": result.get("difficult_examples", []),
        "interpretation": result.get("detailed_interpretation", {
            "readability": result.get("interpretation", ""),
            "education_level": "",
            "complexity": "",
            "complexity_analysis": ""
        })
    }

    return {
        "input_text": text,
        "d_level": d_level_result
    }


def build_algorithms_response(flesch_algo, smog_algo, d_level_algo) -> dict:
    algorithms = [
        {
            "name": flesch_algo.name,
            "endpoint": "/flesch-kincaid",
            "description": "Аналіз складності тексту за алгоритмом Flesch-
Kincaid, адаптованим для української мови"
        },
        {
            "name": smog_algo.name,
            "endpoint": "/smog",
            "description": "SMOG (Simple Measure of Gobbledygook) - аналіз
складності на основі полісилабічних слів"
```

```
        },
        {
            "name": d_level_algo.name,
            "endpoint": "/d-level",
            "description": "D-Level (Dale-Chall) - аналіз складності на основі
відсотка складних слів та довжини речень"
        }
    ]

    return {
        "algorithms": algorithms,
        "total": len(algorithms)
    }
```

alhoritm/nlp/terms.py

```
import re
import string
from pathlib import Path
import warnings

from nlp.pipelines import get_stanza_pipeline
from datasets import Dataset
from transformers import AutoModelForTokenClassification, AutoTokenizer,
pipeline

from utils.explanation_markup import (
    find_explanation_spans,
    is_followed_by_explanation,
    overlaps_any,
)
from repositories.base import JsonRepository

# Налаштування
TERM_CUT = 0.55  # Збалансований поріг для точності та повноти
MIN_TERM_LENGTH = 3  # Мінімальна довжина терміна
MAX_TERM_LENGTH = 50  # Максимальна довжина терміна
HIGH_CONFIDENCE_THRESHOLD = 0.85  # Поріг для високодостовірних термінів

# Глобальні змінні
ner_pipe = None

_NLP_PROCESSORS = 'tokenize,mwt,pos,lemma'

# ======================== BLACKLISTS AND FILTERS ========================

# Власні назви, які не є термінами
PROPER_NAMES_BLACKLIST = {
    # Країни та міста
    'україна', 'київ', 'харків', 'львів', 'одеса', 'дніпро', 'запоріжжя',
'кривий ріг',
    'європа', 'америка', 'росія', 'польща', 'німеччина', 'франція', 'англія',
'італія',

    # Імена та прізвища
    'іван', 'петро', 'марія', 'анна', 'олександр', 'володимир', 'наталія',
'сергій',
    'іванов', 'петренко', 'коваленко', 'бондаренко', 'мельник', 'шевченко',

    # Організації (які не є термінами)
```

```
    'microsoft', 'google', 'apple', 'facebook', 'amazon', 'tesla', 'netflix',
    'openai', 'nvidia', 'intel', 'amd', 'ibm', 'oracle', 'adobe',

    # Торгові марки
    'windows', 'linux', 'android', 'ios', 'chrome', 'firefox', 'safari',
    'office', 'photoshop', 'illustrator', 'autocad', 'maya', 'blender'
}

# Слова, які часто помилково визначаються як терміни
COMMON_WORDS_BLACKLIST = {
    # Займенники та прийменники
    'що', 'як', 'де', 'коли', 'чому', 'куди', 'звідки', 'хто', 'чий', 'який',
'яка', 'яке', 'які',
    'до', 'на', 'за', 'під', 'над', 'при', 'від', 'з', 'у', 'в', 'о', 'об',
'про', 'по', 'через',
    'або', 'та', 'і', 'й', 'але', 'проте', 'однак', 'якщо', 'щоб', 'коли',
'тому', 'тоді',
    'ще', 'вже', 'завжди', 'ніколи', 'часто', 'рідко', 'також', 'зокрема',
'особливо',

    # Дієслова загального вжитку
    'бути', 'мати', 'робити', 'йти', 'іти', 'давати', 'брати', 'говорити',
'казати', 'знати',
    'думати', 'хотіти', 'любити', 'жити', 'працювати', 'вчитися', 'грати',
'читати', 'писати',
    'дивитися', 'слухати', 'розуміти', 'вирішувати', 'створювати', 'будувати',
'ламати',

    # Прикметники загального вжитку
    'великий', 'малий', 'гарний', 'поганий', 'добрий', 'поганий', 'старий',
'новий', 'молодий',
    'білий', 'чорний', 'червоний', 'синій', 'зелений', 'жовтий', 'сірий',
'коричневий',
    'важливий', 'цікавий', 'складний', 'простий', 'легкий', 'важкий', 'швидкий',
'повільний',

    # Іменники загального вжитку (не терміни)
    'людина', 'чоловік', 'жінка', 'дитина', 'хлопець', 'дівчина', 'батько',
'мати', 'син', 'дочка',
    'дім', 'будинок', 'кімната', 'вікно', 'двері', 'стіл', 'стілець', 'ліжко',
'шафа',
    'їжа', 'вода', 'хліб', 'молоко', 'м\'ясо', 'овочі', 'фрукти', 'каша', 'суп',
'чай', 'кава',
    'день', 'ніч', 'ранок', 'вечір', 'час', 'рік', 'місяць', 'тиждень',
'година', 'хвилина',
    'робота', 'школа', 'університет', 'лікарня', 'магазин', 'ресторан', 'театр',
'кіно',
    'місто', 'село', 'вулиця', 'дорога', 'автобус', 'машина', 'поїзд', 'літак',

    # Абстрактні поняття (занадто загальні)
    'любов', 'щастя', 'горе', 'радість', 'смуток', 'страх', 'надія', 'мрія',
'думка', 'ідея',
    'життя', 'смерть', 'здоров\'я', 'хвороба', 'успіх', 'невдача', 'проблема',
'рішення',
    'результат', 'процес', 'метод', 'спосіб', 'варіант', 'можливість',
'необхідність',

    # Загальні слова з IT (які не є термінами)
    'комп\'ютер', 'інтернет', 'сайт', 'програма', 'додаток', 'файл', 'папка',
'документ',
```

```
    'текст', 'зображення', 'відео', 'аудіо', 'музика', 'фільм', 'фото',
'картинка',
    'інформація', 'дані', 'контент', 'матеріал', 'зміст', 'формат', 'версія',
'тип',
    'варіант', 'опція', 'функція', 'опис', 'назва', 'заголовок', 'список',
'меню',
    'сторінка', 'вікно', 'кнопка', 'поле', 'форма', 'таблиця', 'рядок',
'стовпець',

    # Дієслова та дієприкметники (часто помилково визначаються як терміни)
    'оформлені', 'оформлений', 'оформлена', 'оформлене', 'оформлено',
'оформлять', 'оформити',
    'містять', 'містить', 'містити', 'містилося', 'містилася', 'містився',
'міститься',
    'виявлені', 'виявлений', 'виявлена', 'виявлене', 'виявлено', 'виявляють',
'виявити',
    'встановлені', 'встановлений', 'встановлена', 'встановлено', 'встановлюють',
'встановити',
    'розроблені', 'розроблений', 'розроблена', 'розроблено', 'розробляють',
'розробити',
    'створені', 'створений', 'створена', 'створено', 'створюють', 'створити',
    'використані', 'використаний', 'використана', 'використано',
'використовують', 'використати',
    'представлені', 'представлений', 'представлена', 'представлено',
'представляють', 'представити',
    'отримані', 'отриманий', 'отримана', 'отримано', 'отримують', 'отримати',
    'проведені', 'проведений', 'проведена', 'проведено', 'проводять',
'проводити',
    'досліджені', 'досліджений', 'досліджена', 'досліджено', 'досліджують',
'досліджувати',

    # Загальні прикметники у науковому стилі
    'наукові', 'науковий', 'наукова', 'наукове', 'технічні', 'технічний',
'технічна', 'технічне',
    'практичні', 'практичний', 'практична', 'практичне', 'теоретичні',
'теоретичний', 'теоретична',
    'основні', 'основний', 'основна', 'основне', 'головні', 'головний',
'головна', 'головне',
    'сучасні', 'сучасний', 'сучасна', 'сучасне', 'нові', 'новий', 'нова',
'нове',
    'різні', 'різний', 'різна', 'різне', 'різноманітні', 'загальні',
'загальний', 'загальна',

    # Загальні іменники у науковому стилі (які не є термінами)
    'дослідження', 'вивчення', 'аналіз', 'синтез', 'порівняння', 'розгляд',
'огляд',
    'підхід', 'метод', 'методи', 'способи', 'спосіб', 'прийом', 'прийоми',
'засоби', 'засіб',
    'результат', 'результати', 'висновок', 'висновки', 'рекомендація',
'рекомендації',
    'пропозиція', 'пропозиції', 'перспектива', 'перспективи', 'напрям',
'напрями',
    'область', 'сфера', 'галузь', 'розділ', 'частина', 'аспект', 'аспекти',
'фактор', 'фактори',

    # Конкретні проблемні слова (які не повинні бути термінами окремо)
    'положення', 'принципи', 'норми', 'правила', 'стандарти', 'вимоги',
'критерії',
    'показники', 'індикатори', 'характеристики', 'параметри', 'елементи',
'компоненти',
```

```
    'складові', 'етапи', 'стадії', 'фази', 'періоди', 'моменти', 'ситуації',
'умови',
    'обставини', 'причини', 'наслідки', 'ефекти', 'впливи', 'зміни',
'тенденції',
    'тернини', 'труднощі', 'проблеми', 'завдання', 'цілі', 'задачі', 'функції',
    'рівень', 'рівні', 'ступінь', 'міра', 'обсяг', 'розмір', 'форма', 'вид',
'види',
    'тип', 'типи', 'клас', 'класи', 'група', 'групи', 'категорія', 'категорії',
    'статус', 'стан', 'позиція', 'роль', 'значення', 'сенс', 'зміст', 'суть',
    'мета', 'ціль', 'призначення', 'завдання', 'задача', 'план', 'схема',
'модель',
    'приклад', 'зразок', 'взірець', 'випадок', 'случай', 'факт', 'дія',
'діяльність',
    'робота', 'праця', 'діло', 'справа', 'питання', 'тема', 'предмет',
'об\'єкт',
    'суб\'єкт', 'особа', 'особи', 'людина', 'людей', 'індивід', 'персона',
'персони'
}

# Суфікси, що часто зустрічаються в термінах
TERM_SUFFIXES = {
    # Наукові та технічні суфікси
    'ація', 'ізація', 'ування', 'ення', 'ання', 'ість', 'ність', 'ство', 'тво',
    'логія', 'графія', 'скопія', 'метрія', 'терапія', 'патія', 'фобія', 'філія',
    'генез', 'синтез', 'аналіз', 'діагноз', 'прогноз', 'індекс', 'комплекс',

    # Технічні та професійні
    'інг', 'ант', 'ент', 'ор', 'ер', 'іст', 'ist', 'ік', 'ич', 'ач',
}

# Префікси, що часто зустрічаються в термінах
TERM_PREFIXES = {
    # Наукові префікси
    'авто', 'мікро', 'макро', 'мега', 'нано', 'гіпер', 'супер', 'анти', 'проти',
    'псевдо', 'квазі', 'полі', 'моно', 'біо', 'гео', 'аеро', 'гідро', 'термо',
    'електро', 'магніто', 'радіо', 'фото', 'відео', 'аудіо', 'tele', 'мета',

    # Технічні префікси
    'кібер', 'цифро', 'комп', 'інтер', 'транс', 'екстра', 'інфра', 'ультра'
}

# Слова, що часто є частиною термінів
TERM_COMPONENTS = {
    'система', 'модель', 'алгоритм', 'метод', 'технологія', 'процес',
'структура',
    'архітектура', 'платформа', 'фреймворк', 'протокол', 'інтерфейс', 'база',
    'механізм', 'принцип', 'концепція', 'теорія', 'практика', 'підхід',

    # Академічні терміни
    'академічна', 'академічний', 'академічні', 'доброчесність', 'етика',
'мораль',
    'освітня', 'освітній', 'освітні', 'наукова', 'науковий', 'наукові',
    'дослідницька', 'дослідницький', 'дослідницькі',

    # Технічні компоненти
    'мережа', 'мережі', 'сервер', 'сервери', 'клієнт', 'клієнти',
    'додаток', 'додатки', 'програма', 'програми', 'код', 'коди',

    # Документооборот та адміністративні терміни
    'записка', 'службова', 'дозвіл', 'довідка', 'заява', 'звіт', 'протокол',
```

```
    'акт', 'постанова', 'рішення', 'наказ', 'розпорядження', 'інструкція',
    'регламент', 'положення', 'статут', 'договір', 'угода', 'контракт',
    'ліцензія', 'патент', 'сертифікат', 'атестат', 'диплом', 'посвідчення',

    # Географічні та природничі терміни
    'полонина', 'долина', 'ущелина', 'плато', 'височина', 'низовина',
    'басейн', 'водозбір', 'екосистема', 'біосфера', 'заповідник', 'парк'
}


def read_dataset(path):
    data_path = Path(path)
    if data_path.is_absolute():
        base_dir = data_path.parent
        relative = data_path.name
    else:
        base_dir = Path(__file__).resolve().parents[1]
        relative = str(data_path)
    repo = JsonRepository(base_dir)
    data = repo.read_json(relative, default=[])
    return Dataset.from_list(data)


def init_bert_model():
    """Ініціалізація BERT моделі"""
    global ner_pipe

    print("Завантаження оптимізованої моделі...")

    def load_tokenizer(model_id: str):
        # Prefer *Fast* tokenizers to avoid transformers' token-classification
fallback heuristic.
        try:
            return AutoTokenizer.from_pretrained(model_id,
add_prefix_space=True, use_fast=True)
        except Exception as e:
            print(f"⚠️  Fast tokenizer недоступний для {model_id}: {e}")
            warnings.filterwarnings(
                "ignore",
                message=r"Tokenizer does not support real words, using fallback
heuristic",
            )
            return AutoTokenizer.from_pretrained(model_id,
add_prefix_space=True)

    # Спочатку пробуємо завантажити з Hugging Face Hub
    try:
        print("Використовуємо модель з Hugging Face Hub: RomanSavitskyi/ukr-
term-detection")
        tokenizer = load_tokenizer("RomanSavitskyi/ukr-term-detection")
        model =
AutoModelForTokenClassification.from_pretrained("RomanSavitskyi/ukr-term-
detection")
    except Exception as e:
        print(f"Помилка завантаження з HF Hub: {e}")
        # Fallback до локальної моделі
        model_path = "./models/current"
        import os
        if os.path.exists(model_path):
            print(f"Використовуємо локальну модель: {model_path}")
```

```
            tokenizer = load_tokenizer(model_path)
            model = AutoModelForTokenClassification.from_pretrained(model_path)
        else:
            print("Навчена модель не знайдена, використовуємо базову ukr-
roberta-base")
            tokenizer = load_tokenizer("youscan/ukr-roberta-base")
            train_dataset = read_dataset("data/term-bio-extended.json")
            label_list = sorted({label for row in train_dataset for label in
row['tags']})
            label_to_id = {l: i for i, l in enumerate(label_list)}
            id_to_label = {i: l for l, i in label_to_id.items()}
            model = AutoModelForTokenClassification.from_pretrained(
                "youscan/ukr-roberta-base",
                num_labels=len(label_list),
                id2label=id_to_label,
                label2id=label_to_id,
            )
    ner_pipe = pipeline(
        "token-classification",
        model=model,
        tokenizer=tokenizer,
        aggregation_strategy="first",
    )
    print("Модель ukr-roberta-base завантажена успішно")


def get_ner_pipe():
    global ner_pipe
    if ner_pipe is None:
        try:
            init_bert_model()
        except Exception as exc:
            print(f"Failed to initialize term model: {exc}")
            ner_pipe = None
    return ner_pipe


def clean_and_normalize_text(text):
    """Очищення та нормалізація тексту"""
    text = re.sub(r'\s+', ' ', text)
    text = text.strip()
    return text


def is_proper_name(word):
    """Перевіряє, чи є слово власною назвою"""
    word_clean = word.lower().strip(string.punctuation)

    if word_clean in PROPER_NAMES_BLACKLIST:
        return True

    if len(word) > 1 and word[0].isupper():
        if word.isupper() and len(word) <= 6:
            return False
        if any(word_clean.endswith(suffix) for suffix in ['енко', 'ський',
'цький', 'ич', 'ович', 'івна']):
            return True
        if word[1:].islower() and len(word) >= 3:
            return True
```

```
    return False


def is_number_or_code(word):
    """Перевіряє, чи є слово числом або кодом (не термін)"""
    word_clean = word.strip(string.punctuation)

    if word_clean.isdigit():
        return True

    if re.match(r'^\d+[.,]\d+$', word_clean):
        return True

    if len(word_clean) > 0:
        digit_ratio = sum(1 for c in word_clean if c.isdigit()) /
len(word_clean)
        if digit_ratio > 0.7:
            return True

    if len(word_clean) <= 8 and len(word_clean) > 0 and word_clean[0].isdigit():
        digit_count = sum(1 for c in word_clean if c.isdigit())
        if digit_count / len(word_clean) >= 0.5:
            return True

    if len(word_clean) <= 4 and re.match(r'^\d+[а-яіїєґ]{1,2}$',
word_clean.lower()):
        return True

    return False


def is_common_word(word):
    """Перевіряє, чи є слово загальновживаним (не термін)"""
    word_clean = word.lower().strip(string.punctuation)
    return word_clean in COMMON_WORDS_BLACKLIST


def has_term_features(word):
    """Перевіряє, чи має слово ознаки терміна"""
    word_clean = word.lower().strip(string.punctuation)

    for suffix in TERM_SUFFIXES:
        if word_clean.endswith(suffix):
            return True

    for prefix in TERM_PREFIXES:
        if word_clean.startswith(prefix):
            return True

    if word_clean in TERM_COMPONENTS:
        return True

    if '-' in word and len(word_clean) > 6:
        return True

    if re.search(r'[a-zA-Z]', word) and len(word_clean) > 2:
        return True

    if re.search(r'\d', word) and len(word_clean) > 4:
        digit_count = sum(1 for c in word_clean if c.isdigit())
```

```
        if digit_count / len(word_clean) < 0.5:
            return True

    return False


def is_likely_noun(word):
    """Покращена евристична перевірка, чи є слово іменником"""
    word_clean = word.lower().strip(string.punctuation)

    noun_endings = {
        'ич', 'ець', 'ар', 'ор', 'ер', 'ист', 'ант', 'ент',
        'ість', 'ність', 'ка', 'ця', 'ша', 'ща', 'ча', 'ія',
        'ення', 'ання', 'ування', 'ство', 'тво', 'ище', 'ння',
        'ізм', 'ізація', 'ація', 'ція', 'сія', 'тор', 'атор',
        'ник', 'щик', 'льник', 'івник', 'ач', 'яч', 'ець',
    }

    adj_endings = {
        'ий', 'ій', 'ський', 'цький', 'ний', 'тний', 'льний', 'ивний',
        'ові', 'еві', 'ові', 'ому', 'ому', 'им', 'ею', 'ою', 'ими', 'их',
    }

    verb_endings = {
        'ать', 'ить', 'ють', 'ють', 'ти', 'ися', 'ся', 'ується', 'іється',
        'ують', 'іють', 'ємо', 'імо', 'ете', 'іте', 'ує', 'іє', 'ються',
    }

    participle_endings = {
        'ений', 'ений', 'ана', 'ане', 'ані', 'ений', 'ана', 'ані', 'ено',
        'ючий', 'ючий', 'юча', 'юче', 'ючі', 'ачий', 'ача', 'аче', 'ачі',
    }

    if any(word_clean.endswith(adj) for adj in adj_endings):
        return False
    if any(word_clean.endswith(verb) for verb in verb_endings):
        return False
    if any(word_clean.endswith(part) for part in participle_endings):
        return False

    if any(word_clean.endswith(ending) for ending in noun_endings):
        return True

    if has_term_features(word):
        return True

    if '-' in word_clean and len(word_clean) > 6:
        parts = word_clean.split('-')
        if len(parts) == 2 and len(parts[0]) > 2 and len(parts[1]) > 2:
            if any(parts[0].endswith(ending) for ending in noun_endings) or \
               any(parts[1].endswith(ending) for ending in noun_endings):
                return True

    return False


def is_compound_term(word):
    """Перевіряє, чи є слово складеним терміном"""
    if ' ' in word:
        parts = word.split()
```

```
        if len(parts) == 2:
            part1_clean = parts[0].lower().strip(string.punctuation)
            part2_clean = parts[1].lower().strip(string.punctuation)

            if part1_clean in TERM_COMPONENTS or part2_clean in TERM_COMPONENTS:
                return True

            important_combinations = [
                ('службова', 'записка'), ('записка', 'службова'),
                ('академічна', 'доброчесність'), ('наукова', 'етика'),
                ('освітня', 'програма'), ('навчальна', 'програма'),
            ]

            if (part1_clean, part2_clean) in important_combinations:
                return True

    return False


def post_process_bert_results(results, original_text):
    """Покращена постобробка результатів BERT"""
    filtered_terms = []

    for result in results:
        word = result['word'].strip()
        score = result['score']

        if len(word) < MIN_TERM_LENGTH or len(word) > MAX_TERM_LENGTH:
            continue

        word = word.replace('##', '').replace('_', '').strip()

        if not word or word in string.punctuation or len(word) < 2:
            continue

        if is_proper_name(word) or is_common_word(word) or
is_number_or_code(word):
            continue

        if not is_likely_noun(word) and not has_term_features(word) and not
is_compound_term(word) and score < HIGH_CONFIDENCE_THRESHOLD:
            continue

        adaptive_threshold = TERM_CUT

        if has_term_features(word):
            adaptive_threshold -= 0.1

        if len(word) <= 4:
            adaptive_threshold += 0.1

        if len(word) >= 12:
            adaptive_threshold -= 0.05

        if is_compound_term(word):
            adaptive_threshold -= 0.2

        if score >= HIGH_CONFIDENCE_THRESHOLD or (score >= adaptive_threshold
and
                                                  (is_compound_term(word) or
```

```
has_term_features(word))):
            filtered_terms.append({
                'word': word,
                'score': score,
                'entity_group': result['entity_group'],
                'has_term_features': has_term_features(word),
            })

    unique_terms = {}
    for term in filtered_terms:
        word = term['word']
        if word not in unique_terms or term['score'] >
unique_terms[word]['score']:
            unique_terms[word] = term

    return sorted(unique_terms.values(), key=lambda x: x['score'], reverse=True)


def merge_subword_tokens(results):
    """Об'єднує підслова токени в повні терміни"""
    merged_results = []
    current_term = ""
    current_score = 0
    current_entity = ""

    for result in results:
        word = result['word']
        score = result['score']
        entity = result['entity_group']

        if word.startswith('##') or (entity.startswith('I-') and current_term):
            current_term += word.replace('##', '')
            current_score = max(current_score, score)
        else:
            if current_term:
                merged_results.append({
                    'word': current_term,
                    'score': current_score,
                    'entity_group': current_entity,
                })

            current_term = word.replace('_', '')
            current_score = score
            current_entity = entity

    if current_term:
        merged_results.append({
            'word': current_term,
            'score': current_score,
            'entity_group': current_entity,
        })

    return merged_results


def extract_compound_terms(results):
    """Виділяє складені терміни з послідовних B-TERM та I-TERM"""
    compound_terms = []
    i = 0
```

```
    while i < len(results):
        result = results[i]
        word = result['word']
        entity = result['entity_group']
        score = result['score']

        if entity == 'B-TERM':
            compound_word = word
            compound_score = score
            i += 1

            while i < len(results) and results[i]['entity_group'] == 'I-TERM':
                compound_word += ' ' + results[i]['word']
                compound_score = max(compound_score, results[i]['score'])
                i += 1

            if ' ' in compound_word:
                compound_terms.append({
                    'word': compound_word,
                    'score': compound_score,
                    'entity_group': 'COMPOUND-TERM',
                })
            else:
                compound_terms.append({
                    'word': compound_word,
                    'score': compound_score,
                    'entity_group': 'B-TERM',
                })
        else:
            compound_terms.append(result)
            i += 1

    return compound_terms


def add_morphological_info_to_terms(text, terms):
    """Додає морфологічну інформацію до знайдених термінів"""
    results = []
    nlp = get_stanza_pipeline(_NLP_PROCESSORS)
    if nlp is None:
        for term in terms:
            results.append({
                'text': term['word'],
                'lemma': term['word'].lower(),
                'start': -1,
                'end': -1,
                'pos': 'term',
                'category': 'term',
                'upos': 'UNKNOWN',
                'morph_features': {},
                'score': float(term['score']),
                'entity_group': term['entity_group'],
                'has_term_features': term.get('has_term_features', False),
            })
        return results
    doc = nlp(text)

    word_positions = {}
    for sentence in doc.sentences:
        for token in sentence.tokens:
```

```
            if token.text:
                start_pos = token.start_char
                end_pos = token.end_char
                word_text = token.text.lower()

                for word_obj in token.words:
                    lemma = word_obj.lemma.lower() if word_obj.lemma else
word_text
                    upos = word_obj.upos
                    feats = word_obj.feats

                    morph_features = {}
                    if feats:
                        for feat in feats.split('|'):
                            if '=' in feat:
                                key, value = feat.split('=', 1)
                                morph_features[key] = value

                    word_positions[word_text] = {
                        'start': start_pos,
                        'end': end_pos,
                        'lemma': lemma,
                        'upos': upos,
                        'morph_features': morph_features,
                        'original_text': token.text,
                    }

    for term in terms:
        term_word = term['word'].lower()

        morph_info = None
        if term_word in word_positions:
            morph_info = word_positions[term_word]
        else:
            head = term_word.split()[-1] if term_word.split() else ""
            if head and head in word_positions:
                morph_info = word_positions[head]
            else:
                for word_text, info in word_positions.items():
                    if term_word in word_text or word_text in term_word:
                        morph_info = info
                        break

        if morph_info:
            results.append({
                'text': term['word'],
                'lemma': morph_info['lemma'],
                'start': morph_info['start'],
                'end': morph_info['end'],
                'pos': 'term',
                'category': 'term',
                'upos': morph_info['upos'],
                'morph_features': morph_info['morph_features'],
                'score': float(term['score']),
                'entity_group': term['entity_group'],
                'has_term_features': term.get('has_term_features', False),
            })
        else:
            results.append({
                'text': term['word'],
```

```
                'lemma': term['word'].lower(),
                'start': -1,
                'end': -1,
                'pos': 'term',
                'category': 'term',
                'upos': 'UNKNOWN',
                'morph_features': {},
                'score': float(term['score']),
                'entity_group': term['entity_group'],
                'has_term_features': term.get('has_term_features', False),
            })

    return results


def find_context_terms(text, base_terms):
    """Шукає додаткові терміни на основі контексту"""
    context_terms = []
    words = text.split()

    for i, word in enumerate(words):
        word_clean = word.lower().strip(string.punctuation)

        for base_term in base_terms:
            base_word = base_term['word'].lower()

            if i > 0 and base_word in words[i-1].lower():
                if (has_term_features(word) or is_likely_noun(word)) and \
                   not is_common_word(word) and not is_proper_name(word) and \
                   len(word_clean) >= MIN_TERM_LENGTH:
                    context_terms.append({
                        'word': word_clean,
                        'score': 0.5,
                        'entity_group': 'CONTEXT-TERM',
                        'context_base': base_term['word'],
                    })

            if i < len(words) - 1 and base_word in words[i+1].lower():
                if (has_term_features(word) or is_likely_noun(word)) and \
                   not is_common_word(word) and not is_proper_name(word) and \
                   len(word_clean) >= MIN_TERM_LENGTH:
                    context_terms.append({
                        'word': word_clean,
                        'score': 0.5,
                        'entity_group': 'CONTEXT-TERM',
                        'context_base': base_term['word'],
                    })

    return context_terms


def detect_terms_with_morphology(text: str):
    """
    Programmatic API used by batch analysis/simplification.
    Returns detailed term occurrences with morphology (including start/end).
    """
    if not isinstance(text, str) or not text.strip():
        return []

    cleaned_text = clean_and_normalize_text(text)
```

```
    pipe = get_ner_pipe()
    if pipe is None:
        return []
    bert_results = pipe(cleaned_text)
    merged_results = merge_subword_tokens(bert_results)
    compound_results = extract_compound_terms(merged_results)
    filtered_terms = post_process_bert_results(compound_results, cleaned_text)
    context_terms = find_context_terms(cleaned_text, filtered_terms)

    all_terms = filtered_terms + context_terms
    unique_terms = {}
    for term_entry in all_terms:
        word = term_entry['word']
        if word not in unique_terms or term_entry['score'] >
unique_terms[word]['score']:
            unique_terms[word] = term_entry

    final_terms = sorted(unique_terms.values(), key=lambda x: x['score'],
reverse=True)
    terms_with_morphology = add_morphological_info_to_terms(cleaned_text,
final_terms)

    explained_spans = find_explanation_spans(cleaned_text)
    filtered: list = []
    for t in terms_with_morphology:
        start = t.get("start")
        end = t.get("end")
        if isinstance(start, int) and isinstance(end, int) and start >= 0 and
end >= start:
            if overlaps_any(start, end, explained_spans):
                continue
            if is_followed_by_explanation(cleaned_text, end):
                continue
        filtered.append(t)
    return filtered
```

alhoritm/nlp/slang.py

```
import os
import re
import tempfile
from pathlib import Path
from typing import Optional

import joblib
import stanza
from huggingface_hub import hf_hub_download

from config.experiment import DEFAULT_CONFIDENCE_THRESHOLD

try:
    from utils.linguistic_features import LinguisticFeatureExtractor
except ImportError:
    print("Warning: LinguisticFeatureExtractor не знайдено, покращена модель
може не працювати")

slang_pipe = None  # Hugging Face pipeline
nlp = None
_stanza_init_failed = False
```

```
class HuggingFaceSlangPipeline:
    def __init__(self):
        repo_id = "RomanSavitskyi/ukrainian-slang-detection"
        filename = "enhanced_slang_model.pkl"
        print(f"[pid={os.getpid()}] Завантажуємо модель з Hugging Face:
{repo_id}")

        model_path = hf_hub_download(
            repo_id=repo_id,
            filename=filename,
        )
        self.model = joblib.load(model_path)
        print("Модель завантажена успішно!")

    def __call__(self, text: str):
        pred = self.model.predict([text])[0]
        prob = self.model.predict_proba([text])[0]
        slang_prob = prob[1]
        label = "SLANG" if pred == 1 else "NEUTRAL"
        return {"label": label, "score": float(slang_prob)}


def init_slang_model():
    global slang_pipe

    if slang_pipe is not None:
        return

    try:
        print(f"[pid={os.getpid()}] Завантажуємо модель з Hugging Face...")
        slang_pipe = HuggingFaceSlangPipeline()
        print(f"[pid={os.getpid()}] Hugging Face slang модель ініціалізована.")
    except Exception as e:
        print(f"[pid={os.getpid()}] Помилка завантаження моделі з Hugging Face:
{e}")
        slang_pipe = None


def _get_stanza_resources_dir() -> Optional[Path]:
    env_dir = os.environ.get("STANZA_RESOURCES_DIR")
    if env_dir:
        return Path(env_dir)
    repo_root = Path(__file__).resolve().parents[3]
    candidate = repo_root / "stanza_resources"
    if candidate.exists():
        return candidate
    tmp_dir = Path(tempfile.gettempdir()) / "stanza_resources"
    return tmp_dir


def _ensure_stanza_pipeline():
    global nlp, _stanza_init_failed
    if nlp is not None or _stanza_init_failed:
        return nlp
    try:
        resources_dir = _get_stanza_resources_dir()
        if resources_dir is not None:
            resources_dir.mkdir(parents=True, exist_ok=True)
            nlp = stanza.Pipeline(
```

```
                lang="uk",
                processors="tokenize,mwt,pos,lemma",
                tokenize_no_ssplit=True,
                dir=str(resources_dir),
            )
        else:
            nlp = stanza.Pipeline(lang="uk",
processors="tokenize,mwt,pos,lemma", tokenize_no_ssplit=True)
        return nlp
    except Exception as exc:
        print(f"[pid={os.getpid()}] Помилка ініціалізації Stanza: {exc}")
        nlp = None
        _stanza_init_failed = True
        return None


def clean_text(text):
    """Покращене очищення тексту з збереженням важливих символів"""
    text = re.sub(r'\s+', ' ', text).strip()
    return text


def improved_tokenize(text):
    """
    Покращена токенізація для кращого розпізнавання сленгу:
    - Зберігає емотикони та спеціальні символи
    - Розпізнає українські слова з апострофами
    - Обробляє повторювані символи
    - Зберігає числа та змішані alphanumeric токени
    """
    processed_tokens = []

    emoticon_pattern = r'[;:=8][-o\*\']?[)\](\[dDpP/\:\}\{@\|\\]'
    emoji_pattern = r'[\U0001F600-\U0001F64F\U0001F300-\U0001F5FF\U0001F680-
\U0001F6FF\U0001F1E0-\U0001F1FF\U00002600-\U000026FF\U00002700-\U000027BF]'
    special_slang_pattern = r'[#@]\w+'

    special_tokens = []
    for pattern in [emoticon_pattern, emoji_pattern, special_slang_pattern]:
        matches = re.finditer(pattern, text)
        for match in matches:
            special_tokens.append((match.start(), match.end(), match.group()))

    special_tokens.sort()

    current_pos = 0
    all_tokens = []

    for start, end, token in special_tokens:
        if start > current_pos:
            text_part = text[current_pos:start]
            word_tokens = re.findall(
                r"[a-яієїґA-ЯІЄЇҐ]+(?:'[a-яієїґA-ЯІЄЇҐ]+)*|[a-zA-Z]+(?:'[a-zA-
Z]+)*|\w+",
                text_part,
                re.UNICODE,
            )
            all_tokens.extend(word_tokens)

        all_tokens.append(token)
```

```
            current_pos = end

        if current_pos < len(text):
            text_part = text[current_pos:]
            word_tokens = re.findall(
                r"[a-яієїґA-ЯІЄЇҐ]+(?:'[a-яієїґA-ЯІЄЇҐ]+)*|[a-zA-Z]+(?:'[a-zA-
Z]+)*|\w+",
                text_part,
                re.UNICODE,
            )
            all_tokens.extend(word_tokens)

        for token in all_tokens:
            if not token.strip():
                continue

            if re.match(f'^({emoticon_pattern}|{emoji_pattern})$', token):
                processed_tokens.append(token)
                continue

            if re.match(f'^{special_slang_pattern}$', token):
                processed_tokens.append(token.lower())
                continue

            token_lower = token.lower()

            if len(token_lower) >= 2:
                processed_tokens.append(token_lower)

        return processed_tokens


    def split_text_to_tokens(text):
        """Обгортка для зворотної сумісності"""
        return improved_tokenize(text)


    def get_slang_words_detailed(text,
    confidence_threshold=DEFAULT_CONFIDENCE_THRESHOLD):
        """Знаходить сленгові слова з детальною інформацією включно з морфологією"""
        results = []
        pipeline = _ensure_stanza_pipeline()
        if pipeline is None:
            return results
        doc = pipeline(text)

        for sentence in doc.sentences:
            for token in sentence.tokens:
                if not token.text:
                    continue

                word = token.text.lower()
                word_start = token.start_char
                word_end = token.end_char

                is_special = is_special_slang_token(word)

                word_entry = None

                if is_special:
```

```
                if token.words:
                    word_obj = token.words[0]
                    lemma = word_obj.lemma.lower() if word_obj.lemma else word
                    upos = word_obj.upos if word_obj.upos else "SPECIAL"
                    feats = word_obj.feats
                else:
                    lemma = word
                    upos = "SPECIAL"
                    feats = None

                morph_features = {}
                if feats:
                    for feat in feats.split('|'):
                        if '=' in feat:
                            key, value = feat.split('=', 1)
                            morph_features[key] = value

                word_entry = {
                    "text": token.text,
                    "lemma": lemma,
                    "start": word_start,
                    "end": word_end,
                    "pos": "slang",
                    "category": "slang",
                    "upos": upos,
                    "morph_features": morph_features,
                    "type": "special",
                    "score": 1.0
                }
            else:
                if slang_pipe:
                    pred = slang_pipe(word)
                    is_confident_slang = pred["label"] == "SLANG" and
pred["score"] >= confidence_threshold

                    if is_confident_slang:
                        for word_obj in token.words:
                            lemma = word_obj.lemma.lower()
                            upos = word_obj.upos
                            feats = word_obj.feats

                            morph_features = {}
                            if feats:
                                for feat in feats.split('|'):
                                    if '=' in feat:
                                        key, value = feat.split('=', 1)
                                        morph_features[key] = value

                            word_entry = {
                                "text": token.text,
                                "lemma": lemma,
                                "start": word_start,
                                "end": word_end,
                                "pos": "slang",
                                "category": "slang",
                                "upos": upos,
                                "morph_features": morph_features,
                                "type": "word",
                                "score": pred["score"],
                                "original_prediction": pred["label"],
```

```
                                "confident": pred["score"] >=
confidence_threshold,
                            }
                            break

            if word_entry:
                results.append(word_entry)

    return results


def is_special_slang_token(token):
    """
    Визначає чи є токен спеціальним (емотикон, повторювані символи, тощо)
    Такі токени часто використовуються в сленгу
    """
    emoticon_pattern = r'^[;:=8][-o\*\']?[)\](\[dDpP/\:\}\{@\|\\]$'
    if re.match(emoticon_pattern, token):
        return True

    emoji_pattern = r'^[\U0001F600-\U0001F64F\U0001F300-\U0001F5FF\U0001F680-
\U0001F6FF\U0001F1E0-\U0001F1FF\U00002600-\U000026FF\U00002700-\U000027BF]$'
    if re.match(emoji_pattern, token):
        return True

    if re.match(r'^[#@]\w+$', token):
        return True

    if re.search(r'([а-яієїґА-ЯІЄЇҐa-zA-Z])\1{2,}', token):
        return True

    if re.match(r'^[а-яієїґА-ЯІЄЇҐa-zA-Z]+\d+$', token):
        return True

    return False
```

alhoritm/nlp/lexical/complex.py

```
from pathlib import Path

from nlp.pipelines import get_stanza_pipeline
from repositories.base import JsonRepository

_NLP_PROCESSORS = 'tokenize,mwt,pos,lemma'

_BASE_DIR = Path(__file__).resolve().parents[2]
_repo = JsonRepository(_BASE_DIR / "data")
_COMPLEX_PATH = "source/complex.json"

complex_words = _repo.read_json(_COMPLEX_PATH, default=[])


def build_complex_index(complex_words):
    index = {}
    for entry in complex_words:
        for key in ["word"]:
            val = entry.get(key, "")
            if val:
                val = val.lower()
                if val not in index:
```

```
                    index[val] = entry
    return index


complex_index = build_complex_index(complex_words)


def find_complex_words(text: str):
    results = []
    nlp = get_stanza_pipeline(_NLP_PROCESSORS)
    if nlp is None:
        return results
    doc = nlp(text)

    for sentence in doc.sentences:
        for token in sentence.tokens:
            word = token.text.lower()
            word_start = token.start_char
            word_end = token.end_char

            word_entry = None

            for word_obj in token.words:
                lemma = word_obj.lemma.lower()
                upos = word_obj.upos
                feats = word_obj.feats

                entry = complex_index.get(lemma) or complex_index.get(word)
                if entry:
                    morph_features = {}
                    if feats:
                        for feat in feats.split('|'):
                            if '=' in feat:
                                key, value = feat.split('=', 1)
                                morph_features[key] = value

                    word_entry = {
                        "text": token.text,
                        "lemma": lemma,
                        "start": word_start,
                        "end": word_end,
                        "pos": entry.get("pos", upos.lower()),
                        "category": entry.get("category", ""),
                        "upos": upos,
                        "morph_features": morph_features
                    }
                    break

            if word_entry:
                results.append(word_entry)

    return results
```

alhoritm/nlp/lexical/obsolete.py

```
from pathlib import Path

from nlp.pipelines import get_stanza_pipeline
from repositories.base import JsonRepository
```

```
_NLP_PROCESSORS = 'tokenize,mwt,pos,lemma'

_BASE_DIR = Path(__file__).resolve().parents[2]
_repo = JsonRepository(_BASE_DIR / "data")
_OBSOLETE_PATH = "source/arch.json"

obsolete_words = _repo.read_json(_OBSOLETE_PATH, default=[])


def build_obsolete_index(obsolete_words):
    index = {}
    for entry in obsolete_words:
        for key in ["word"]:
            val = entry.get(key, "")
            if val:
                val = val.lower()
                if val not in index:
                    index[val] = entry
    return index


obsolete_index = build_obsolete_index(obsolete_words)


def find_obsolete_words(text: str):
    results = []
    nlp = get_stanza_pipeline(_NLP_PROCESSORS)
    if nlp is None:
        return results
    doc = nlp(text)

    for sentence in doc.sentences:
        for token in sentence.tokens:
            word = token.text.lower()
            word_start = token.start_char
            word_end = token.end_char

            word_entry = None

            for word_obj in token.words:
                lemma = word_obj.lemma.lower()
                upos = word_obj.upos
                feats = word_obj.feats

                entry = obsolete_index.get(lemma) or obsolete_index.get(word)
                if entry:
                    morph_features = {}
                    if feats:
                        for feat in feats.split('|'):
                            if '=' in feat:
                                key, value = feat.split('=', 1)
                                morph_features[key] = value

                    word_entry = {
                        "text": token.text,
                        "lemma": lemma,
                        "start": word_start,
                        "end": word_end,
                        "pos": entry.get("pos", upos.lower()),
                        "category": entry.get("category", ""),
```

```
                        "upos": upos,
                        "morph_features": morph_features
                    }
                    break

            if word_entry:
                results.append(word_entry)

    return results
```

alhoritm/nlp/lexical/profanity.py

```
import re
from difflib import SequenceMatcher
from pathlib import Path

from thefuzz import fuzz

from nlp.pipelines import get_stanza_pipeline
from repositories.base import JsonRepository

_NLP_PROCESSORS = 'tokenize,mwt,pos,lemma'

DATA_DIR = Path(__file__).resolve().parents[2] / "data"
_repo = JsonRepository(DATA_DIR)
PROFANITY_FILE = "source/profanity.json"
VULG_FILE = "source/vulg.json"
OBSCENE_FILE = "source/obscene.json"

PROFANITY_WORDS = set()


def load_profanity_data(dict_files):
    for file_path in dict_files:
        dict_json = _repo.read_json(file_path, default=[])
        if isinstance(dict_json, list):
            for entry in dict_json:
                word = entry.get("word")
                if word:
                    PROFANITY_WORDS.add(word)


load_profanity_data([PROFANITY_FILE, VULG_FILE, OBSCENE_FILE])


def normalize_text(text: str) -> str:
    text = text.lower()
    text = re.sub(r"[,\.\?\!\*\@\#\$%\^&\(\)\-_=+;:\[\]{}]+", "", text)
    text = re.sub(r"\s+", " ", text)
    return text.strip()


def is_similar(word1, word2, threshold=0.8):
    return SequenceMatcher(None, word1, word2).ratio() >= threshold


def is_fuzzy_similar(word1: str, word2: str, threshold: int = 80) -> bool:
    similarity = fuzz.ratio(word1, word2)
    return similarity >= threshold
```

```
def find_profanity_words_detailed(text: str):
    results = []
    nlp = get_stanza_pipeline(_NLP_PROCESSORS)
    if nlp is None:
        return results
    doc = nlp(text)

    for sentence in doc.sentences:
        for token in sentence.tokens:
            if not token.text:
                continue
            word = token.text.lower()
            word_start = token.start_char
            word_end = token.end_char

            word_entry = None

            for word_obj in token.words:
                lemma = word_obj.lemma.lower()
                upos = word_obj.upos
                feats = word_obj.feats

                found_match = False
                if word in PROFANITY_WORDS or lemma in PROFANITY_WORDS:
                    found_match = True
                else:
                    for p_word in PROFANITY_WORDS:
                        if is_fuzzy_similar(word, p_word, threshold=85) or
is_fuzzy_similar(lemma, p_word, threshold=85):
                            found_match = True
                            break

                if found_match:
                    morph_features = {}
                    if feats:
                        for feat in feats.split('|'):
                            if '=' in feat:
                                key, value = feat.split('=', 1)
                                morph_features[key] = value

                    word_entry = {
                        "text": token.text,
                        "lemma": lemma,
                        "start": word_start,
                        "end": word_end,
                        "pos": "profanity",
                        "category": "profanity",
                        "upos": upos,
                        "morph_features": morph_features
                    }
                    break

            if word_entry:
                results.append(word_entry)

    return results
```

alhoritm/nlp/lexical/rare.py

```
from pathlib import Path

from nlp.pipelines import get_stanza_pipeline
from repositories.base import JsonRepository

_NLP_PROCESSORS = 'tokenize,mwt,pos,lemma'

_BASE_DIR = Path(__file__).resolve().parents[2]
_repo = JsonRepository(_BASE_DIR / "data")
_RARE_PATH = "source/rare.json"

rare_words = _repo.read_json(_RARE_PATH, default=[])


def build_rare_index(rare_words):
    index = {}
    for entry in rare_words:
        for key in ["word", "form"]:
            val = entry.get(key, "")
            if val:
                val = val.lower()
                if val not in index:
                    index[val] = entry
    return index


rare_index = build_rare_index(rare_words)


def find_rare_words(text: str):
    results = []
    nlp = get_stanza_pipeline(_NLP_PROCESSORS)
    if nlp is None:
        return results
    doc = nlp(text)

    for sentence in doc.sentences:
        for token in sentence.tokens:
            word = token.text.lower()
            word_start = token.start_char
            word_end = token.end_char

            word_entry = None

            for word_obj in token.words:
                lemma = word_obj.lemma.lower()
                upos = word_obj.upos
                feats = word_obj.feats

                entry = rare_index.get(lemma) or rare_index.get(word)
                if entry:
                    morph_features = {}
                    if feats:
                        for feat in feats.split('|'):
                            if '=' in feat:
                                key, value = feat.split('=', 1)
                                morph_features[key] = value

                    word_entry = {
```

```
                        "text": token.text,
                        "lemma": lemma,
                        "start": word_start,
                        "end": word_end,
                        "pos": entry.get("pos", upos.lower()),
                        "category": entry.get("category", ""),
                        "upos": upos,
                        "morph_features": morph_features
                    }
                    break

            if word_entry:
                results.append(word_entry)

    return results
```

alhoritm/services/abbr_service.py

```
import re
import threading
from pathlib import Path

import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction import DictVectorizer
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

from services.sentences import sentences
from repositories.base import JsonRepository

# ======================== BLACKLIST - Слова, які НІКОЛИ не є абревіатурами
========================

COMMON_WORDS_BLACKLIST = {
    # Прийменники та сполучники
    'до', 'на', 'за', 'під', 'над', 'при', 'від', 'з', 'у', 'в', 'о', 'об',
'про', 'по', 'через', 'між',
    'що', 'як', 'де', 'коли', 'чому', 'куди', 'звідки', 'або', 'та', 'і', 'й',
'але', 'проте', 'однак',

    # Займенники
    'я', 'ти', 'він', 'вона', 'воно', 'ми', 'ви', 'вони', 'той', 'та', 'те',
'ті', 'цей', 'ця', 'це', 'ці',
    'мій', 'твій', 'його', 'її', 'наш', 'ваш', 'їх', 'себе', 'свій', 'своя',
'своє', 'свої',

    # Дієслова-зв'язки та допоміжні
    'є', 'був', 'була', 'було', 'були', 'буду', 'будеш', 'буде', 'будемо',
'будете', 'будуть',
    'мати', 'має', 'маю', 'маєш', 'маємо', 'маєте', 'мають',

    # Часто вживані прикметники та прислівники
    'також', 'тому', 'тоді', 'тут', 'там', 'ось', 'ще', 'вже', 'завжди',
'ніколи', 'часто', 'рідко',
    'дуже', 'більше', 'менше', 'краще', 'гірше', 'добре', 'погано', 'швидко',
'повільно',

    # Кольори (що часто помилково визначаються)
```

```
    'білий', 'чорний', 'червоний', 'синій', 'зелений', 'жовтий', 'сірий',
'коричневий', 'рожевий', 'помаранчевий',

    # Числівники
    'один', 'два', 'три', 'чотири', 'п\'ять', 'шість', 'сім', 'вісім',
'дев\'ять', 'десять',
    'перший', 'другий', 'третій', 'останній', 'багато', 'мало', 'кілька',
'декілька',

    # Часто вживані іменники
    'час', 'рік', 'день', 'ніч', 'ранок', 'вечір', 'люди', 'людина', 'чоловік',
'жінка', 'дитина',
    'дім', 'будинок', 'місто', 'село', 'країна', 'світ', 'земля', 'небо',
'море', 'річка',
    'робота', 'справа', 'справи', 'діло', 'питання', 'проблема', 'рішення',
'відповідь',
    'період', 'момент', 'хвилина', 'година', 'тиждень', 'місяць', 'життя',
'смерть',

    # Власні назви, що не є абревіатурами
    'україна', 'київ', 'харків', 'львів', 'одеса', 'дніпро', 'європа',
'америка', 'росія',
    'черемон', 'черемош', 'дніпро', 'волга', 'дунай',
}

# ======================== PRECISE REGEX PATTERNS FOR UKRAINIAN ABBREVIATIONS
========================

UKRAINIAN_ABBR_PATTERNS = [
    r'^[СсCc]\.$',
    r'^[ММмm]\.$',
    r'^[ВвVv]ул\.$',
    r'^[ПпPp]л\.$',
    r'^[ПпPp]р\.$',
    r'^[ОоOo]бл\.$',
    r'^[РрRr][-\.]?н\.$',
    r'^[СсCc]мт$',
    r'^[КкKk]ор\.$',
    r'^[ПпPp][-\.]?ща?\.$',

    r'^[ДдDd][-\.]?р\.$',
    r'^[ПпPp]роф\.$',
    r'^[АаAa]кад\.$',
    r'^[КкKk]анд\.$',
    r'^[ГгGg][-\.]?н\.$',
    r'^[ПпPp][-\.]?ні\.$',
    r'^[ВвVv]\.\s*[оОoO]\.$',

    r'^(?:ООН|НАТО|ЄС|США|КНУ|НАН|МОЗ|МОН|КМУ|ВР|НАНУ|КНУТШ|КНЕУ|КНУБА)$',

    r'^(?:р|рр|ст)\.$',

    r'^(?:кг|км|см|мм|дм|л|мл|г|т|га|м|км/г|м/с²|м²|км²)\.$',

    r'^(?:ім|імені)\.$',
    r'^(?:тд|тп|та\s+ін|та\s+інш|та\s+п)\.$',

    r'^№\d+[а-яёіїєґ]?$',

    r'^(?:пн|вт|ср|чт|пт|сб|нд)\.$',
```

```
    r'^(?:січ|лют|бер|кві|тра|чер|лип|сер|вер|жов|лис|гру)\.$',
]

COMPILED_PATTERNS = [re.compile(pattern, re.IGNORECASE | re.UNICODE) for pattern
in UKRAINIAN_ABBR_PATTERNS]


def is_abbr_by_regex(word, context=""):
    """Перевіряє, чи є слово абревіатурою за допомогою регулярних виразів"""
    word_clean = word.rstrip('.,!?;:').lower()

    if word_clean in COMMON_WORDS_BLACKLIST:
        return False

    for pattern in COMPILED_PATTERNS:
        if pattern.fullmatch(word):
            return True

    if len(word_clean) <= 3 and word.endswith('.') and word_clean.isalpha():
        common_endings = ['ся', 'сь', 'те', 'ти', 'ту', 'то']
        if not any(word_clean.endswith(ending) for ending in common_endings):
            return True

    if (
        2 <= len(word_clean) <= 5
        and word_clean.isupper()
        and word_clean.isalpha()
        and word_clean not in COMMON_WORDS_BLACKLIST
    ):
        return True

    return False


TOKEN_PATTERN = r'''
   \b\d{1,2}:\d{2}\b |
   \b\d+[.,]\d+\b  |
   \b[\w''`’]+(?:\.[\w''`’]+)+\.? |
   \b[\w''`’]+(?:[/-][\w''`’]+)+\.? |
   \b№\d+[a-яёіїєґ]?\b |
   \b[\w''`’]+\.(?!\w) |
   \b[\w''`’]+
'''


def tokenize_sentence(sentence):
    """Покращена токенізація з урахуванням українських особливостей"""
    return re.findall(TOKEN_PATTERN, sentence, flags=re.VERBOSE | re.UNICODE)


def tokenize_sentence_with_spans(sentence: str):
    """Токенізація з позиціями start/end у вихідному тексті."""
    tokens = []
    for m in re.finditer(TOKEN_PATTERN, sentence, flags=re.VERBOSE |
re.UNICODE):
        token = m.group(0)
        if not token:
            continue
        tokens.append((token, m.start(), m.end()))
```

```
    return tokens


def enhanced_token_features(tokens, i):
    """Покращені фічі для токенів з урахуванням чорного списку"""
    word = tokens[i]

    prev = tokens[i - 1] if i > 0 else ""
    next_ = tokens[i + 1] if i < len(tokens) - 1 else ""
    prev2 = tokens[i - 2] if i > 1 else ""
    next2 = tokens[i + 2] if i < len(tokens) - 2 else ""

    word_clean = word.rstrip('.,!?;:')

    base_features = {
        "word": word.lower(),
        "word_clean": word_clean.lower(),
        "prev": prev.lower(),
        "next": next_.lower(),
        "prev2": prev2.lower(),
        "next2": next2.lower(),

        "is_blacklisted": word_clean.lower() in COMMON_WORDS_BLACKLIST,

        "has_dot": '.' in word,
        "ends_with_dot": word.endswith('.'),
        "starts_with_dot": word.startswith('.'),
        "has_dash": '-' in word,
        "has_slash": '/' in word,
        "has_number": any(c.isdigit() for c in word),
        "has_hash": '#' in word or '№' in word,

        "is_cap": word[:1].isupper() if word else False,
        "is_upper": word_clean.isupper() if word_clean else False,
        "is_lower": word_clean.islower() if word_clean else False,
        "is_title": word_clean.istitle() if word_clean else False,

        "len": len(word),
        "len_clean": len(word_clean),
        "len_range": "short" if len(word_clean) <= 3 else "medium" if
len(word_clean) <= 6 else "long",

        "prefix_1": word[:1].lower() if word else "",
        "prefix_2": word[:2].lower() if len(word) >= 2 else "",
        "prefix_3": word[:3].lower() if len(word) >= 3 else "",
        "suffix_1": word[-1:].lower() if word else "",
        "suffix_2": word[-2:].lower() if len(word) >= 2 else "",
        "suffix_3": word[-3:].lower() if len(word) >= 3 else "",

        "dot_count": word.count('.'),
        "dash_count": word.count('-'),
        "uppercase_count": sum(1 for c in word if c.isupper()),
        "lowercase_count": sum(1 for c in word if c.islower()),
        "digit_count": sum(1 for c in word if c.isdigit()),

        "uppercase_ratio": sum(1 for c in word if c.isupper()) / len(word) if
len(word) > 0 else 0,
        "alpha_ratio": sum(1 for c in word if c.isalpha()) / len(word) if
len(word) > 0 else 0,
```

```
        "known_abbr": word.lower() in known_abbrs_set,
        "known_abbr_clean": word_clean.lower() in known_abbrs_set,

        "prev_is_preposition": prev.lower() in {'у', 'в', 'на', 'до', 'з',
'від', 'за', 'під', 'над', 'при', 'про'},
        "next_is_cap": next_[:1].isupper() if next_ else False,
        "prev_is_cap": prev[:1].isupper() if prev else False,
        "is_first_word": i == 0,
        "is_last_word": i == len(tokens) - 1,

        "regex_match": is_abbr_by_regex(word),

        "position_ratio": i / len(tokens) if len(tokens) > 0 else 0,
        "distance_from_start": i,
        "distance_from_end": len(tokens) - i - 1,

        "has_ukrainian_chars": bool(re.search(r'[іїєґ]', word, re.IGNORECASE)),
        "typical_ua_abbr_pattern": bool(re.search(r'^[а-яіїєґ]{1,3}\.?$', word,
re.IGNORECASE)),

        "short_with_dot": len(word_clean) <= 4 and word.endswith('.'),
        "cap_with_dot": word[:1].isupper() and word.endswith('.'),
        "all_caps_short": word_clean.isupper() and 2 <= len(word_clean) <= 6,
        "has_abbr_pattern": bool(re.search(r'^[А-ЯІЇЄҐ]{2,6}$', word_clean)),

        "is_common_word_pattern": bool(re.search(r'^[а-яіїєґ]{4,}$',
word_clean.lower())),
        "ends_with_common_suffix":
bool(re.search(r'(ся|сь|ти|ту|то|ня|ка|ко|ль|сть)$', word_clean.lower())),
    }

    return base_features


_BASE_DIR = Path(__file__).resolve().parents[1]
_repo = JsonRepository(_BASE_DIR / "data")
_ABBR_PATH = "source/abbr.json"
abbr_dict = _repo.read_json(_ABBR_PATH, default=[])

known_abbrs_set = {
    entry["word"].lower().rstrip('.')
    for entry in abbr_dict
    if entry.get("category") == "abbr"
}

genuine_abbrs = {
    'оон', 'нато', 'єс', 'сша', 'кну', 'нан', 'моз', 'мон', 'кму', 'вр',
    'проф', 'акад', 'канд', 'докт', 'др',
    'вул', 'пл', 'пр', 'обл', 'смт', 'кор', 'рн', 'м', 'с',
    'ім', 'імені', 'тд', 'тп', 'та ін', 'та інш',
    'кг', 'км', 'см', 'мм', 'дм', 'га', 'л', 'мл', 'г', 'т', 'во',
    'год', 'р', 'рр', 'ст', 'пн', 'вт', 'ср', 'чт', 'пт', 'сб', 'нд',
    'січ', 'лют', 'бер', 'кві', 'тра', 'чер', 'лип', 'сер', 'вер', 'жов', 'лис',
'гру'
}
known_abbrs_set.update(genuine_abbrs)

vec = None
lab = None
model = None
```

```
_model_init_lock = threading.Lock()


def _ensure_abbr_model():
    global vec, lab, model
    if vec is not None and lab is not None and model is not None:
        return

    with _model_init_lock:
        if vec is not None and lab is not None and model is not None:
            return

        x_train_rows = []
        y_labels = []

        print("Підготовка тренувальних даних з покращеною фільтрацією...")

        successful_sentences = 0
        total_sentences = len(sentences)

        for sent, tags in sentences:
            tokens = tokenize_sentence(sent)

            if len(tokens) != len(tags):
                if len(tokens) > len(tags):
                    tokens = tokens[:len(tags)]
                else:
                    continue

            if len(tokens) == len(tags):
                successful_sentences += 1
                for i in range(len(tokens)):
                    feats = enhanced_token_features(tokens, i)
                    x_train_rows.append(feats)
                    y_labels.append(tags[i])

        print(f"Успішно оброблено {successful_sentences}/{total_sentences}
речень")
        print(f"Підготовлено {len(x_train_rows)} зразків для тренування")

        if len(x_train_rows) == 0:
            raise RuntimeError("Немає тренувальних даних для моделі
абревіатур.")

        fitted_vec = DictVectorizer(sparse=False)
        x_vec = fitted_vec.fit_transform(x_train_rows)

        normalized_labels = ['O' if label == 'O' else label for label in
y_labels]

        fitted_lab = LabelEncoder()
        y_vec = fitted_lab.fit_transform(normalized_labels)

        print(f"Класи: {fitted_lab.classes_}")
        print(f"Розподіл класів: {np.bincount(y_vec)}")

        X_train, X_test, y_train, y_test = train_test_split(
            x_vec, y_vec, test_size=0.2, random_state=42, stratify=y_vec
        )
```

```
        fitted_model = RandomForestClassifier(
            n_estimators=100,
            max_depth=10,
            min_samples_split=5,
            min_samples_leaf=3,
            max_features='sqrt',
            random_state=42,
            class_weight='balanced'
        )

        print("Тренування моделі...")
        fitted_model.fit(X_train, y_train)

        y_pred = fitted_model.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)

        print(f"Точність моделі: {accuracy:.3f}")
        print("\nДетальний звіт:")
        print(classification_report(y_test, y_pred,
target_names=fitted_lab.classes_))

        feature_names = fitted_vec.get_feature_names_out()
        feature_importance = fitted_model.feature_importances_
        top_features = sorted(zip(feature_names, feature_importance),
                              key=lambda x: x[1], reverse=True)[:15]

        print("\nТоп-15 найважливіших фіч:")
        for feat, importance in top_features:
            print(f"{feat}: {importance:.4f}")

        vec = fitted_vec
        lab = fitted_lab
        model = fitted_model


def hybrid_predict_token_spans(text: str):
    """Гібридний підхід, але повертає токени з позиціями."""
    _ensure_abbr_model()
    token_spans = tokenize_sentence_with_spans(text)
    tokens = [t for (t, _, _) in token_spans]
    results = []

    for i, token in enumerate(tokens):
        if token.rstrip('.,!?;:').lower() in COMMON_WORDS_BLACKLIST:
            results.append((token, 'O'))
            continue

        if is_abbr_by_regex(token):
            results.append((token, 'ABBR'))
        else:
            feats = enhanced_token_features(tokens, i)

            if feats['is_blacklisted']:
                results.append((token, 'O'))
            else:
                feats_vec = vec.transform([feats])
                pred = model.predict(feats_vec)[0]
                label = lab.inverse_transform([pred])[0]
                results.append((token, label))
```

```
    return token_spans, results


def detect_abbr_occurrences(text: str):
    """Return abbreviation occurrences with start/end positions."""
    token_spans, res = hybrid_predict_token_spans(text)
    span_by_index = {i: (s, e) for i, (_, s, e) in enumerate(token_spans)}

    abbr_objects = []
    for i, (token, label) in enumerate(res):
        if label != 'ABBR':
            continue
        start, end = span_by_index.get(i, (-1, -1))
        if start < 0 or end <= start:
            continue
        abbr_objects.append({
            "text": token,
            "lemma": token.rstrip('.'),
            "start": start,
            "end": end,
            "pos": "abbr",
            "category": "abbreviation"
        })
    return abbr_objects


def hybrid_predict_sentence(text):
    """Гібридний підхід з обов'язковою перевіркою чорного списку"""
    _ensure_abbr_model()
    tokens = tokenize_sentence(text)
    results = []

    for i, token in enumerate(tokens):
        if token.rstrip('.,!?;:').lower() in COMMON_WORDS_BLACKLIST:
            results.append((token, 'O'))
            continue

        if is_abbr_by_regex(token):
            results.append((token, 'ABBR'))
        else:
            feats = enhanced_token_features(tokens, i)

            if feats['is_blacklisted']:
                results.append((token, 'O'))
            else:
                feats_vec = vec.transform([feats])
                pred = model.predict(feats_vec)[0]
                label = lab.inverse_transform([pred])[0]
                results.append((token, label))

    return results


def predict_sentence(text):
    """Чистий ML підхід з перевіркою чорного списку"""
    _ensure_abbr_model()
    tokens = tokenize_sentence(text)
    results = []

    for i, token in enumerate(tokens):
```

```
        if token.rstrip('.,!?;:').lower() in COMMON_WORDS_BLACKLIST:
            results.append((token, 'O'))
            continue

        feats = enhanced_token_features(tokens, i)

        if feats['is_blacklisted']:
            results.append((token, 'O'))
        else:
            feats_vec = vec.transform([feats])
            pred = model.predict(feats_vec)[0]
            label = lab.inverse_transform([pred])[0]
            results.append((token, label))

    return results
```

alhoritm/services/cognitive_accessibility_service.py

```
from nlp.cognitive_accessibility import (
    calculate_cognitive_accessibility_index,
    get_syntax_improvement_service,
)
```

alhoritm/services/parse_file_service.py

```
from utils.textNornalize import extract_text_from_file, save_latest_file

ALLOWED_EXTENSIONS = {'pdf', 'txt', 'doc', 'docx'}


def allowed_file(filename: str) -> bool:
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in
ALLOWED_EXTENSIONS


def parse_uploaded_file(file) -> tuple[dict, int]:
    if file is None:
        return {"error": "No selected file"}, 400

    if not file.filename:
        return {"error": "No selected file"}, 400

    if not allowed_file(file.filename):
        return {"error": "Unsupported file type"}, 400

    file_bytes = file.read()
    filepath = save_latest_file(file_bytes, file.filename)
    text = extract_text_from_file(filepath)
    if not text.strip():
        return {"error": "No text extracted"}, 400

    return {"text": text}, 200
```

alhoritm/services/problems_service.py

```
from __future__ import annotations

from dataclasses import dataclass
from typing import Any, Callable, Dict, List, Optional, Tuple

from services.complex_service import find_complex_words
```

```
from services.obsolete_service import find_obsolete_words
from services.profanity_service import find_profanity_words_detailed
from services.rare_service import find_rare_words


@dataclass(frozen=True)
class TextScope:
    start: int
    end: int


def parse_scope(payload: Dict[str, Any], text_length: int) ->
Tuple[Optional[TextScope], Optional[Tuple[Any, int]]]:
    scope = payload.get("scope")
    if scope is None:
        return TextScope(0, text_length), None
    if not isinstance(scope, dict):
        return None, ({"error": "Field 'scope' must be an object like
{start,end}."}, 400)

    start = scope.get("start", 0)
    end = scope.get("end", text_length)
    if not isinstance(start, int) or not isinstance(end, int):
        return None, ({"error": "Fields 'scope.start' and 'scope.end' must be
integers."}, 400)
    if start < 0 or end < 0 or start > end or end > text_length:
        return None, (
            {
                "error": "Invalid scope range.",
                "details": {"start": start, "end": end, "textLength":
text_length},
            },
            400,
        )
    return TextScope(start, end), None


def normalize_entries(
    entries: List[Dict[str, Any]],
    problem_type: str,
    offset: int,
) -> List[Dict[str, Any]]:
    normalized: List[Dict[str, Any]] = []
    for entry in entries:
        if not isinstance(entry, dict):
            continue
        start = entry.get("start")
        end = entry.get("end")
        if not isinstance(start, int) or not isinstance(end, int):
            continue
        normalized.append(
            {
                "type": problem_type,
                "text": entry.get("text"),
                "lemma": entry.get("lemma"),
                "start": start + offset,
                "end": end + offset,
                "upos": entry.get("upos"),
                "pos": entry.get("pos"),
                "category": entry.get("category"),
```

```
                "morph_features": entry.get("morph_features") or {},
                "score": entry.get("score"),
            }
        )
    return normalized


def get_problems_payload(payload: Dict[str, Any]) -> Tuple[Dict[str, Any], int]:
    if not isinstance(payload, dict):
        return {"error": "Request body must be a JSON object."}, 400

    text = payload.get("text")
    if not isinstance(text, str):
        return {"error": "Field 'text' must be a string."}, 400

    scope, err = parse_scope(payload, len(text))
    if err is not None:
        body, status = err
        return body, status
    assert scope is not None

    requested_categories = payload.get("categories")
    if requested_categories is None:
        categories = ["profanity", "complex", "obsolete", "rare"]
    elif isinstance(requested_categories, list) and all(isinstance(x, str) for x
in requested_categories):
        categories = requested_categories
    else:
        return {"error": "Field 'categories' must be an array of strings."}, 400

    registry: Dict[str, Tuple[str, Callable[[str], List[Dict[str, Any]]]]] = {
        "profanity": ("profanity", find_profanity_words_detailed),
        "complex": ("complex", find_complex_words),
        "obsolete": ("obsolete", find_obsolete_words),
        "rare": ("rare", find_rare_words),
    }

    scoped_text = text[scope.start : scope.end]

    results: List[Dict[str, Any]] = []
    skipped: List[str] = []

    for cat in categories:
        item = registry.get(cat)
        if item is None:
            skipped.append(cat)
            continue
        problem_type, fn = item
        try:
            entries = fn(scoped_text) or []
            results.extend(normalize_entries(entries, problem_type=problem_type,
offset=scope.start))
        except Exception:
            skipped.append(cat)

    counts: Dict[str, int] = {}
    for r in results:
        t = r.get("type")
        if isinstance(t, str):
            counts[t] = counts.get(t, 0) + 1
```

```
    return {
        "scope": {"start": scope.start, "end": scope.end},
        "text_length": len(text),
        "problems": results,
        "counts": counts,
        "available_categories": sorted(registry.keys()),
        "skipped_categories": skipped,
    }, 200
```

alhoritm/services/registry.py

```
from __future__ import annotations

import threading
from dataclasses import dataclass, field
from typing import Any, Callable, Dict, Optional


Factory = Callable[[], Any]


@dataclass
class ServiceRegistry:
    _instances: Dict[str, Any] = field(default_factory=dict)
    _factories: Dict[str, Factory] = field(default_factory=dict)
    _lock: threading.Lock = field(default_factory=threading.Lock)

    def register_instance(self, name: str, instance: Any) -> None:
        with self._lock:
            self._instances[name] = instance

    def register_factory(self, name: str, factory: Factory) -> None:
        with self._lock:
            self._factories[name] = factory

    def get(self, name: str) -> Optional[Any]:
        with self._lock:
            if name in self._instances:
                return self._instances[name]
            factory = self._factories.get(name)
        if factory is None:
            return None
        instance = factory()
        with self._lock:
            self._instances[name] = instance
        return instance


_registry = ServiceRegistry()


def get_registry() -> ServiceRegistry:
    return _registry
```

alhoritm/services/replacement_service_layer.py

```
from utils.morphology import apply_morphological_agreement,
get_compatible_replacements
from utils.replacement_service import ReplacementService
from services.registry import get_registry
```

```
def _get_replacement_service() -> ReplacementService:
    registry = get_registry()
    service = registry.get("replacement_service")
    if service is None:
        service = ReplacementService(enable_substring_heuristic=False)
        registry.register_instance("replacement_service", service)
    return service


def replace_word_payload(data: dict) -> dict:
    if not data or 'word' not in data:
        return {
            'error': 'Missing required field: word',
            'found': False
        }

    word = data['word'].strip()
    word_type = data.get('type', 'profanity')
    morph_info = data.get('morph_info', None)
    allow_llm = bool(data.get("allow_llm", False))

    if not word:
        return {
            'error': 'Word cannot be empty',
            'found': False
        }

    lemma = None
    if isinstance(morph_info, dict):
        lemma = morph_info.get("lemma")
    replacement_service = _get_replacement_service()
    replacement_data, group_type =
replacement_service.get_dictionary_replacement(word, word_type, lemma=lemma)

    if not replacement_data and allow_llm:
            replacement_data, group_type =
replacement_service.generate_and_store_llm_replacement(
                word, word_type, context=data.get('context')
            )

    if replacement_data:
        priority_replacements = replacement_data.get('priority_replacements',
[])
        replacements_list = [r['replacement'] for r in
sorted(priority_replacements, key=lambda x: x['priority'])]

        if morph_info and 'upos' in morph_info:
            original_upos = morph_info['upos']
            print(f"🔍 Фільтруємо заміни для {original_upos}:
{replacements_list}")
            compatible_replacements =
get_compatible_replacements(replacements_list, original_upos)
            if compatible_replacements:
                replacements_list = compatible_replacements
                print(f"✅ Сумісні заміни: {replacements_list}")
            else:
                print(f"⚠️  Жодна заміна не сумісна з {original_upos},
пропускаємо заміну")
```

```
                print(f"❌ ЗАМІНУ НЕ ЗНАЙДЕНО: '{word}' (несумісні частини мови)")
                return {
                    'original_word': word,
                    'replacements': [],
                    'best_replacement': word,
                    'found': False,
                    'reason': f'Несумісні частини мови: потребує {original_upos}'
                }

        best_replacement = replacements_list[0] if replacements_list else word

        morphologically_adjusted = best_replacement
        if morph_info and replacements_list:
            try:
                morphologically_adjusted = apply_morphological_agreement(
                    word, morph_info, best_replacement
                )
            except Exception as e:
                print(f"Помилка морфологічного узгодження: {e}")
                morphologically_adjusted = best_replacement

        return {
            'original_word': word,
            'replacements': replacements_list,
            'best_replacement': best_replacement,
            'morphologically_adjusted': morphologically_adjusted,
            'found': True,
            'context': priority_replacements[0].get('context', 'general') if
priority_replacements else 'general'
        }

    print(f"❌ ЗАМІНУ НЕ ЗНАЙДЕНО: '{word}' (тип: {word_type})")
    return {
        'original_word': word,
        'replacements': [],
        'best_replacement': word,
        'found': False
    }


def replace_multiple_payload(data: dict) -> dict:
    if not data or 'words' not in data:
        return {
            'error': 'Missing required field: words',
            'results': []
        }

    words = data['words']
    if not isinstance(words, list):
        return {
            'error': 'Words must be an array',
            'results': []
        }

    results = []
    replaced_count = 0

    for word_item in words:
        if isinstance(word_item, str):
            word = word_item
```

```
            word_type = 'profanity'
        else:
            word = word_item.get('word', '').strip()
            word_type = word_item.get('type', 'profanity')

        if not word:
            continue

        replacement_service = _get_replacement_service()
        replacement_data, group_type =
replacement_service.get_dictionary_replacement(word, word_type)

        if not replacement_data and bool(word_item.get("allow_llm", False)):
            replacement_data, group_type =
replacement_service.generate_and_store_llm_replacement(
                word, word_type
            )

        if replacement_data:
            priority_replacements =
replacement_data.get('priority_replacements', [])
            best_replacement = priority_replacements[0]['replacement'] if
priority_replacements else word
            replaced_count += 1

            results.append({
                'original_word': word,
                'best_replacement': best_replacement,
                'found': True
            })
        else:
            results.append({
                'original_word': word,
                'best_replacement': word,
                'found': False
            })

    return {
        'results': results,
        'total_processed': len(results),
        'total_replaced': replaced_count
    }


def dictionary_stats_payload() -> dict:
    replacement_service = _get_replacement_service()
    all_dictionaries = replacement_service.get_all_dictionaries()
    for group_type in ReplacementService.GROUP_FILES.keys():
        all_dictionaries.setdefault(group_type, {})

    combined_dictionary = {}
    for group_dict in all_dictionaries.values():
        combined_dictionary.update(group_dict)

    categories = {}
    for word, data in combined_dictionary.items():
        for replacement in data.get('priority_replacements', []):
            context = replacement.get('context', 'general')
            categories[context] = categories.get(context, 0) + 1
```

```
    group_stats = {}
    for group_type, group_dict in all_dictionaries.items():
        group_stats[group_type] = len(group_dict)

    return {
        'total_words': len(combined_dictionary),
        'categories': categories,
        'groups': group_stats,
        'version': '4.0',
        'last_updated': '2025-07-29'
    }


def pending_replacements_payload() -> dict:
    replacement_service = _get_replacement_service()
    pending = replacement_service.list_pending_entries()
    return {
        'total': len(pending),
        'pending': pending
    }


def verify_replacement_payload(data: dict) -> tuple[dict, int]:
    word = data.get('word')
    group = data.get('group')
    replacements = data.get('replacements')

    if not word or not group:
        return {'error': 'word and group are required'}, 400

    if replacements is not None and not isinstance(replacements, list):
        return {'error': 'replacements must be a list'}, 400

    replacement_service = _get_replacement_service()
    success = replacement_service.set_entry_verification(word, group, True,
replacements=replacements)
    status = 200 if success else 404
    return {'success': success}, status


def decline_replacement_payload(data: dict) -> tuple[dict, int]:
    word = data.get('word')
    group = data.get('group')

    if not word or not group:
        return {'error': 'word and group are required'}, 400

    replacement_service = _get_replacement_service()
    success = replacement_service.delete_entry(word, group)
    status = 200 if success else 404
    return {'success': success}, status
```

alhoritm/services/sentences.py

```
sentences = [
    ("С. Велика Димерка постраждала від негоди.", ["ABBR", "O", "O", "O", "O",
"O"]),
    ("Р-н працює на п-щі ім. Артема.", ["ABBR", "O", "O", "ABBR", "ABBR", "O"]),
    ("М. Київ — столиця України.", ["ABBR", "O", "O", "O"]),
    ("Він працює у смт Великий Луг.", ["O", "O", "O", "ABBR", "O", "O"]),
```

```
    ("Вул. Хрещатик розташована в центрі.", ["ABBR", "O", "O", "O", "O"]),
    ("Пл. Незалежності є головною площею.", ["ABBR", "O", "O", "O", "O"]),
    ("Пр. Перемоги тягнется через все місто.", ["ABBR", "O", "O", "O", "O",
"O"]),
    ("Обл. лікарня приймає пацієнтів.", ["ABBR", "O", "O", "O"]),
    ("Він живе на вул. Шевченка.", ["O", "O", "O", "ABBR", "O"]),
    ("Кор. будинку №5 потребує ремонту.", ["ABBR", "O", "O", "O", "O"]),

    ("Д-р Іванов провів операцію.", ["ABBR", "O", "O", "O"]),
    ("Проф. Петренко читає лекцію.", ["ABBR", "O", "O", "O"]),
    ("Акад. Богомолець заснував інститут.", ["ABBR", "O", "O", "O"]),
    ("Канд. наук працює в університеті.", ["ABBR", "O", "O", "O", "O"]),
    ("Г-н Сидоренко очолює відділ.", ["ABBR", "O", "O", "O"]),
    ("П-ні Коваленко вручили нагороду.", ["ABBR", "O", "O", "O"]),
    ("Генерал-майор в.о. командира.", ["O", "ABBR", "O"]),

    ("ООН проводить засідання щотижня.", ["ABBR", "O", "O", "O"]),
    ("НАТО розширює свою діяльність.", ["ABBR", "O", "O", "O"]),
    ("ЄС підтримує нові ініціативи.", ["ABBR", "O", "O", "O"]),
    ("США відправили гуманітарну допомогу.", ["ABBR", "O", "O", "O"]),
    ("КНУ ім. Шевченка набирає студентів.", ["ABBR", "ABBR", "O", "O", "O"]),
    ("НАН України проводить дослідження.", ["ABBR", "O", "O", "O"]),
    ("МОЗ оголосило нові правила.", ["ABBR", "O", "O", "O"]),
    ("СБУ розслідує справу.", ["ABBR", "O", "O"]),
    ("ЗСУ захищають кордони.", ["ABBR", "O", "O"]),
    ("ДСНС реагує на виклики.", ["ABBR", "O", "O", "O"]),

    ("У січ. планується реконструкція.", ["O", "ABBR", "O", "O"]),
    ("Роботи завершаться у лют. 2024 р.", ["O", "O", "O", "ABBR", "O", "ABBR"]),
    ("Конференція відбудеться 15 бер.", ["O", "O", "O", "ABBR"]),
    ("У кв. планується відкриття.", ["O", "ABBR", "O", "O"]),
    ("Травневі свята тривають до 10 трав.", ["O", "O", "O", "O", "O", "ABBR"]),
    ("Екзамени розпочнуться у чер.", ["O", "O", "O", "ABBR"]),
    ("Відпустки планують на лип.-серп.", ["O", "O", "O", "ABBR"]),
    ("Навчання розпочнеться у вер.", ["O", "O", "O", "ABBR"]),
    ("Жовтень скорочено як жовт.", ["O", "O", "O", "ABBR"]),
    ("У листоп. очікується похолодання.", ["O", "ABBR", "O", "O"]),
    ("Грудневі свята у груд.", ["O", "O", "O", "ABBR"]),

    ("Відстань становить 15 км.", ["O", "O", "O", "ABBR"]),
    ("Вага дорівнює 2,5 кг.", ["O", "O", "O", "ABBR"]),
    ("Температура 25 град. за Цельсієм.", ["O", "O", "ABBR", "O", "O"]),
    ("Швидкість 60 км/год.", ["O", "O", "ABBR"]),
    ("Площа кімнати 20 кв.м.", ["O", "O", "O", "ABBR"]),
    ("Об'єм бака 100 л.", ["O", "O", "O", "ABBR"]),
    ("Потужність 1500 Вт.", ["O", "O", "ABBR"]),
    ("Напруга 220 В.", ["O", "O", "ABBR"]),
    ("Тиск становить 2 атм.", ["O", "O", "O", "ABBR"]),

    ('ТОВ "Будівник" отримало ліцензію.', ["ABBR", "O", "O", "O"]),
    ("ЗАТ випустило нові акції.", ["ABBR", "O", "O", "O"]),
    ('ПП "Торгівля" веде діяльність.', ["ABBR", "O", "O", "O"]),
    ("ФОП сплачує податки вчасно.", ["ABBR", "O", "O", "O"]),
    ("АТ оголосило про реорганізацію.", ["ABBR", "O", "O", "O"]),
    ("ПАТ збільшило статутний капітал.", ["ABBR", "O", "O", "O"]),
    ("ТзОВ подало документи.", ["ABBR", "O", "O"]),

    ("Він здобув ступінь к.т.н.", ["O", "O", "O", "ABBR"]),
    ("Вона захистила д.філол.н.", ["O", "O", "ABBR"]),
    ("Студент навчається на ф-ті.", ["O", "O", "O", "ABBR"]),
```

```
    ("Каф. математики набирає викладачів.", ["ABBR", "O", "O", "O"]),
    ("Деп-т освіти оголосив конкурс.", ["ABBR", "O", "O", "O"]),
    ("Лаб. проводить дослідження.", ["ABBR", "O", "O"]),
    ("НДР фінансується державою.", ["ABBR", "O", "O"]),

    ("Лікарня №5 ім. Амосова.", ["O", "O", "ABBR", "O"]),
    ("Відділення інтенсивної терапії у т.ч. реанімація.", ["O", "O", "O", "O",
"ABBR", "O"]),
    ("Пацієнт поступив о 14:30 год.", ["O", "O", "O", "O", "ABBR"]),
    ("Лікування триває 2 тиж.", ["O", "O", "O", "ABBR"]),
    ("Прийом ведеться пн.-пт.", ["O", "O", "ABBR"]),
    ("Аналізи здають з 8:00 до 12:00 год.", ["O", "O", "O", "O", "O", "O",
"ABBR"]),

    ("Автобус №15А їде до ж/д вокзалу.", ["O", "O", "O", "O", "ABBR", "O"]),
    ("Потяг відправляється з ж/в о 15:30.", ["O", "O", "O", "ABBR", "O", "O"]),
    ("Тролейбус курсує від а/с до центру.", ["O", "O", "O", "ABBR", "O", "O"]),
    ("Маршрутне таксі зупиняється біля з/п.", ["O", "O", "O", "O", "ABBR"]),

    ("Це тварина як. Вона крута.", ["O", "O", "O", "O", "O", "O"]),
    ("Документи подаються до 10:00 ранку.", ["O", "O", "O", "O", "O"]),
    ("Робочий день триває з 9:00 до 18:00.", ["O", "O", "O", "O", "O", "O",
"O"]),
    ("Засідання відбудеться в ауд. №205.", ["O", "O", "O", "ABBR", "O"]),
    ("Зустріч призначена на поні. 14:00.", ["O", "O", "O", "ABBR", "O"]),
    ("Термін виконання до 31 груд. поточного року.", ["O", "O", "O", "O",
"ABBR", "O", "O"]),
    ("З повагою, директор школи.", ["O", "O", "O", "O"]),
    ("П.С. Забув додати важливу інформацію.", ["ABBR", "O", "O", "O", "O"]),
    ("До речі, т.зв. експерти помилялися.", ["O", "O", "ABBR", "O", "O"]),
    ("Наприклад, т.б. немає достатньо даних.", ["O", "ABBR", "O", "O", "O",
"O"]),
    ("У т.ч. враховуючи всі фактори.", ["O", "ABBR", "O", "O", "O"]),
    ("Всього п. 15 різних варіантів.", ["O", "ABBR", "O", "O", "O"]),

    ("Сьогодні гарна погода на вулиці.", ["O", "O", "O", "O", "O"]),
    ("Він живе в кв. 15.", ["O", "O", "O", "ABBR", "O"]),
    ("Будинок знаходиться в буд. №7.", ["O", "O", "O", "ABBR", "O"]),
    ("Офіс розташований в корп. А.", ["O", "O", "O", "ABBR", "O"]),
    ("Зустріч призначена в каб. 205.", ["O", "O", "O", "ABBR", "O"]),
    ("Лекція відбудеться в ауд. 15.", ["O", "O", "O", "ABBR", "O"]),
    ("Товари складуються в скл. №3.", ["O", "O", "O", "ABBR", "O"]),
    ("Машина стоїть в гар. 12.", ["O", "O", "O", "ABBR", "O"]),
    ("Документи в п. 5 статті.", ["O", "O", "ABBR", "O", "O"]),
    ("Розділ 3, підрозділ пп. 2.1.", ["O", "O", "O", "ABBR", "O"]),
    ("Згідно зі ст. 15 закону.", ["O", "O", "ABBR", "O", "O"]),
    ("Відповідно до ч. 2 статті.", ["O", "O", "ABBR", "O", "O"]),
    ("Порушення абз. 3 пункту.", ["O", "ABBR", "O", "O"]),
    ("Застосовується примітка прим. до статті.", ["O", "O", "ABBR", "O", "O"]),
    ("Додаток дод. містить таблиці.", ["O", "ABBR", "O", "O"]),
    ("Розглянемо рис. 5 на сторінці.", ["O", "ABBR", "O", "O", "O"]),
    ("Таблиця табл. 3 показує дані.", ["O", "ABBR", "O", "O", "O"]),
    ("Формула ф-ла розраховує відсоток.", ["O", "ABBR", "O", "O"]),
    ("Схема сх. демонструє процес.", ["O", "ABBR", "O", "O"]),
    ("Графік гр. відображає динаміку.", ["O", "ABBR", "O", "O"]),
    ("Діаграма діагр. ілюструє зміни.", ["O", "ABBR", "O", "O"]),
    ("Малюнок мал. пояснює структуру.", ["O", "ABBR", "O", "O"]),
    ("Сторінка ст. 25 книги.", ["O", "ABBR", "O", "O"]),
    ("Том т. 2 енциклопедії.", ["O", "ABBR", "O", "O"]),
    ("Частина ч. 1 дослідження.", ["O", "ABBR", "O", "O"]),
```

```
("Глава гл. 5 розповідає про.", ["O", "ABBR", "O", "O", "O"]),
("Параграф пар. 10 містить висновки.", ["O", "ABBR", "O", "O", "O"]),
("Експеримент експ. тривав місяць.", ["O", "ABBR", "O", "O"]),
("Дослідження досл. показало результати.", ["O", "ABBR", "O", "O"]),
("Спостереження спост. велося щодня.", ["O", "ABBR", "O", "O"]),

("Київська обл. має 25 районів.", ["O", "ABBR", "O", "O", "O"]),
("Харківська обл. розвивається.", ["O", "ABBR", "O"]),
("Львівська обл. приваблює туристів.", ["O", "ABBR", "O", "O"]),
("Одеська обл. має вихід до моря.", ["O", "ABBR", "O", "O", "O", "O"]),
("Дніпропетровська обл. промислова.", ["O", "ABBR", "O"]),
("Полтавська обл. сільськогосподарська.", ["O", "ABBR", "O"]),
("Житомирська обл. лісиста.", ["O", "ABBR", "O"]),
("Вінницька обл. родюча.", ["O", "ABBR", "O"]),
("Чернігівська обл. має річки.", ["O", "ABBR", "O", "O"]),
("Сумська обл. прикордонна.", ["O", "ABBR", "O"]),
("Черкаська обл. центральна.", ["O", "ABBR", "O"]),
("Кіровоградська обл. степова.", ["O", "ABBR", "O"]),
("Миколаївська обл. портова.", ["O", "ABBR", "O"]),
("Херсонська обл. південна.", ["O", "ABBR", "O"]),
("Запорізька обл. має ГЕС.", ["O", "ABBR", "O", "O"]),
("Донецька обл. промислова.", ["O", "ABBR", "O"]),
("Луганська обл. прикордонна.", ["O", "ABBR", "O"]),
("Волинська обл. лісиста.", ["O", "ABBR", "O"]),
("Рівненська обл. має озера.", ["O", "ABBR", "O", "O"]),
("Тернопільська обл. горбиста.", ["O", "ABBR", "O"]),
("Хмельницька обл. аграрна.", ["O", "ABBR", "O"]),
("Чернівецька обл. гориста.", ["O", "ABBR", "O"]),
("Івано-Франківська обл. туристична.", ["O", "ABBR", "O"]),
("Львівська обл. культурна.", ["O", "ABBR", "O"]),
("Закарпатська обл. прикордонна.", ["O", "ABBR", "O"]),

("Він мешкає в м. Київ.", ["O", "O", "O", "ABBR", "O"]),
("Працює в м. Харків.", ["O", "O", "ABBR", "O"]),
("Їдемо до м. Львів.", ["O", "O", "ABBR", "O"]),
("Живу в м. Одеса.", ["O", "O", "ABBR", "O"]),
("Навчаюся в м. Дніпро.", ["O", "O", "ABBR", "O"]),
("Відпочиваю в м. Запоріжжя.", ["O", "O", "ABBR", "O"]),
("Працюю в м. Кривий Ріг.", ["O", "O", "ABBR", "O", "O"]),
("Мешкаю в м. Маріуполь.", ["O", "O", "ABBR", "O"]),
("Навчаюся в м. Полтава.", ["O", "O", "ABBR", "O"]),
("Їжджу до м. Чернігів.", ["O", "O", "ABBR", "O"]),
("Працюю в м. Суми.", ["O", "O", "ABBR", "O"]),
("Живу в м. Житомир.", ["O", "O", "ABBR", "O"]),
("Мешкаю в м. Вінниця.", ["O", "O", "ABBR", "O"]),
("Працюю в м. Черкаси.", ["O", "O", "ABBR", "O"]),
("Навчаюся в м. Хмельницький.", ["O", "O", "ABBR", "O"]),
("Живу в м. Рівне.", ["O", "O", "ABBR", "O"]),
("Мешкаю в м. Луцьк.", ["O", "O", "ABBR", "O"]),
("Працюю в м. Тернопіль.", ["O", "O", "ABBR", "O"]),
("Навчаюся в м. Ужгород.", ["O", "O", "ABBR", "O"]),
("Їжджу до м. Чернівці.", ["O", "O", "ABBR", "O"]),
("Працюю в м. Івано-Франківськ.", ["O", "O", "ABBR", "O"]),
("Живу в м. Херсон.", ["O", "O", "ABBR", "O"]),
("Мешкаю в м. Миколаїв.", ["O", "O", "ABBR", "O"]),
("Навчаюся в м. Кропивницький.", ["O", "O", "ABBR", "O"]),

("Батько працює в с. Березівка.", ["O", "O", "O", "ABBR", "O"]),
("Бабуся живе в с. Петрівка.", ["O", "O", "O", "ABBR", "O"]),
("Дідусь мешкає в с. Іванівка.", ["O", "O", "O", "ABBR", "O"]),
```

```
    ("Тітка переїхала до с. Михайлівка.", ["O", "O", "O", "ABBR", "O"]),
    ("Дядько володіє землею в с. Степанівка.", ["O", "O", "O", "O", "ABBR",
"O"]),
    ("Сусіди мешкають в с. Олександрівка.", ["O", "O", "O", "ABBR", "O"]),
    ("Родичі живуть в с. Григорівка.", ["O", "O", "O", "ABBR", "O"]),
    ("Друзі переїхали до с. Павлівка.", ["O", "O", "O", "ABBR", "O"]),
    ("Знайомі купили будинок в с. Василівка.", ["O", "O", "O", "O", "ABBR",
"O"]),
    ("Колеги відпочивають в с. Миколаївка.", ["O", "O", "O", "ABBR", "O"]),

    ("Автобус їде до смт. Буча.", ["O", "O", "O", "ABBR", "O"]),
]
```

alhoritm/services/slang_service.py

```
from nlp.slang import (
    get_slang_words_detailed,
    init_slang_model,
    slang_pipe,
)
```

alhoritm/services/survey_service.py

```
import json
import os
import threading
import uuid
from datetime import datetime, timezone
from typing import Any, Dict, List, Tuple


_storage_lock = threading.Lock()
_data_dir = os.path.join(os.path.dirname(__file__), "..", "data")
_storage_file = os.path.join(_data_dir, "surveys.json")


def _ensure_storage() -> None:
    os.makedirs(_data_dir, exist_ok=True)
    if not os.path.exists(_storage_file):
        with open(_storage_file, "w", encoding="utf-8") as storage:
            json.dump([], storage)


def _load_surveys() -> List[Dict[str, Any]]:
    _ensure_storage()
    with open(_storage_file, "r", encoding="utf-8") as storage:
        try:
            return json.load(storage)
        except json.JSONDecodeError:
            return []


def _save_surveys(entries: List[Dict[str, Any]]) -> None:
    with open(_storage_file, "w", encoding="utf-8") as storage:
        json.dump(entries, storage, ensure_ascii=False, indent=2)


def save_survey_payload(payload: Dict[str, Any]) -> Tuple[Dict[str, Any], int]:
    rating = payload.get("rating")

    if not isinstance(rating, (int, float)) or not (1 <= rating <= 10):
```

```
        return {"success": False, "message": "Rating must be between 1 and
10."}, 400

    reasons = payload.get("reasons", [])
    if reasons is None:
        reasons = []
    if not isinstance(reasons, list):
        return {"success": False, "message": "Reasons must be a list."}, 400

    entry = {
        "id": str(uuid.uuid4()),
        "timestamp": payload.get("date") or
datetime.now(timezone.utc).isoformat(),
        "rating": rating,
        "reasons": reasons,
        "fileName": payload.get("fileName"),
        "page": payload.get("page"),
        "simplificationLevel": payload.get("simplificationLevel"),
    }

    with _storage_lock:
        entries = _load_surveys()
        entries.append(entry)
        _save_surveys(entries)

    return {"success": True, "surveyId": entry["id"]}, 201


def list_surveys_payload(limit: str | None) -> Tuple[Dict[str, Any], int]:
    try:
        limit_value = int(limit) if limit is not None else None
    except ValueError:
        return {"success": False, "message": "Limit must be an integer."}, 400

    with _storage_lock:
        entries = _load_surveys()

    entries = sorted(entries, key=lambda entry: entry.get("timestamp", ""),
reverse=True)

    if limit_value is not None:
        entries = entries[:limit_value]

    return {"success": True, "data": entries}, 200
```

alhoritm/services/syntax_service.py

```
from nlp.syntax_metrics import (
    nlp,
    flesch_kincaid_algorithm,
    smog_algorithm,
    d_level_algorithm,
    get_nlp,
    build_flesch_kincaid_response,
    build_syntax_stats_response,
    build_smog_response,
    build_d_level_response,
    build_algorithms_response,
)
```

alhoritm/services/term_service.py

```
from nlp.terms import (
    HIGH_CONFIDENCE_THRESHOLD,
    TERM_CUT,
    COMMON_WORDS_BLACKLIST,
    PROPER_NAMES_BLACKLIST,
    TERM_COMPONENTS,
    TERM_PREFIXES,
    TERM_SUFFIXES,
    add_morphological_info_to_terms,
    clean_and_normalize_text,
    detect_terms_with_morphology,
    extract_compound_terms,
    find_context_terms,
    get_ner_pipe,
    merge_subword_tokens,
    post_process_bert_results,
    ner_pipe,
)
```

alhoritm/routers/abbr.py

```
from flask import Blueprint, jsonify, request

from services.abbr_service import (
    detect_abbr_occurrences,
    hybrid_predict_sentence,
    hybrid_predict_token_spans,
    predict_sentence,
)
from utils.utilResponse import check_response

abbr = Blueprint("abbr", __name__)
@abbr.route("/abbr-score", methods=["POST"])
def check_term():
    text = check_response(request)
    raw_text = text

    # Використовуємо гібридний підхід з позиціями
    token_spans, res = hybrid_predict_token_spans(raw_text)

    # Виводимо для дебагу
    hybrid_abbrs = [word for word, label in res if label == 'ABBR']

    print(f"Гібридний підхід знайшов: {hybrid_abbrs}")
    print(f"Всі токени: {[(w, l) for w, l in res]}")

    # Формуємо об'єкти для уніфікації з іншими ендпоінтами (всі входження +
точні позиції)
    abbr_objects = detect_abbr_occurrences(raw_text)
    unique_text = set([o.get("text") for o in abbr_objects])

    return jsonify({
        "input_text": raw_text,
        "found_count": len(unique_text),
        "words": abbr_objects,
        "debug": {
            "all_tokens": [{"word": w, "label": l} for w, l in res],
            "tokens_count": len(res),
            "tokens_with_spans": [{"word": t, "start": s, "end": e} for (t, s,
```

```
e) in token_spans],
        }
    }), 200


@abbr.route("/abbr-test", methods=["POST"])
def test_models():
    """Тестовий endpoint для порівняння підходів"""
    text = check_response(request)

    original_res = predict_sentence(text)
    hybrid_res = hybrid_predict_sentence(text)

    return jsonify({
        "input_text": text,
        "original_ml": [{"word": w, "label": l} for w, l in original_res],
        "hybrid": [{"word": w, "label": l} for w, l in hybrid_res],
        "comparison": {
            "ml_abbrs": [w for w, l in original_res if l == 'ABBR'],
            "hybrid_abbrs": [w for w, l in hybrid_res if l == 'ABBR']
        }
    }), 200
```

alhoritm/routers/analyze.py

```
from __future__ import annotations

import hashlib
from collections import OrderedDict
from typing import Any, Dict, List, Optional, Tuple

from flask import Blueprint, jsonify, request

from services.abbr_service import detect_abbr_occurrences
from routers.complex import find_complex_words
from services.cognitive_accessibility_service import
calculate_cognitive_accessibility_index
from routers.obsolete import find_obsolete_words
from routers.profanity import find_profanity_words_detailed
from routers.rare import find_rare_words
from services.slang_service import get_slang_words_detailed
from services.term_service import detect_terms_with_morphology
from utils.errors import ApiError
from utils.schema import parse_json
from schemas.analyze import AnalyzeRequest


analyze = Blueprint("analyze", __name__)

from config.experiment import ANALYSIS_CACHE_MAX, DEFAULT_CONFIDENCE_THRESHOLD

_ANALYSIS_CACHE: "OrderedDict[str, Dict[str, Any]]" = OrderedDict()


def _unique_count_by_text(entries: List[Dict[str, Any]]) -> int:
    unique = set()
    for e in entries:
        if not isinstance(e, dict):
            continue
        text = e.get("text")
```

```
        if text is None:
            continue
        unique.add(text)
    # Note: this preserves legacy behavior (unique by raw text, not by
position).
    return len(unique)


def _parse_scope(payload: Dict[str, Any], text_length: int) -> Tuple[Tuple[int,
int], Optional[Tuple[Any, int]]]:
    scope = payload.get("scope")
    if scope is None:
        return (0, text_length), None
    if not isinstance(scope, dict):
        return (0, 0), ({"error": "Field 'scope' must be an object like
{start,end}."}, 400)
    start = scope.get("start", 0)
    end = scope.get("end", text_length)
    if not isinstance(start, int) or not isinstance(end, int):
        return (0, 0), ({"error": "Fields 'scope.start' and 'scope.end' must be
integers."}, 400)
    if start < 0 or end < 0 or start > end or end > text_length:
        return (0, 0), (
            {"error": "Invalid scope range.", "details": {"start": start, "end":
end, "textLength": text_length}},
            400,
        )
    return (start, end), None


def run_batch_analysis(
    text: str,
    *,
    start: int = 0,
    end: Optional[int] = None,
    include_cai: bool = True,
    confidence_threshold: float = DEFAULT_CONFIDENCE_THRESHOLD,
) -> Dict[str, Any]:
    """
    In-process batch analysis helper:
    - lexical detections with start/end offsets for highlighting
    - CAI (optional)

    Offsets in the result are absolute to the original `text`.
    """
    if end is None:
        end = len(text)
    scoped_text = text[start:end]

    # Cache by scope + include_cai + threshold + scoped hash.
    # Note: scoped_text is what detectors and CAI operate on; caching is safe
and avoids repeated expensive NLP work.
    try:
        threshold_key = round(float(confidence_threshold), 4)
    except Exception:
        threshold_key = DEFAULT_CONFIDENCE_THRESHOLD
    scoped_hash = hashlib.sha1(scoped_text.encode("utf-8")).hexdigest()
    cache_key =
f"{start}:{end}:{int(bool(include_cai))}:{threshold_key}:{scoped_hash}"
    cached = _ANALYSIS_CACHE.get(cache_key)
```

```
    if cached is not None:
        _ANALYSIS_CACHE.move_to_end(cache_key)
        return cached

    # Lexical detections (return occurrences with positions, but keep
found_count as unique-by-text metric).
    profanity = find_profanity_words_detailed(scoped_text) or []
    obsolete = find_obsolete_words(scoped_text) or []
    rare = find_rare_words(scoped_text) or []
    complex_words = find_complex_words(scoped_text) or []
    slang = get_slang_words_detailed(scoped_text, confidence_threshold) or []
    terms = detect_terms_with_morphology(scoped_text) or []
    abbrs = detect_abbr_occurrences(scoped_text) or []

    def shift(entries: List[Dict[str, Any]]) -> List[Dict[str, Any]]:
        shifted = []
        for e in entries:
            if not isinstance(e, dict):
                continue
            s = e.get("start")
            t = e.get("end")
            if isinstance(s, int) and isinstance(t, int) and s >= 0 and t >= 0:
                e2 = dict(e)
                e2["start"] = s + start
                e2["end"] = t + start
                shifted.append(e2)
            else:
                shifted.append(e)
        return shifted

    detection = {
        "profanity": {"found_count": _unique_count_by_text(profanity), "words":
shift(profanity)},
        "obsolete": {"found_count": _unique_count_by_text(obsolete), "words":
shift(obsolete)},
        "rare": {"found_count": _unique_count_by_text(rare), "words":
shift(rare)},
        "complex": {"found_count": _unique_count_by_text(complex_words),
"words": shift(complex_words)},
        "slang": {"found_count": _unique_count_by_text(slang), "words":
shift(slang)},
        "terms": {"found_count": _unique_count_by_text(terms), "words":
shift(terms)},
        "abbr": {"found_count": _unique_count_by_text(abbrs), "words":
shift(abbrs)},
    }

    response: Dict[str, Any] = {"scope": {"start": start, "end": end},
**detection}
    if include_cai:
        response["cai"] = calculate_cognitive_accessibility_index(scoped_text)

    _ANALYSIS_CACHE[cache_key] = response
    _ANALYSIS_CACHE.move_to_end(cache_key)
    if len(_ANALYSIS_CACHE) > ANALYSIS_CACHE_MAX:
        _ANALYSIS_CACHE.popitem(last=False)
    return response


@analyze.route("/analyze", methods=["POST"])
```

```
def analyze_text():
    """
    Batch analysis endpoint (single request instead of many):
    - lexical detections (with start/end for highlighting)
    - CAI (optionally)

    Request body:
      {
        "text": "...",
        "scope": { "start": 0, "end": 123 },          // optional; defaults to
full text
        "include_cai": true                            // optional; default true
      }
    """
    payload = request.get_json(silent=True) or {}
    parsed = parse_json(payload, AnalyzeRequest)
    text = parsed.text

    (start, end), err = _parse_scope(payload, len(text))
    if err is not None:
        body, status = err
        return jsonify(body), status

    include_cai = bool(parsed.include_cai)
    confidence_threshold = parsed.confidence_threshold

    return (
        jsonify(
            run_batch_analysis(
                text,
                start=start,
                end=end,
                include_cai=include_cai,
                confidence_threshold=confidence_threshold,
            )
        ),
        200,
    )
```

alhoritm/routers/cognitive_accessibility.py

```
from flask import Blueprint, jsonify, request

from services.cognitive_accessibility_service import (
    calculate_cognitive_accessibility_index,
    get_syntax_improvement_service,
)
from utils.utilResponse import check_response
from utils.errors import ApiError

cognitive_accessibility = Blueprint("cognitive_accessibility", __name__)


@cognitive_accessibility.route("/cognitive-accessibility-index",
methods=["POST"])
def get_cai():
    text = check_response(request)

    try:
        result = calculate_cognitive_accessibility_index(text)
```

```
        print("=== COGNITIVE ACCESSIBILITY INDEX ===")
        print(f"CAI: {result['cai']} ({result['category']})")
        print(f"Лексичний: {result['components']['lexical']['lexical_score']}")
        print(f"Синтаксичний:
{result['components']['syntactic']['syntactic_score']}")
        print(f"Читабельність:
{result['components']['readability']['readability_score']}")

        return jsonify(result), 200

    except Exception as e:
        if isinstance(e, ApiError):
            raise
        raise ApiError('Помилка розрахунку індексу когнітивної доступності',
status_code=500, details={"reason": str(e)})


@cognitive_accessibility.route("/syntax-improvement", methods=["POST"])
def improve_syntax():
    text = check_response(request)
    if not isinstance(text, str):
        return text
    payload = request.get_json() or {}
    max_iterations = int(payload.get('max_iterations', 3))
    min_improvement = float(payload.get('min_improvement', 0.01))
    try:
        syntax_improvement_service = get_syntax_improvement_service()
        if syntax_improvement_service is None:
            raise RuntimeError("NLP pipeline is not available")
        result = syntax_improvement_service.improve_sentence_structure(
            text,
            max_iterations=max_iterations,
            min_improvement=min_improvement
        )
        return jsonify(result), 200
    except Exception as exc:
        if isinstance(exc, ApiError):
            raise
        raise ApiError('Не вдалося застосувати синтаксичне спрощення',
status_code=500, details={"reason": str(exc)})
```

alhoritm/routers/complex.py

```
from flask import Blueprint, request, jsonify
from utils.utilResponse import check_response
from services.complex_service import find_complex_words


complex = Blueprint("complex", __name__)

@complex.route("/complex-score", methods=["POST"])
def check_complex():
    text = check_response(request)
    raw_text = text.lower()

    found = find_complex_words(text)
    print("=== FOUND COMPLEX WORDS ===")
    for entry in found:
        print(
```

```
            f"{entry.get('text')} (lemma: {entry.get('lemma')}, pos:
{entry.get('pos')}, category: {entry.get('category')})"
        )

    # Унікальні за текстом слова (метрика), але повертаємо всі входження для
точного підсвічування
    unique_text = set()
    for entry in found:
        unique_text.add(entry.get('text'))

    return jsonify({
        "input_text": raw_text,
        "found_count": len(unique_text),
        "words": found,  # повертаємо всі входження з позиціями
    }), 200
```

alhoritm/routers/iterative_simplify.py

```
from __future__ import annotations

import time
from typing import Any, Dict, List, Optional, Tuple

from flask import Blueprint, jsonify, request

from routers.analyze import _parse_scope  # reuse validation
from routers.analyze import run_batch_analysis
from services.cognitive_accessibility_service import
get_syntax_improvement_service
from utils.morphology import get_compatible_replacements
from utils.morphology import apply_morphological_agreement
from utils.term_explainer import explain_term
from utils.errors import ApiError
from utils.schema import parse_json
from schemas.iterative_simplify import IterativeSimplifyRequest
from config.experiment import (
    CURSOR_LEXICAL_WINDOW_MAX_CHARS,
    CURSOR_SYNTAX_WINDOW_MAX_CHARS,
    DEFAULT_CONFIDENCE_THRESHOLD,
    ENABLE_SUBSTRING_HEURISTIC_DEFAULT,
    ITERATIVE_SIMPLIFY_MODE_DEFAULTS,
    ITERATIVE_SIMPLIFY_MODE_LIMITS,
    ITERATIVE_SIMPLIFY_REDETECT_EVERY_BATCHES,
    ITERATIVE_SIMPLIFY_STRICT_POS_DEFAULT,
    SYNTAX_MAX_ITERATIONS_DEFAULT,
    SYNTAX_MIN_IMPROVEMENT_DEFAULT,
)
from utils.replacement_service import ReplacementService


iterative_simplify = Blueprint("iterative_simplify", __name__)
replacement_service =
ReplacementService(enable_substring_heuristic=ENABLE_SUBSTRING_HEURISTIC_DEFAULT
)

ALGORITHM_ID = "iterative_simplify"
ALGORITHM_VERSION = 1

MODE_DEFAULTS = ITERATIVE_SIMPLIFY_MODE_DEFAULTS
MODE_LIMITS = ITERATIVE_SIMPLIFY_MODE_LIMITS
```

```
MODE_REDETECT_EVERY_BATCHES = ITERATIVE_SIMPLIFY_REDETECT_EVERY_BATCHES


TYPE_PRIORITY = [
    "profanity",
    "abbr",
    "slang",
    "terms",
    "obsolete",
    "complex",
    "rare",
]

TYPE_TO_REPLACEMENT = {
    "profanity": "profanity",
    "abbr": "abbr",
    "slang": "slang",
    "terms": "terms",
    "obsolete": "obsolete",
    "complex": "complex",
    "rare": "rare",
}


def _pick_replacement(
    word: str,
    word_type: str,
    morph_info: Optional[Dict[str, Any]],
    allow_llm: bool,
    *,
    strict_pos: bool,
) -> Optional[str]:
    lemma = morph_info.get("lemma") if isinstance(morph_info, dict) else None

    replacement_data, _group_type =
replacement_service.get_dictionary_replacement(word, word_type, lemma=lemma)

    # Optional LLM fallback (must be explicitly enabled by flag/env; default is
off).
    if not replacement_data and allow_llm:
        replacement_data, _group_type =
replacement_service.generate_and_store_llm_replacement(word, word_type)

    if not replacement_data:
        # If no dictionary replacement found, fall back to term explainer for
terms.
        if word_type in {"term", "terms"}:
            explained = explain_term(word, lemma=lemma)
            if explained:
                return explained
        return None

    priority_replacements = replacement_data.get("priority_replacements", []) or
[]
    if not priority_replacements:
        return None

    sorted_repls = sorted(priority_replacements, key=lambda x: x.get("priority",
9999))
    candidates = [r.get("replacement") for r in sorted_repls if isinstance(r,
```

```
dict)]
    candidates = [c for c in candidates if isinstance(c, str) and c.strip()]
    if not candidates:
        return None

    # Strict POS filtering when we know the original UPOS.
    if strict_pos and morph_info and morph_info.get("upos"):
        try:
            compatible = get_compatible_replacements(candidates,
morph_info.get("upos"))
            if compatible:
                candidates = compatible
            else:
                return None
        except Exception:
            pass

    best = candidates[0]

    # For multi-word / punctuated replacements, avoid morphology
transformations.
    if any(ch in best for ch in (" ", "(", "—", "-")):
        return best

    if morph_info and morph_info.get("morph_features"):
        try:
            return apply_morphological_agreement(word, morph_info, best)
        except Exception:
            return best

    return best


def _apply_patches(text: str, patches: List[Dict[str, Any]]) -> Tuple[str,
List[Dict[str, Any]]]:
    # Apply from right to left.
    applied: List[Dict[str, Any]] = []
    out = text
    for p in sorted(patches, key=lambda x: x["start"], reverse=True):
        start = p["start"]
        end = p["end"]
        repl = p["to"]
        if not isinstance(start, int) or not isinstance(end, int) or start < 0
or end < start:
            continue
        if end > len(out):
            continue
        out = out[:start] + repl + out[end:]
        applied.append(p)
    applied.reverse()
    return out, applied


def _parse_confidence_threshold(payload: Dict[str, Any]) -> float:
    confidence_threshold = payload.get("confidence_threshold",
DEFAULT_CONFIDENCE_THRESHOLD)
    try:
        confidence_threshold = float(confidence_threshold)
    except Exception:
        confidence_threshold = DEFAULT_CONFIDENCE_THRESHOLD
```

```
    return confidence_threshold


def _experiment_config_snapshot() -> Dict[str, Any]:
    return {
        "defaults": {
            "confidence_threshold": DEFAULT_CONFIDENCE_THRESHOLD,
            "strict_pos": ITERATIVE_SIMPLIFY_STRICT_POS_DEFAULT,
            "syntax": {
                "max_iterations": SYNTAX_MAX_ITERATIONS_DEFAULT,
                "min_improvement": SYNTAX_MIN_IMPROVEMENT_DEFAULT,
            },
            "enable_substring_heuristic": ENABLE_SUBSTRING_HEURISTIC_DEFAULT,
            "mode": MODE_DEFAULTS,
            "redetect_every_batches": MODE_REDETECT_EVERY_BATCHES,
        },
        "limits": MODE_LIMITS,
        "cursor_window_max_chars": {
            "lexical": CURSOR_LEXICAL_WINDOW_MAX_CHARS,
            "syntax": CURSOR_SYNTAX_WINDOW_MAX_CHARS,
        },
    }


def _cursor_right_sentence_window(text: str, *, cursor: int, max_chars: int) ->
Tuple[int, int]:
    """
    Cursor-mode window helper: returns a right-bounded sentence-ish segment
starting at `cursor`.

    Constraints:
    - Always starts at cursor (never modifies text before cursor).
    - Ends at the first sentence-ending punctuation in ".!?" if it exists within
`max_chars`,
      otherwise ends at `cursor + max_chars` (clamped to text length).
    """
    if not isinstance(cursor, int):
        cursor = 0
    if cursor < 0:
        cursor = 0
    if cursor > len(text):
        cursor = len(text)
    try:
        max_chars = int(max_chars)
    except Exception:
        max_chars = 0
    if max_chars <= 0:
        return cursor, cursor

    right_bound = min(len(text), cursor + max_chars)
    right_slice = text[cursor:right_bound]
    indices = [right_slice.find("."), right_slice.find("!"),
right_slice.find("?")]
    indices = [i for i in indices if i != -1]
    if indices:
        return cursor, cursor + min(indices) + 1
    return cursor, right_bound


def _simplify_iteration(
```

```
    text: str,
    *,
    start: int,
    end: int,
    max_changes: int,
    allow_llm: bool,
    confidence_threshold: float,
) -> Tuple[str, List[Dict[str, Any]]]:
    # Deprecated: lexical iterations are handled in _handle_iterative_simplify
with batched re-detects.
    return text, []


def _detect_occurrences(scope_text: str, confidence_threshold: float) ->
Dict[str, List[Dict[str, Any]]]:
    # Import helpers directly to avoid request-context coupling.
    from services.abbr_service import detect_abbr_occurrences
    from routers.complex import find_complex_words
    from routers.obsolete import find_obsolete_words
    from routers.profanity import find_profanity_words_detailed
    from routers.rare import find_rare_words
    from services.slang_service import get_slang_words_detailed
    from services.term_service import detect_terms_with_morphology

    return {
        "profanity": find_profanity_words_detailed(scope_text) or [],
        "obsolete": find_obsolete_words(scope_text) or [],
        "rare": find_rare_words(scope_text) or [],
        "complex": find_complex_words(scope_text) or [],
        "slang": get_slang_words_detailed(scope_text, confidence_threshold) or
[],
        "terms": detect_terms_with_morphology(scope_text) or [],
        "abbr": detect_abbr_occurrences(scope_text) or [],
    }


def _detect_occurrences_timed(
    scope_text: str,
    confidence_threshold: float,
    *,
    mode: str,
    cursor_abs_start: Optional[int] = None,
) -> Tuple[Dict[str, List[Dict[str, Any]]], Dict[str, float], int]:
    """
    Returns (occurrences_by_type, timings_ms, window_offset).
    For cursor mode we optionally detect only within a small punctuation-bounded
window around cursor for speed.
    window_offset is relative to the provided scope_text (i.e. offsets inside
returned occurrences must be shifted by +window_offset).
    """
    import time as _time

    window_offset = 0
    text_for_detection = scope_text

    if mode == "cursor" and isinstance(cursor_abs_start, int):
        # Cursor mode must work "from cursor onward". Use a bounded window to
keep heavy detectors fast.
        # Note: in cursor mode `scope_text` is usually `text[cursor:...]`, so
cursor is local index 0 here.
```

```
        seg_start, seg_end = _cursor_right_sentence_window(
            scope_text, cursor=0, max_chars=CURSOR_LEXICAL_WINDOW_MAX_CHARS
        )
        window_offset = seg_start
        text_for_detection = scope_text[seg_start:seg_end]

    # Import helpers directly to avoid request-context coupling.
    from services.abbr_service import detect_abbr_occurrences
    from routers.complex import find_complex_words
    from routers.obsolete import find_obsolete_words
    from routers.profanity import find_profanity_words_detailed
    from routers.rare import find_rare_words
    from services.slang_service import get_slang_words_detailed
    from services.term_service import detect_terms_with_morphology

    timings_ms: Dict[str, float] = {}

    def timed(name: str, fn):
        t0 = _time.perf_counter()
        out = fn()
        timings_ms[name] = (_time.perf_counter() - t0) * 1000.0
        return out

    occurrences = {
        "profanity": timed("profanity", lambda:
find_profanity_words_detailed(text_for_detection) or []),
        "obsolete": timed("obsolete", lambda:
find_obsolete_words(text_for_detection) or []),
        "rare": timed("rare", lambda: find_rare_words(text_for_detection) or
[]),
        "complex": timed("complex", lambda:
find_complex_words(text_for_detection) or []),
        "slang": timed("slang", lambda:
get_slang_words_detailed(text_for_detection, confidence_threshold) or []),
        "terms": timed("terms", lambda:
detect_terms_with_morphology(text_for_detection) or []),
        "abbr": timed("abbr", lambda:
detect_abbr_occurrences(text_for_detection) or []),
    }
    timings_ms["total"] = sum(timings_ms.values())
    return occurrences, timings_ms, window_offset


def _build_candidates(
    occurrences_by_type: Dict[str, List[Dict[str, Any]]], *, base_start: int,
scope_len: int
) -> List[Dict[str, Any]]:
    candidates: List[Dict[str, Any]] = []
    for t in TYPE_PRIORITY:
        for occ in occurrences_by_type.get(t, []):
            if not isinstance(occ, dict):
                continue
            s = occ.get("start")
            e = occ.get("end")
            if not isinstance(s, int) or not isinstance(e, int):
                continue
            if s < 0 or e <= s or e > scope_len:
                continue
            candidates.append({"type": t, "start": s + base_start, "end": e +
base_start, "occ": occ})
```

```
    type_rank = {t: i for i, t in enumerate(TYPE_PRIORITY)}
    candidates.sort(key=lambda c: (type_rank.get(c["type"], 9999), c["start"]))
    return candidates


def _handle_iterative_simplify(payload: Dict[str, Any], *, mode: str, start:
int, end: int):
    t0 = time.perf_counter()
    text = payload.get("text")
    if not isinstance(text, str):
        raise ApiError("Field 'text' must be a string.", status_code=400)

    if not isinstance(start, int) or not isinstance(end, int) or start < 0 or
end < 0 or start > end or end > len(text):
        raise ApiError(
            "Invalid scope range.",
            status_code=400,
            details={"start": start, "end": end, "textLength": len(text)},
        )

    defaults = MODE_DEFAULTS.get(mode, MODE_DEFAULTS["custom"])
    limits = MODE_LIMITS.get(mode, MODE_LIMITS["custom"])

    max_changes = payload.get("max_changes", defaults["max_changes"])
    if not isinstance(max_changes, int) or max_changes < 0:
        raise ApiError("Field 'max_changes' must be a non-negative integer.",
status_code=400)
    max_changes = min(max_changes, int(limits["max_changes"]))

    steps = payload.get("steps", defaults["steps"])
    if not isinstance(steps, int) or steps < 1:
        raise ApiError("Field 'steps' must be a positive integer.",
status_code=400)
    steps = min(steps, int(limits["steps"]))

    allow_llm = bool(payload.get("allow_llm", False))
    strict_pos = bool(payload.get("strict_pos",
ITERATIVE_SIMPLIFY_STRICT_POS_DEFAULT))
    include_analysis = bool(payload.get("include_analysis", False))
    include_cai = bool(payload.get("include_cai", True))
    confidence_threshold = _parse_confidence_threshold(payload)
    syntax_cfg = payload.get("syntax") or {}
    redetect_every = int(payload.get("redetect_every_batches") or
MODE_REDETECT_EVERY_BATCHES.get(mode, 1))
    if redetect_every < 1:
        redetect_every = 1

    steps_performed = 0
    iterations: List[Dict[str, Any]] = []
    iteration_timings_ms: List[float] = []
    last_applied: List[Dict[str, Any]] = []
    out_text = text
    out_start = start
    out_end = end

    lexical_initial_counts: Dict[str, int] = {t: 0 for t in TYPE_PRIORITY}
    lexical_applied_counts: Dict[str, int] = {t: 0 for t in TYPE_PRIORITY}
    redetects = 0
    redetect_timings: List[Dict[str, Any]] = []
```

```
    base_text = out_text
    base_start = out_start
    base_end = out_end
    base_shift = 0
    candidate_idx = 0
    candidates: List[Dict[str, Any]] = []

    def redetect() -> None:
        nonlocal base_text, base_start, base_end, base_shift, candidate_idx,
candidates, redetects
        base_text = out_text
        base_start = out_start
        base_end = out_end
        base_shift = 0
        candidate_idx = 0
        scoped = base_text[base_start:base_end]
        try:
            occ, det_ms, window_offset = _detect_occurrences_timed(
                scoped,
                confidence_threshold,
                mode=mode,
                cursor_abs_start=base_start,
            )
        except TypeError:
            try:
                occ, det_ms, window_offset = _detect_occurrences_timed(scoped,
confidence_threshold)
            except TypeError:
                occ, det_ms, window_offset = _detect_occurrences_timed(scoped)
        if window_offset:
            for entries in occ.values():
                for e in entries:
                    if isinstance(e, dict) and isinstance(e.get("start"), int)
and isinstance(e.get("end"), int):
                        e["start"] = e["start"] + window_offset
                        e["end"] = e["end"] + window_offset
        candidates = _build_candidates(occ, base_start=base_start,
scope_len=len(scoped))
        redetects += 1
        redetect_timings.append({"redetect_index": redetects, "timings_ms":
det_ms, "window_offset": window_offset})
        if redetects == 1:
            for t in TYPE_PRIORITY:
                lexical_initial_counts[t] = len(occ.get(t, []) or [])

    redetect()

    batch_i = 0
    while batch_i < steps:
        if batch_i > 0 and batch_i % redetect_every == 0:
            redetect()

        retry_after_redetect = False
        iter_t0 = time.perf_counter()

        while True:
            applied: List[Dict[str, Any]] = []
            applied_in_batch = 0
            batch_delta_total = 0
```

```
            # Select up to max_changes candidates, but apply replacements
sequentially left-to-right,
            # adjusting indices with base_shift (cumulative delta from earlier
replacements since last redetect).
            used_until_base = -1
            used_until_current = -1

            while candidate_idx < len(candidates) and applied_in_batch <
max_changes:
                cand = candidates[candidate_idx]
                candidate_idx += 1

                s_base = cand["start"]
                e_base = cand["end"]
                if s_base < used_until_base:
                    continue

                s = s_base + base_shift
                e = e_base + base_shift
                if s < used_until_current:
                    continue
                if s < 0 or e > len(out_text) or e <= s:
                    continue

                original = out_text[s:e]
                occ = cand["occ"]
                word_type = TYPE_TO_REPLACEMENT.get(cand["type"], cand["type"])
                morph_info = {
                    "lemma": occ.get("lemma"),
                    "upos": occ.get("upos"),
                    "morph_features": occ.get("morph_features") or {},
                }
                repl = _pick_replacement(original, word_type, morph_info,
allow_llm=allow_llm, strict_pos=strict_pos)
                if not repl or repl == original:
                    continue

                out_text = out_text[:s] + repl + out_text[e:]
                delta = len(repl) - (e - s)
                base_shift += delta
                batch_delta_total += delta

                applied_change = {"type": cand["type"], "start": s, "end": e,
"from": original, "to": repl}
                applied.append(applied_change)
                applied_in_batch += 1
                lexical_applied_counts[cand["type"]] =
lexical_applied_counts.get(cand["type"], 0) + 1

                used_until_base = e_base
                used_until_current = s + len(repl)

            # If nothing applied and candidates are exhausted, force one
redetect and retry once.
            if len(applied) == 0 and candidate_idx >= len(candidates) and not
retry_after_redetect:
                redetect()
                retry_after_redetect = True
                continue
```

```
            # Finalize batch
            steps_performed += 1
            last_applied = applied
            iterations.append({"applied_count": len(applied), "applied_changes":
applied})
            iteration_timings_ms.append((time.perf_counter() - iter_t0) *
1000.0)

            # Adjust scope end when text length changes inside scope.
            if batch_delta_total != 0:
                out_end = max(out_start, min(len(out_text), out_end +
batch_delta_total))

            break

        if len(last_applied) == 0:
            break
        batch_i += 1

    total_applied_count = sum(i.get("applied_count", 0) for i in iterations)

    syntax_ms = 0.0
    if (
        (total_applied_count > 0 or mode == "cursor")
        and isinstance(syntax_cfg, dict)
        and syntax_cfg.get("enabled")
        and out_end > out_start
    ):
        max_iterations = int(syntax_cfg.get("max_iterations",
SYNTAX_MAX_ITERATIONS_DEFAULT))
        min_improvement = float(syntax_cfg.get("min_improvement",
SYNTAX_MIN_IMPROVEMENT_DEFAULT))

        if mode == "cursor":
            # Cursor mode is "from cursor onward": never rewrite text before
cursor.
            segment_abs_start, segment_abs_end = _cursor_right_sentence_window(
                out_text, cursor=out_start,
max_chars=CURSOR_SYNTAX_WINDOW_MAX_CHARS
            )
            segment = out_text[segment_abs_start:segment_abs_end]
        else:
            segment_abs_start, segment_abs_end = out_start, out_end
            segment = out_text[segment_abs_start:segment_abs_end]
        syntax_t0 = time.perf_counter()
        try:
            syntax_improvement_service = get_syntax_improvement_service()
            if syntax_improvement_service is None:
                raise RuntimeError("NLP pipeline is not available")
            syntax_result =
syntax_improvement_service.improve_sentence_structure(
                segment, max_iterations=max_iterations,
min_improvement=min_improvement
            )
            improved_segment = syntax_result.get("improved_text")
            if isinstance(improved_segment, str) and improved_segment.strip()
and improved_segment != segment:
                out_text = out_text[:segment_abs_start] + improved_segment +
out_text[segment_abs_end:]
```

```
                if mode == "cursor":
                    out_start = segment_abs_start
                out_end = max(out_start, min(len(out_text), out_end +
(len(improved_segment) - len(segment))))
        except Exception:
            pass
        syntax_ms = (time.perf_counter() - syntax_t0) * 1000.0

    analysis_t0 = time.perf_counter()
    analysis_result = None
    if include_analysis:
        analysis_result = run_batch_analysis(
            out_text,
            start=0,
            end=len(out_text),
            include_cai=include_cai,
            confidence_threshold=confidence_threshold,
        )
    analysis_ms = (time.perf_counter() - analysis_t0) * 1000.0
    total_ms = (time.perf_counter() - t0) * 1000.0

    return (
        jsonify(
            {
                "algorithm_id": ALGORITHM_ID,
                "algorithm_version": ALGORITHM_VERSION,
                "experiment_config": _experiment_config_snapshot(),
                "mode": mode,
                "scope": {"start": out_start, "end": out_end},
                "limits_used": {"max_changes": max_changes, "steps": steps},
                "timings_ms": {
                    "total": total_ms,
                    "analysis": analysis_ms,
                    "syntax": syntax_ms,
                    "iterations": iteration_timings_ms,
                },
                "redetects": redetects,
                "detector_timings_ms": {
                    "per_redetect": redetect_timings,
                    "total": {
                        k: sum((rt.get("timings_ms") or {}).get(k, 0.0) for rt
in redetect_timings)
                        for k in ["total", *TYPE_PRIORITY]
                    },
                },
                "lexical_progress": {
                    "initial_counts": lexical_initial_counts,
                    "applied_counts": lexical_applied_counts,
                    "remaining_estimated": {k: max(0,
lexical_initial_counts.get(k, 0) - lexical_applied_counts.get(k, 0)) for k in
lexical_initial_counts},
                },
                "steps_requested": steps,
                "steps_performed": steps_performed,
                "applied_changes": last_applied,
                "applied_count": len(last_applied),
                "total_applied_count": total_applied_count,
                "iterations": iterations,
                "text": out_text,
                "analysis": analysis_result,
```

```
            }
        ),
        200,
    )


@iterative_simplify.route("/iterative-simplify", methods=["POST"])
def simplify_once():
    """
    One iteration of simplification for a given scope.

    Request:
      {
        "text": "...",
        "scope": { "start": 0, "end": 123 },      // optional; defaults to full
text
        "max_changes": 5,                          // optional; default 5
        "allow_llm": false,                        // optional; default false
        "syntax": { "enabled": true, "max_iterations": 1, "min_improvement":
0.01 } // optional
      }
    """
    payload = request.get_json(silent=True) or {}
    parsed = parse_json(payload, IterativeSimplifyRequest)
    payload = parsed.model_dump()
    text = parsed.text

    (start, end), err = _parse_scope(payload, len(text))
    if err is not None:
        body, status = err
        raise ApiError(body.get("error", "Invalid scope range"),
status_code=status, details=body.get("details"))

    return _handle_iterative_simplify(payload, mode="custom", start=start,
end=end)


@iterative_simplify.route("/iterative-simplify/document", methods=["POST"])
def simplify_document():
    payload = request.get_json(silent=True) or {}
    parsed = parse_json(payload, IterativeSimplifyRequest)
    payload = parsed.model_dump()
    text = parsed.text
    return _handle_iterative_simplify(payload, mode="document", start=0,
end=len(text))


@iterative_simplify.route("/iterative-simplify/page", methods=["POST"])
def simplify_page():
    payload = request.get_json(silent=True) or {}
    parsed = parse_json(payload, IterativeSimplifyRequest)
    payload = parsed.model_dump()
    text = parsed.text
    (start, end), err = _parse_scope(payload, len(text))
    if err is not None:
        body, status = err
        raise ApiError(body.get("error", "Invalid scope range"),
status_code=status, details=body.get("details"))
    return _handle_iterative_simplify(payload, mode="page", start=start,
end=end)
```

```
@iterative_simplify.route("/iterative-simplify/cursor", methods=["POST"])
def simplify_cursor():
    payload = request.get_json(silent=True) or {}
    parsed = parse_json(payload, IterativeSimplifyRequest)
    payload = parsed.model_dump()
    text = parsed.text
    cursor_index = parsed.cursor_index
    if not isinstance(cursor_index, int):
        raise ApiError("Field 'cursor_index' must be an integer.",
status_code=400)
    if cursor_index < 0 or cursor_index > len(text):
        raise ApiError(
            "Invalid cursor_index.",
            status_code=400,
            details={"cursor_index": cursor_index, "textLength": len(text)},
        )
    return _handle_iterative_simplify(payload, mode="cursor",
start=cursor_index, end=len(text))
```

alhoritm/routers/obsolete.py

```
from flask import Blueprint, request, jsonify
from utils.utilResponse import check_response
from services.obsolete_service import find_obsolete_words


obsolete = Blueprint("obsolete", __name__)

@obsolete.route("/obsolete-score", methods=["POST"])
def check_obsolete():
    text = check_response(request)
    raw_text = text.lower()

    found = find_obsolete_words(text)
    print("=== FOUND obsolete WORDS ===")
    for entry in found:
        print(
            f"{entry.get('text')} (lemma: {entry.get('lemma')}, pos:
{entry.get('pos')}, category: {entry.get('category')})"
        )
    # Унікальні за текстом слова (метрика), але повертаємо всі входження для
точного підсвічування
    unique_text = set()
    for entry in found:
        unique_text.add(entry.get('text'))

    return jsonify({
        "input_text": raw_text,
        "found_count": len(unique_text),
        "words": found,  # повертаємо всі входження з позиціями
    }), 200
```

alhoritm/routers/parseFile.py

```
from flask import Blueprint, jsonify, request

from services.parse_file_service import parse_uploaded_file
from utils.errors import ApiError
```

```
parseFile = Blueprint("parseFile", __name__)


@parseFile.route("/upload", methods=["POST"])
def parseText():
    if 'file' not in request.files:
        raise ApiError("No file part", status_code=400)
    file = request.files['file']

    try:
        payload, status = parse_uploaded_file(file)
        if status >= 400:
            raise ApiError(payload.get("error", "Failed to parse file"),
status_code=status)
        return jsonify(payload), status
    except Exception as e:
        if isinstance(e, ApiError):
            raise
        raise ApiError(f"Failed to parse file: {str(e)}", status_code=500)
```

alhoritm/routers/problems.py

```
from __future__ import annotations

from flask import Blueprint, jsonify, request

from services.problems_service import get_problems_payload
from utils.errors import ApiError

problems = Blueprint("problems", __name__)


@problems.route("/problems", methods=["POST"])
def get_problems():
    payload = request.get_json(silent=True) or {}
    body, status = get_problems_payload(payload)
    if status >= 400:
        raise ApiError(body.get("error", "Invalid request"), status_code=status,
details=body.get("details"))
    return jsonify(body), status
```

alhoritm/routers/profanity.py

```
from flask import Blueprint, request, jsonify
from utils.utilResponse import check_response
from services.profanity_service import find_profanity_words_detailed
from utils.errors import ApiError
from utils.schema import parse_json
from schemas.text_only import TextRequest


profanity = Blueprint("profanity", __name__)

@profanity.route("/profanity-score", methods=["POST"])
def check_profanity():
    data = request.get_json()
    parsed = parse_json(data, TextRequest)
    text = parsed.text

    raw_text = text.lower()
```

```
    # Використовуємо нову детальну функцію
    found = find_profanity_words_detailed(text)

    print("=== FOUND PROFANITY WORDS ===")
    for entry in found:
        print(
            f"{entry.get('text')} (lemma: {entry.get('lemma')}, pos:
{entry.get('pos')}, morph: {entry.get('morph_features')})"
        )

    # Унікальні за текстом слова (метрика), але повертаємо всі входження для
точного підсвічування
    unique_text = set()
    for entry in found:
        unique_text.add(entry.get('text'))

    return jsonify({
        "input_text": raw_text,
        "found_count": len(unique_text),
        "words": found,  # повертаємо всі входження з позиціями
    }), 200
```

alhoritm/routers/rare.py

```
from flask import Blueprint, request, jsonify
from utils.utilResponse import check_response
from services.rare_service import find_rare_words


rare = Blueprint("rare", __name__)

@rare.route("/rare-score", methods=["POST"])
def check_rare():
    text = check_response(request)
    raw_text = text.lower()

    found = find_rare_words(text)
    print("=== FOUND RARE WORDS ===")
    for entry in found:
        print(
            f"{entry.get('text')} (lemma: {entry.get('lemma')}, pos:
{entry.get('pos')}, category: {entry.get('category')})"
        )
    # Унікальні за текстом слова (метрика), але повертаємо всі входження для
точного підсвічування
    unique_text = set()
    for entry in found:
        unique_text.add(entry.get('text'))

    return jsonify({
        "input_text": raw_text,
        "found_count": len(unique_text),
        "words": found,  # повертаємо всі входження з позиціями
    }), 200
```

alhoritm/routers/replacements.py

```
from flask import Blueprint, current_app, jsonify, request

from services import replacement_service_layer
from utils.errors import ApiError
```

```
from utils.schema import parse_json
from schemas.replacements import ReplaceWordRequest, ReplaceMultipleRequest

replacements = Blueprint('replacements', __name__)


def _set_cors_headers(response):
    allowed_origins = current_app.config.get("CORS_ORIGINS", [])
    request_origin = request.headers.get("Origin")
    if request_origin and request_origin in allowed_origins:
        response.headers["Access-Control-Allow-Origin"] = request_origin
    elif allowed_origins:
        response.headers["Access-Control-Allow-Origin"] = allowed_origins[0]
    response.headers["Access-Control-Allow-Headers"] = "Content-
Type,Authorization"
    response.headers["Access-Control-Allow-Methods"] =
"GET,PUT,POST,DELETE,OPTIONS"
    return response


@replacements.route('/replace-word', methods=['POST', 'OPTIONS'])
def replace_word():
    if request.method == 'OPTIONS':
        response = jsonify({})
        return _set_cors_headers(response)
    """
    API для заміни одного слова
    """
    try:
        data = request.get_json()
        parsed = parse_json(data, ReplaceWordRequest)
        payload =
replacement_service_layer.replace_word_payload(parsed.model_dump())

        status = 200
        if payload.get("error"):
            raise ApiError(payload.get("error"), status_code=400)
        response = jsonify(payload)
        return _set_cors_headers(response), status

    except Exception as e:
        if isinstance(e, ApiError):
            raise
        raise ApiError(f"Server error: {str(e)}", status_code=500)


@replacements.route('/replace-multiple', methods=['POST'])
def replace_multiple():
    try:
        data = request.get_json()
        parsed = parse_json(data, ReplaceMultipleRequest)
        payload =
replacement_service_layer.replace_multiple_payload(parsed.model_dump())
        if payload.get("error"):
            raise ApiError(payload.get("error"), status_code=400)
        return jsonify(payload), 200
    except Exception as e:
        if isinstance(e, ApiError):
            raise
        raise ApiError(f"Server error: {str(e)}", status_code=500)
```

```
@replacements.route('/dictionary-stats', methods=['GET'])
def dictionary_stats():
    try:
        payload = replacement_service_layer.dictionary_stats_payload()
        return jsonify(payload)
    except Exception as e:
        if isinstance(e, ApiError):
            raise
        raise ApiError(f"Server error: {str(e)}", status_code=500)


@replacements.route('/moderation/pending', methods=['GET'])
def pending_replacements():
    try:
        payload = replacement_service_layer.pending_replacements_payload()
        response = jsonify(payload)
    except Exception as exc:
        if isinstance(exc, ApiError):
            raise
        raise ApiError(f"Server error: {exc}", status_code=500)

    return _set_cors_headers(response)


@replacements.route('/moderation/verify', methods=['POST'])
def verify_replacement():
    data = request.get_json() or {}
    payload, status = replacement_service_layer.verify_replacement_payload(data)
    if status >= 400:
        raise ApiError(payload.get("error", "Verification failed"),
status_code=status)
    response = jsonify(payload)
    return _set_cors_headers(response), status


@replacements.route('/moderation/decline', methods=['POST'])
def decline_replacement():
    data = request.get_json() or {}
    payload, status =
replacement_service_layer.decline_replacement_payload(data)
    if status >= 400:
        raise ApiError(payload.get("error", "Decline failed"),
status_code=status)
    response = jsonify(payload)
    return _set_cors_headers(response), status
```

alhoritm/routers/slang.py

```
from flask import Blueprint, jsonify, request

from config.experiment import DEFAULT_CONFIDENCE_THRESHOLD
import services.slang_service as slang_service
from utils.errors import ApiError
from utils.schema import parse_json
from schemas.slang import SlangRequest

slang = Blueprint("slang", __name__)
```

```
@slang.route("/slang-score", methods=["POST"])
def detect_slang():
    data = request.get_json(silent=True)
    parsed = parse_json(data, SlangRequest)
    text = parsed.text

    confidence_threshold = parsed.confidence_threshold
    if confidence_threshold is None:
        confidence_threshold = DEFAULT_CONFIDENCE_THRESHOLD

    if slang_service.slang_pipe is None:
        slang_service.init_slang_model()
    if slang_service.slang_pipe is None:
        raise ApiError("Модель не ініціалізована", status_code=500)

    found = slang_service.get_slang_words_detailed(text, confidence_threshold)

    print("=== FOUND SLANG WORDS ===")
    for entry in found:
        print(
            f"{entry['text']} (lemma: {entry['lemma']}, pos: {entry['pos']},
morph: {entry['morph_features']}, score: {entry.get('score', 'N/A')})"
        )

    unique = {}
    for entry in found:
        unique[entry['text']] = entry

    old_format_tokens = []
    for entry in found:
        old_format_tokens.append({
            "label": "SLANG",
            "word": entry['text'],
            "score": entry.get('score', 1.0),
            "type": entry.get('type', 'word'),
            "original_prediction": entry.get('original_prediction', 'SLANG'),
            "confident": entry.get('confident', True),
        })

    response = {
        "debug": {
            "all_tokens": old_format_tokens,
            "confidence_threshold": confidence_threshold,
        },
        "found_count": len(unique),
        "input_text": text,
        "words": found,
        "confidence_threshold": confidence_threshold,
    }

    return jsonify(response), 200
```

alhoritm/routers/survey.py

```
from flask import Blueprint, jsonify, request

from services.survey_service import list_surveys_payload, save_survey_payload
from utils.errors import ApiError
from utils.schema import parse_json
```

```
from schemas.survey import SurveyRequest

survey = Blueprint("survey", __name__)


@survey.route("/survey", methods=["POST"])
def save_survey():
    payload = request.get_json(silent=True) or {}
    parsed = parse_json(payload, SurveyRequest)
    body, status = save_survey_payload(parsed.model_dump())
    if status >= 400:
        raise ApiError(body.get("message", "Invalid survey payload"),
status_code=status)
    return jsonify(body), status


@survey.route("/survey", methods=["GET"])
def list_surveys():
    body, status = list_surveys_payload(request.args.get("limit"))
    if status >= 400:
        raise ApiError(body.get("message", "Invalid request"),
status_code=status)
    return jsonify(body), status
```

alhoritm/routers/syntax.py

```
from flask import Blueprint, jsonify, request

from services import syntax_service
from utils.utilResponse import check_response

syntax = Blueprint("syntax", __name__)

nlp = syntax_service.nlp
flesch_kincaid_algorithm = syntax_service.flesch_kincaid_algorithm
smog_algorithm = syntax_service.smog_algorithm
d_level_algorithm = syntax_service.d_level_algorithm


@syntax.route("/flesch-kincaid", methods=["POST"])
def flesch_kincaid_analysis():
    text = check_response(request)

    response = syntax_service.build_flesch_kincaid_response(text,
flesch_kincaid_algorithm)

    if "error" in response.get("flesch_kincaid", {}):
        return jsonify(response), 400

    return jsonify(response), 200


@syntax.route("/syntax-stats", methods=["POST"])
def syntax_statistics():
    text = check_response(request)

    pipeline = nlp or syntax_service.get_nlp()
    response = syntax_service.build_syntax_stats_response(text,
flesch_kincaid_algorithm, pipeline)
    return jsonify(response), 200
```

```
@syntax.route("/smog", methods=["POST"])
def smog_analysis():
    text = check_response(request)

    response = syntax_service.build_smog_response(text, smog_algorithm)

    if "error" in response.get("smog", {}):
        return jsonify(response), 400

    return jsonify(response), 200


@syntax.route("/d-level", methods=["POST"])
def d_level_analysis():
    text = check_response(request)

    response = syntax_service.build_d_level_response(text, d_level_algorithm)

    if "error" in response.get("d_level", {}):
        return jsonify(response), 400

    return jsonify(response), 200


@syntax.route("/algorithms", methods=["GET"])
def list_algorithms():
    response = syntax_service.build_algorithms_response(
        flesch_kincaid_algorithm,
        smog_algorithm,
        d_level_algorithm,
    )
    return jsonify(response), 200
```

alhoritm/routers/term.py

```
from flask import Blueprint, jsonify, request

from services.term_service import (
    HIGH_CONFIDENCE_THRESHOLD,
    TERM_CUT,
    add_morphological_info_to_terms,
    clean_and_normalize_text,
    COMMON_WORDS_BLACKLIST,
    extract_compound_terms,
    find_context_terms,
    get_ner_pipe,
    merge_subword_tokens,
    ner_pipe,
    post_process_bert_results,
    PROPER_NAMES_BLACKLIST,
    TERM_COMPONENTS,
    TERM_PREFIXES,
    TERM_SUFFIXES,
)
from utils.utilResponse import check_response
from utils.errors import ApiError
from utils.schema import parse_json
from schemas.text_only import TextRequest
```

```
term = Blueprint("term", __name__)


@term.route("/term-score", methods=["POST"])
def check_term():
    data = request.get_json()
    parsed = parse_json(data, TextRequest)
    text = parsed.text

    print("=== ORIGINAL TEXT ===", text)

    cleaned_text = clean_and_normalize_text(text)

    pipe = get_ner_pipe()
    if pipe is None:
        raise ApiError("Модель не ініціалізована", status_code=500)
    bert_results = pipe(cleaned_text)

    print("=== BERT RAW RESULTS ===")
    for r in bert_results:
        print(f"{r['word']} — {r['entity_group']} (score={r['score']:.3f})")

    merged_results = merge_subword_tokens(bert_results)

    print("=== MERGED RESULTS ===")
    for r in merged_results:
        print(f"{r['word']} — {r['entity_group']} (score={r['score']:.3f})")

    compound_results = extract_compound_terms(merged_results)

    print("=== COMPOUND RESULTS ===")
    for r in compound_results:
        print(f"{r['word']} — {r['entity_group']} (score={r['score']:.3f})")

    filtered_terms = post_process_bert_results(compound_results, cleaned_text)

    print("=== FILTERED TERMS ===")
    for r in filtered_terms:
        print(
            f"{r['word']} — {r['entity_group']} (score={r['score']:.3f})
[features: {r.get('has_term_features', False)}]"
        )

    context_terms = find_context_terms(cleaned_text, filtered_terms)

    if context_terms:
        print("=== CONTEXT TERMS ===")
        for r in context_terms:
            print(
                f"{r['word']} — {r['entity_group']} (score={r['score']:.3f})
[base: {r['context_base']}]"
            )

    all_terms = filtered_terms + context_terms

    unique_terms = {}
    for term_entry in all_terms:
        word = term_entry['word']
        if word not in unique_terms or term_entry['score'] >
```

```
unique_terms[word]['score']:
            unique_terms[word] = term_entry

    final_terms = sorted(unique_terms.values(), key=lambda x: x['score'],
reverse=True)

    terms_with_morphology = add_morphological_info_to_terms(cleaned_text,
final_terms)

    print("=== TERMS WITH MORPHOLOGY ===")
    for r in terms_with_morphology:
        print(
            f"{r['text']} (lemma: {r['lemma']}, pos: {r['pos']}, morph:
{r['morph_features']}, score: {r['score']:.3f})"
        )

    unique_morphological = {}
    for entry in terms_with_morphology:
        unique_morphological[entry['text']] = entry

    terms_words = [r['word'] for r in final_terms]

    return jsonify({
        "input_text": text,
        "found_count": len(unique_morphological),
        "words": list(unique_morphological.values()),
        "debug": {
            "bert_raw_count": len(bert_results),
            "merged_count": len(merged_results),
            "compound_count": len(compound_results),
            "filtered_count": len(filtered_terms),
            "context_count": len(context_terms) if context_terms else 0,
            "final_count": len(final_terms),
            "morphological_count": len(terms_with_morphology),
            "threshold_used": TERM_CUT,
            "high_confidence_threshold": HIGH_CONFIDENCE_THRESHOLD,
            "legacy_terms": terms_words,
        },
    }), 200


@term.route("/term-test", methods=["POST"])
def test_term():
    """Тестовий endpoint для аналізу роботи алгоритму"""
    text = check_response(request)

    cleaned_text = clean_and_normalize_text(text)
    pipe = get_ner_pipe()
    if pipe is None:
        raise ApiError("Модель не ініціалізована", status_code=500)
    bert_results = pipe(cleaned_text)
    merged_results = merge_subword_tokens(bert_results)
    compound_results = extract_compound_terms(merged_results)
    filtered_terms = post_process_bert_results(compound_results, cleaned_text)

    return jsonify({
        "input_text": text,
        "steps": {
            "1_bert_raw": [{"word": r['word'], "score": r['score'], "entity":
r['entity_group']} for r in bert_results],
```

```
            "2_merged": [{"word": r['word'], "score": r['score'], "entity":
r['entity_group']} for r in merged_results],
            "3_compound": [{"word": r['word'], "score": r['score'], "entity":
r['entity_group']} for r in compound_results],
            "4_filtered": [{"word": r['word'], "score": r['score'], "entity":
r['entity_group'], "has_features": r.get('has_term_features', False)} for r in
filtered_terms],
        },
        "stats": {
            "bert_count": len(bert_results),
            "merged_count": len(merged_results),
            "compound_count": len(compound_results),
            "filtered_count": len(filtered_terms),
            "blacklist_size": len(PROPER_NAMES_BLACKLIST) +
len(COMMON_WORDS_BLACKLIST),
            "term_features_count": len(TERM_SUFFIXES) + len(TERM_PREFIXES) +
len(TERM_COMPONENTS),
            "filters": {
                "proper_names": len(PROPER_NAMES_BLACKLIST),
                "common_words": len(COMMON_WORDS_BLACKLIST),
                "numbers_filtered": True,
            },
        },
    }), 200
```

alhoritm/schemas/analyze.py

```
from __future__ import annotations

from typing import Optional, Tuple

from pydantic import BaseModel, Field, field_validator


class Scope(BaseModel):
    start: int = 0
    end: int = 0


class AnalyzeRequest(BaseModel):
    text: str
    scope: Optional[Scope] = None
    include_cai: bool = True
    confidence_threshold: float = Field(default=0.7, ge=0.0, le=1.0)

    @field_validator("text")
    @classmethod
    def text_not_empty(cls, v: str) -> str:
        if not isinstance(v, str) or not v.strip():
            raise ValueError("Field 'text' must be a non-empty string.")
        return v
```

alhoritm/schemas/iterative_simplify.py

```
from __future__ import annotations

from typing import Any, Dict, Optional

from pydantic import BaseModel, Field, field_validator
```

```
class Scope(BaseModel):
    start: int = 0
    end: int = 0


class SyntaxConfig(BaseModel):
    enabled: bool = False
    max_iterations: int = Field(default=1, ge=1)
    min_improvement: float = Field(default=0.01, ge=0.0)


class IterativeSimplifyRequest(BaseModel):
    text: str
    scope: Optional[Scope] = None
    max_changes: Optional[int] = Field(default=None, ge=0)
    steps: Optional[int] = Field(default=None, ge=1)
    allow_llm: bool = False
    strict_pos: bool = True
    include_analysis: bool = False
    include_cai: bool = True
    confidence_threshold: Optional[float] = Field(default=None, ge=0.0, le=1.0)
    syntax: Optional[SyntaxConfig] = None
    redetect_every_batches: Optional[int] = Field(default=None, ge=1)
    cursor_index: Optional[int] = Field(default=None, ge=0)

    @field_validator("text")
    @classmethod
    def text_not_empty(cls, v: str) -> str:
        if not isinstance(v, str) or not v.strip():
            raise ValueError("Field 'text' must be a non-empty string.")
        return v
```

alhoritm/schemas/replacements.py

```
from __future__ import annotations

from typing import List, Optional

from pydantic import BaseModel, Field, field_validator


class MorphInfo(BaseModel):
    lemma: Optional[str] = None
    upos: Optional[str] = None
    morph_features: dict = Field(default_factory=dict)


class ReplaceWordRequest(BaseModel):
    word: str
    type: str = "profanity"
    morph_info: Optional[MorphInfo] = None
    allow_llm: bool = False
    context: Optional[str] = None

    @field_validator("word")
    @classmethod
    def word_not_empty(cls, v: str) -> str:
        if not isinstance(v, str) or not v.strip():
            raise ValueError("word cannot be empty")
        return v.strip()
```

```
class ReplaceMultipleItem(BaseModel):
    word: str
    type: str = "profanity"
    allow_llm: bool = False

    @field_validator("word")
    @classmethod
    def word_not_empty(cls, v: str) -> str:
        if not isinstance(v, str) or not v.strip():
            raise ValueError("word cannot be empty")
        return v.strip()


class ReplaceMultipleRequest(BaseModel):
    words: List[ReplaceMultipleItem]
```

alhoritm/schemas/slang.py

```
from __future__ import annotations

from typing import Optional

from pydantic import BaseModel, Field, field_validator


class SlangRequest(BaseModel):
    text: str
    confidence_threshold: Optional[float] = Field(default=None, ge=0.0, le=1.0)

    @field_validator("text")
    @classmethod
    def text_not_empty(cls, v: str) -> str:
        if not isinstance(v, str) or not v.strip():
            raise ValueError("Field 'text' must be a non-empty string.")
        return v
```

alhoritm/schemas/survey.py

```
from __future__ import annotations

from typing import List, Optional

from pydantic import BaseModel, Field, field_validator


class SurveyRequest(BaseModel):
    rating: float
    reasons: List[str] = Field(default_factory=list)
    fileName: Optional[str] = None
    page: Optional[int] = None
    simplificationLevel: Optional[str] = None
    date: Optional[str] = None

    @field_validator("rating")
    @classmethod
    def rating_range(cls, v: float) -> float:
        if not (1 <= v <= 10):
            raise ValueError("Rating must be between 1 and 10.")
```

```
        return v
```

alhoritm/schemas/text_only.py

```
from __future__ import annotations

from pydantic import BaseModel, field_validator


class TextRequest(BaseModel):
    text: str

    @field_validator("text")
    @classmethod
    def text_not_empty(cls, v: str) -> str:
        if not isinstance(v, str) or not v.strip():
            raise ValueError("Field 'text' must be a non-empty string.")
        return v
```

alhoritm/utils/config.py

```
"""
Утиліти для роботи з конфігурацією та змінними середовища
"""
import os
from pathlib import Path

# Шлях до .env файлу
ENV_FILE = Path(__file__).parent.parent / '.env'


def load_env():
    """Завантажує змінні з .env файлу"""
    if not ENV_FILE.exists():
        print(f"⚠️  Файл .env не знайдено: {ENV_FILE}")
        return

    with open(ENV_FILE, 'r') as f:
        for line in f:
            line = line.strip()
            # Пропускаємо пусті рядки та коментарі
            if not line or line.startswith('#'):
                continue

            # Парсимо KEY=VALUE
            if '=' in line:
                key, value = line.split('=', 1)
                os.environ[key.strip()] = value.strip()

    print(f"✅ Завантажено змінні з {ENV_FILE}")


def get_hf_token():
    """Отримує HuggingFace токен"""
    # Спробувати завантажити з .env якщо ще не завантажено
    if 'HF_TOKEN' not in os.environ:
        load_env()

    token = os.environ.get('HF_TOKEN')

    if not token:
```

```
        raise ValueError(
            "HF_TOKEN не знайдено! "
            "Додайте HF_TOKEN=... у файл .env"
        )

    return token


def get_openai_key():
    """Отримує OpenAI API ключ (опціонально)"""
    if 'OPENAI_API_KEY' not in os.environ:
        load_env()

    return os.environ.get('OPENAI_API_KEY')


def get_google_api_key():
    """Отримує Google Generative Language API ключ"""
    if 'GOOGLE_API_KEY' not in os.environ:
        load_env()

    key = os.environ.get('GOOGLE_API_KEY')
    if not key:
        raise ValueError("GOOGLE_API_KEY не знайдено! Додайте GOOGLE_API_KEY=...
у файл .env")

    return key


# Автоматично завантажити при імпорті
load_env()
```

alhoritm/utils/errors.py

```
from __future__ import annotations

import uuid
from dataclasses import dataclass
from typing import Any, Dict, Optional


@dataclass
class ApiError(Exception):
    message: str
    status_code: int = 500
    code: Optional[str] = None
    details: Optional[Dict[str, Any]] = None

    def to_dict(self) -> Dict[str, Any]:
        payload = {
            "error": self.message,
            "code": self.code,
            "details": self.details or {},
        }
        return payload


def new_trace_id() -> str:
    return uuid.uuid4().hex
```

alhoritm/utils/explanation_markup.py

```
from __future__ import annotations

import re
from typing import List, Tuple


_EXPL_START_RE = re.compile(r"\(\s*тобто\b", re.IGNORECASE)


def find_explanation_spans(text: str) -> List[Tuple[int, int]]:
    """
    Finds spans of parenthetical explanations in the form "(тобто ...)".
    Returns list of (start, end) where end is exclusive.
    """
    if not isinstance(text, str) or not text:
        return []

    spans: List[Tuple[int, int]] = []
    for m in _EXPL_START_RE.finditer(text):
        start = m.start()
        close = text.find(")", m.end())
        if close == -1:
            continue
        end = close + 1
        spans.append((start, end))
    return spans


def is_followed_by_explanation(text: str, end_pos: int) -> bool:
    """
    True if the substring starting at end_pos contains an explanation marker
"(тобто ...)".
    Designed to detect already-expanded terms: "<term> (тобто ...)".
    """
    if not isinstance(text, str) or not text:
        return False
    if not isinstance(end_pos, int) or end_pos < 0 or end_pos > len(text):
        return False
    tail = text[end_pos:]
    # Allow whitespace between term and '('.
    tail = tail.lstrip()
    return bool(_EXPL_START_RE.match(tail))


def overlaps_any(start: int, end: int, spans: List[Tuple[int, int]]) -> bool:
    if not isinstance(start, int) or not isinstance(end, int) or end <= start:
        return False
    for s, e in spans:
        if start < e and end > s:
            return True
    return False
```

alhoritm/utils/linguistic_features.py

```
"""
Модуль для лінгвістичних ознак слів для покращеної сленг-детекції
"""
import re
```

```
import numpy as np
from sklearn.base import BaseEstimator, TransformerMixin

class LinguisticFeatureExtractor(BaseEstimator, TransformerMixin):
    """Extractor для лінгвістичних ознак слів"""

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        features = []
        for word in X:
            word_features = self._extract_word_features(word)
            features.append(word_features)
        return np.array(features)

    def _extract_word_features(self, word):
        """Витягує лінгвістичні ознаки з слова"""
        word = word.lower()

        # Основні метрики
        length = len(word)

        # Повторювані символи
        repeated_chars = len(re.findall(r'([а-яієїґa-z])\1{1,}', word))
        max_repeated = max([len(match.group()) for match in re.finditer(r'([а-
яієїґa-z])\1+', word)] + [1])

        # Голосні та приголосні (українська мова)
        ukrainian_vowels = 'аеєиіїоуюя'
        vowels = sum(1 for c in word if c in ukrainian_vowels)
        consonants = sum(1 for c in word if c.isalpha() and c not in
ukrainian_vowels)
        vowel_ratio = vowels / length if length > 0 else 0

        # Цифри та спеціальні символи
        digits = sum(1 for c in word if c.isdigit())
        special_chars = sum(1 for c in word if not c.isalnum())

        # Початкові/кінцеві символи
        starts_with_vowel = 1 if word and word[0] in ukrainian_vowels else 0
        ends_with_vowel = 1 if word and word[-1] in ukrainian_vowels else 0

        # Частотні n-грами в українській мові (індикатори сленгу)
        slang_patterns = [
            'жж', 'кк', 'хх', 'gg', 'оо', 'ее', 'іі',  # повтори
            'кул', 'топ', 'лол', 'кек', 'ржу',  # сленгові основи
        ]
        slang_pattern_count = sum(1 for pattern in slang_patterns if pattern in
word)

        # Характеристики довжини
        is_very_short = 1 if length <= 3 else 0
        is_very_long = 1 if length >= 12 else 0

        # Співвідношення типів символів
        alpha_ratio = sum(1 for c in word if c.isalpha()) / length if length > 0
else 0

        return [
```

```
            length,                     # 0: довжина слова
            repeated_chars,             # 1: кількість повторюваних символів
            max_repeated,               # 2: максимальна довжина повтору
            vowels,                     # 3: кількість голосних
            consonants,                 # 4: кількість приголосних
            vowel_ratio,                # 5: співвідношення голосних
            digits,                     # 6: кількість цифр
            special_chars,              # 7: кількість спеціальних символів
            starts_with_vowel,          # 8: починається з голосної
            ends_with_vowel,            # 9: закінчується голосною
            slang_pattern_count,        # 10: кількість сленгових патернів
            is_very_short,              # 11: дуже коротке слово
            is_very_long,               # 12: дуже довге слово
            alpha_ratio,                # 13: співвідношення букв
        ]
```

alhoritm/utils/llm_clients.py

```
"""
Клієнти для роботи з LLM моделями
- MamayLLM логіка тимчасово працює через Google Generative Language API (Gemma
2)
- Lapa LLM використовує HuggingFace Inference API
"""
import os
import re
import warnings
import time
import json
from typing import List, Optional, Any
from .config import get_hf_token, get_google_api_key

warnings.filterwarnings("ignore", message="urllib3 v2 only supports OpenSSL*",
category=Warning)
try:
    from urllib3.exceptions import NotOpenSSLWarning  # type: ignore

    warnings.filterwarnings("ignore", category=NotOpenSSLWarning)
except Exception:
    pass

import requests
from huggingface_hub import InferenceClient

_JSON_OBJECT_RE = re.compile(r"\{.*\}", re.DOTALL)


def _try_parse_replacements_json(text: str) -> List[str]:
    """
    Tries to parse:
      {"replacements":["...","...","..."]}
    Returns list of strings or [].
    """
    if not text or not isinstance(text, str):
        return []
    m = _JSON_OBJECT_RE.search(text.strip())
    if not m:
        return []
    raw = m.group(0).strip()
    try:
        obj: Any = json.loads(raw)
```

```
    except Exception:
        return []
    if not isinstance(obj, dict):
        return []
    reps = obj.get("replacements")
    if not isinstance(reps, list):
        return []
    out: List[str] = []
    for r in reps:
        if isinstance(r, str) and r.strip():
            out.append(r.strip())
    return out


class GemmaLLMClient:
    """
    Клієнт для роботи з Gemma 3 (тимчасова альтернатива MamayLLM) через Google
Generative Language API
    Використовується для простих категорій: slang, rare, abbreviations,
profanity
    """

    DEFAULT_MODEL_ID = "models/gemma-3-12b-it"
    MODEL_ENV_KEY = "GOOGLE_GEMMA_MODEL"
    TIMEOUT = 30  # секунд
    MAX_RETRIES = 3
    API_URL_TEMPLATE =
"https://generativelanguage.googleapis.com/v1beta/models/{model}:generateContent
"

    def __init__(self, model_id: Optional[str] = None):
        """Ініціалізує клієнт з Google API ключем"""
        self.api_key = get_google_api_key()
        self.model_id = model_id or os.environ.get(self.MODEL_ENV_KEY,
self.DEFAULT_MODEL_ID)
        model_slug = self.model_id[7:] if self.model_id.startswith("models/")
else self.model_id
        self.api_url = self.API_URL_TEMPLATE.format(model=model_slug)

    def get_synonyms(self, word: str, category: str, context: Optional[str] =
None) -> List[str]:
        """
        Отримує синоніми для слова через Gemma (Google API)

        Args:
            word: Слово для заміни
            category: Категорія слова (slang, rare, abbreviations, profanity)
            context: Опціональний контекст використання слова

        Returns:
            Список з 3 українських синонімів (простіших/доступніших)
        """
        prompt = self._build_prompt(word, category, context)

        for attempt in range(self.MAX_RETRIES):
            try:
                response = self._query_api(prompt)
                synonyms = self._parse_response(response)

                if synonyms and len(synonyms) > 0:
```

```
                    return synonyms[:3]  # Повертаємо топ-3

            except Exception as e:
                print(f"⚠️ Спроба {attempt + 1}/{self.MAX_RETRIES} не вдалась:
{e}")
                if attempt < self.MAX_RETRIES - 1:
                    time.sleep(2 ** attempt)  # Exponential backoff

        # Якщо всі спроби не вдались
        return []

    def _build_prompt(self, word: str, category: str, context: Optional[str] =
None) -> str:
        """Будує промпт для LLM"""
        category_descriptions = {
            "slang": "сленгове",
            "rare": "рідковживане",
            "abbreviations": "абревіатура",
            "profanity": "нецензурне"
        }

        category_desc = category_descriptions.get(category, "складне")

        base_prompt = f"""Ти експерт з української мови та спрощення текстів.
Потрібно підібрати простішу заміну для слова так, щоб її можна було підставити в
речення.

Вхідне слово (лема, називний відмінок): "{word}"
Категорія: {category_desc}

Завдання:
- Дай рівно 3 варіанти заміни українською: простіші синоніми або короткі
пояснення.
- Пріоритет: однослівні варіанти. Якщо одним словом не можна без втрати сенсу —
дозволено до 5 слів (виняток).
- Якщо значення неоднозначне — обери найпоширеніше в сучасній українській.

Вимоги:
- Використовуй лише українську мову
- Без дужок: не використовуй "(" або ")" і не пиши скорочення типу
"організа(тор)" — пиши слова повністю
- Не давай обрізків слів (типу "організа", "керів", "тва", "заві")
- Не повторюй вхідне слово дослівно
- Переважно називний відмінок (Н.в.) для іменників/словосполучень
- Уникай речень; це мають бути слова або короткі словосполучення
- Не повертай жодних "роздумів"/пояснень процесу/chain-of-thought; тільки
кінцевий результат

Формат відповіді: тільки JSON, без пояснень, без нумерації, без зайвого тексту:
{{"replacements":["...","...","..."]}}

Самоперевірка перед відповіддю:
- рівно 3 різні варіанти
- немає "(" або ")"
- жоден варіант не є обрізаним словом"""

        if context:
            base_prompt += f"\n\nКонтекст використання: {context}"

        return base_prompt
```

```
    def _query_api(self, prompt: str) -> str:
        """Відправляє запит до Google Generative Language API"""
        payload = {
            "contents": [
                {
                    "role": "user",
                    "parts": [{"text": prompt}]
                }
            ],
            "generationConfig": {
                "temperature": 0.7,
                "topP": 0.9,
                "maxOutputTokens": 180
            }
        }
        try:
            resp = requests.post(
                self.api_url,
                params={"key": self.api_key},
                json=payload,
                timeout=self.TIMEOUT
            )
            resp.raise_for_status()
            data = resp.json()
            candidates = data.get("candidates", [])
            if not candidates:
                return ""
            parts = candidates[0].get("content", {}).get("parts", [])
            texts = [part.get("text", "") for part in parts if part.get("text")]
            return "\n".join(texts)
        except requests.RequestException as exc:
            msg = str(exc)
            # Redact API key if it leaks into exception (e.g. URL with
?key=...).
            msg = re.sub(r"([?&]key=)[^&\\s()]+", r"\1REDACTED", msg)

            body = ""
            try:
                resp = getattr(exc, "response", None)
                if resp is not None:
                    body = getattr(resp, "text", "") or ""
            except Exception:
                body = ""
            if body:
                body = body.strip().replace("\n", " ")
                if len(body) > 500:
                    body = body[:500] + "…"
                raise Exception(f"Помилка запиту до Google API: {msg};
body={body}") from exc

            raise Exception(f"Помилка запиту до Google API: {msg}") from exc

    def _parse_response(self, response: str) -> List[str]:
        """Парсить відповідь від API і витягує синоніми"""
        if not response:
            return []

        reps = _try_parse_replacements_json(response)
        if reps:
```

```
            return reps

        # Fallback: numbered list (1. ..., 2. ..., ...)
        synonyms = []
        lines = response.strip().split('\n')

        for line in lines:
            line = line.strip()
            # Шукаємо формат "1. синонім" або "1) синонім"
            if line and (line[0].isdigit() or line.startswith('-')):
                # Видаляємо номер та очищаємо
                synonym = line.lstrip('0123456789.-) ').strip()
                if synonym and len(synonym.split()) <= 5:  # Макс 5 слів
                    synonyms.append(synonym)

        return synonyms


class MamayLLMClient(GemmaLLMClient):
    """
    Сумісний клас для старих тестів/викликів.
    Фактично використовує Gemma через Google API.
    """

    pass


class LapaLLMClient:
    """
    Клієнт для роботи з Lapa v0.1.2 через HuggingFace Inference API
    Використовується для складних категорій: terms, complex, obsolete + всі
речення
    """

    MODEL_ID = "lapa-llm/lapa-v0.1.2-instruct"
    TIMEOUT = 45  # секунд (більше, бо модель більша)
    MAX_RETRIES = 3

    def __init__(self):
        """Ініціалізує клієнт з HuggingFace токеном"""
        self.token = get_hf_token()
        # Використовуємо прямий URL router.huggingface.co, щоб обійти pipeline-
tag image-text-to-text
        self.model_url = f"https://router.huggingface.co/hf-
inference/models/{self.MODEL_ID}"
        self.client = InferenceClient(model=self.model_url, token=self.token,
timeout=self.TIMEOUT)

    def get_synonyms(self, word: str, category: str, context: Optional[str] =
None) -> List[str]:
        """
        Отримує синоніми для слова через Lapa LLM

        Args:
            word: Слово для заміни
            category: Категорія слова (terms, complex, obsolete)
            context: Опціональний контекст використання слова

        Returns:
            Список з 3 українських синонімів (простіших/доступніших)
```

```
        """
        prompt = self._build_prompt(word, category, context)

        for attempt in range(self.MAX_RETRIES):
            try:
                response = self._query_api(prompt)
                synonyms = self._parse_response(response)

                if synonyms and len(synonyms) > 0:
                    return synonyms[:3]  # Повертаємо топ-3

            except Exception as e:
                print(f"⚠️ Спроба {attempt + 1}/{self.MAX_RETRIES} не вдалась:
{e}")
                if attempt < self.MAX_RETRIES - 1:
                    time.sleep(2 ** attempt)  # Exponential backoff

        return []

    def simplify_sentence(self, sentence: str) -> Optional[str]:
        """
        Спрощує складне речення

        Args:
            sentence: Речення для спрощення

        Returns:
            Спрощена версія речення або None
        """
        prompt = self._build_sentence_prompt(sentence)

        for attempt in range(self.MAX_RETRIES):
            try:
                response = self._query_api(prompt)
                simplified = self._parse_sentence_response(response)

                if simplified:
                    return simplified

            except Exception as e:
                print(f"⚠️ Спроба {attempt + 1}/{self.MAX_RETRIES} не вдалась:
{e}")
                if attempt < self.MAX_RETRIES - 1:
                    time.sleep(2 ** attempt)

        return None

    def _build_prompt(self, word: str, category: str, context: Optional[str] =
None) -> str:
        """Будує промпт для LLM"""
        category_descriptions = {
            "terms": "професійний термін",
            "complex": "складне слово",
            "obsolete": "застаріле слово"
        }

        category_desc = category_descriptions.get(category, "складне")

        base_prompt = f"""Ти експерт з української мови та спрощення текстів.
Допоможи зробити текст доступнішим для людей з когнітивними особливостями.
```

```
Слово: "{word}"
Категорія: {category_desc}

Завдання: Запропонуй 3 простіші українські синоніми або пояснення одним-двома словами.

Вимоги:
- Використовуй лише українську мову
- Синоніми мають бути значно простішими
- Уникай складних термінів та професіоналізмів
- Використовуй повсякденну мову
- Кожен варіант на новому рядку

Формат відповіді:
1. перший_синонім
2. другий_синонім
3. третій_синонім"""

        if context:
            base_prompt += f"\n\nКонтекст: {context}"

        return base_prompt

    def _build_sentence_prompt(self, sentence: str) -> str:
        """Будує промпт для спрощення речення"""
        return f"""Ти експерт з української мови. Спрости наступне речення для людей з когнітивними особливостями.

Речення: "{sentence}"

Вимоги:
- Розбий складне речення на кілька простих
- Використовуй короткі речення (8-12 слів)
- Уникай пасивного стану
- Використовуй прямий порядок слів (підмет-присудок-додаток)
- Зберігай весь зміст оригіналу

Спрощена версія:"""

    def _query_api(self, prompt: str) -> str:
        """Відправляє запит до HuggingFace Inference API"""
        try:
            response = self.client.text_generation(
                prompt,
                max_new_tokens=200,
                temperature=0.7,
                top_p=0.9,
                do_sample=True,
                return_full_text=False
            )
            return response
        except Exception as e:
            # Модель може бути недоступна або завантажується
            if "503" in str(e) or "loading" in str(e).lower():
                raise Exception("Модель завантажується, спробуйте ще раз через кілька секунд")
            raise

    def _parse_response(self, response: str) -> List[str]:
```

```
        """Парсить відповідь від API і витягує синоніми"""
        if not response:
            return []

        synonyms = []
        lines = response.strip().split('\n')

        for line in lines:
            line = line.strip()
            if line and (line[0].isdigit() or line.startswith('-')):
                synonym = line.lstrip('0123456789.-) ').strip()
                if synonym and len(synonym.split()) <= 3:
                    synonyms.append(synonym)

        return synonyms

    def _parse_sentence_response(self, response: str) -> Optional[str]:
        """Парсить відповідь з спрощеним реченням"""
        if not response:
            return None

        # Очищаємо та повертаємо спрощений текст
        simplified = response.strip()
        return simplified if simplified else None


class GeminiFlashLLMClient:
    """
    Клієнт для Google Gemini 1.5 Flash (sentence-level задачі та складні слова)
    Використовується як тимчасова заміна Lapa для складних категорій/речень
    """

    DEFAULT_MODEL_ID = "gemini-1.5-flash-latest"
    MODEL_ENV_KEY = "GOOGLE_FLASH_MODEL"
    TIMEOUT = 45
    MAX_RETRIES = 3
    API_URL_TEMPLATE =
"https://generativelanguage.googleapis.com/v1beta/models/{model}:generateContent
"

    def __init__(self, model_id: Optional[str] = None):
        self.api_key = get_google_api_key()
        self.model_id = model_id or os.environ.get(self.MODEL_ENV_KEY,
self.DEFAULT_MODEL_ID)
        model_slug = self.model_id[7:] if self.model_id.startswith("models/")
else self.model_id
        self.api_url = self.API_URL_TEMPLATE.format(model=model_slug)

    def get_synonyms(self, word: str, category: str, context: Optional[str] =
None) -> List[str]:
        prompt = self._build_prompt(word, category, context)

        for attempt in range(self.MAX_RETRIES):
            try:
                # Lower temperature for more consistent lemma-level suggestions.
                response = self._query_api(prompt, temperature=0.65,
max_output_tokens=220)
                synonyms = self._parse_response(response)
                if synonyms:
                    return synonyms[:3]
```

```
            except Exception as exc:
                print(f"⚠️ Flash спроба {attempt + 1}/{self.MAX_RETRIES} не
вдалась: {exc}")
                if attempt < self.MAX_RETRIES - 1:
                    time.sleep(2 ** attempt)

        return []

    def simplify_sentence(self, sentence: str, context: Optional[str] = None) ->
Optional[str]:
        prompt = self._build_sentence_prompt(sentence, context)

        for attempt in range(self.MAX_RETRIES):
            try:
                response = self._query_api(prompt, temperature=0.7,
max_output_tokens=1000)
                simplified = self._parse_sentence_response(response)
                if simplified:
                    return simplified
            except Exception as exc:
                print(f"⚠️ Flash sentence спроба {attempt +
1}/{self.MAX_RETRIES} не вдалась: {exc}")
                if attempt < self.MAX_RETRIES - 1:
                    time.sleep(2 ** attempt)

        return None

    def _build_prompt(self, word: str, category: str, context: Optional[str] =
None) -> str:
        category_descriptions = {
            "complex": "складне слово або конструкція",
            "obsolete": "застаріле слово",
            "archaic": "архаїзм",
            "term": "професійний термін",
            "terms": "професійний термін",
            "slang": "сленгове слово",
            "rare": "рідковживане слово",
            "abbreviations": "абревіатура",
            "profanity": "нецензурне слово"
        }
        category_desc = category_descriptions.get(category, "складний фрагмент")

        prompt = f"""Ти експерт з української мови та спрощення текстів.
Потрібно підібрати простішу заміну для слова так, щоб її можна було підставити в
речення.

Вхідне слово (лема, називний відмінок): "{word}"
Категорія: {category_desc}

Завдання:
- Дай рівно 3 варіанти заміни українською: простіші синоніми або короткі
пояснення.
- Пріоритет: однослівні варіанти. Якщо одним словом не можна без втрати сенсу —
дозволено до 5 слів (виняток).
- Якщо значення неоднозначне — обери найпоширеніше в сучасній українській.

Вимоги:
- Тільки українська мова
- Без дужок: не використовуй "(" або ")" і не пиши скорочення типу
"організа(тор)" — пиши слова повністю
```

```
- Не давай обрізків слів (типу "організа", "керів", "тва", "заві")
- Не повторюй вхідне слово дослівно
- Переважно називний відмінок (Н.в.) для іменників/словосполучень
- Уникай речень; це мають бути слова або короткі словосполучення
- Не повертай жодних "роздумів"/пояснень процесу/chain-of-thought; тільки
кінцевий результат

Формат відповіді: тільки JSON, без пояснень, без нумерації, без зайвого тексту:
{{"replacements":["...","...","..."]}}

Самоперевірка перед відповіддю:
- рівно 3 різні варіанти
- немає "(" або ")"
- жоден варіант не є обрізаним словом"""

        if context:
            prompt += f"\n\nКонтекст: {context}"

        return prompt

    def _build_sentence_prompt(self, sentence: str, context: Optional[str] =
None) -> str:
        prompt = f"""Ти асистент доступності текстів. Спрости речення
українською.

Оригінал: "{sentence}"

Вимоги:
- Поділи довгі речення на 1-2 короткі (8-12 слів)
- Використовуй прямий порядок слів
- Забезпеч однаковий зміст
- Уникай складних підрядних конструкцій
- Пиши простою мовою
"""
        if context:
            prompt += f"\nКонтекст: {context}\n"

        prompt += "\nСпрощений варіант:"
        return prompt

    def _query_api(self, prompt: str, *, temperature: float, max_output_tokens:
int) -> str:
        # Gemini 2.5 models may spend most of the output budget on "thoughts",
producing truncated visible text.
        # For dictionary batches we prefer complete, deterministic JSON over
hidden reasoning.
        thinking_cfg = None
        try:
            if isinstance(self.model_id, str) and
self.model_id.startswith("gemini-2.5"):
                thinking_cfg = {"thinkingBudget": 0}
        except Exception:
            thinking_cfg = None

        payload = {
            "contents": [
                {
                    "role": "user",
                    "parts": [{"text": prompt}]
                }
```

```
            ],
            "generationConfig": {
                "temperature": float(temperature),
                "topP": 0.9,
                "maxOutputTokens": int(max_output_tokens),
                **({"thinkingConfig": thinking_cfg} if thinking_cfg else {}),
            }
        }
        try:
            resp = requests.post(
                self.api_url,
                params={"key": self.api_key},
                json=payload,
                timeout=self.TIMEOUT
            )
            resp.raise_for_status()
            data = resp.json()
            candidates = data.get("candidates", [])
            if not candidates:
                return ""
            parts = candidates[0].get("content", {}).get("parts", [])
            texts = [part.get("text", "") for part in parts if part.get("text")]
            return "\n".join(texts)
        except requests.RequestException as exc:
            msg = str(exc)
            msg = re.sub(r"([?&]key=)[^&\\s()]+", r"\1REDACTED", msg)

            body = ""
            try:
                resp = getattr(exc, "response", None)
                if resp is not None:
                    body = getattr(resp, "text", "") or ""
            except Exception:
                body = ""
            if body:
                body = body.strip().replace("\n", " ")
                if len(body) > 500:
                    body = body[:500] + "…"
                raise Exception(f"Помилка запиту до Google API: {msg};
body={body}") from exc

            raise Exception(f"Помилка запиту до Google API: {msg}") from exc

    def _parse_response(self, response: str) -> List[str]:
        if not response:
            return []

        reps = _try_parse_replacements_json(response)
        if reps:
            return reps

        synonyms = []
        lines = response.strip().split('\n')
        for line in lines:
            line = line.strip()
            if line and (line[0].isdigit() or line.startswith('-')):
                synonym = line.lstrip('0123456789.-) ').strip()
                if synonym and len(synonym.split()) <= 5:
                    synonyms.append(synonym)
        return synonyms
```

```
    def _parse_sentence_response(self, response: str) -> Optional[str]:
        if not response:
            return None
        simplified = response.strip()
        return simplified if simplified else None
```

alhoritm/utils/logging_setup.py

```
from __future__ import annotations

import json
import logging
import time
from typing import Any, Dict


class JsonFormatter(logging.Formatter):
    def format(self, record: logging.LogRecord) -> str:
        payload: Dict[str, Any] = {
            "ts": time.strftime("%Y-%m-%dT%H:%M:%SZ",
time.gmtime(record.created)),
            "level": record.levelname,
            "logger": record.name,
            "message": record.getMessage(),
        }
        if hasattr(record, "trace_id"):
            payload["trace_id"] = getattr(record, "trace_id")
        if record.exc_info:
            payload["exc_info"] = self.formatException(record.exc_info)
        return json.dumps(payload, ensure_ascii=False)


def configure_logging(level: int = logging.INFO) -> None:
    root = logging.getLogger()
    if root.handlers:
        return
    handler = logging.StreamHandler()
    handler.setFormatter(JsonFormatter())
    root.addHandler(handler)
    root.setLevel(level)
```

alhoritm/utils/morphology.py

```
"""
Утиліти для морфологічного узгодження замін
"""
import inspect
import stanza
from typing import Dict, Optional, Set

# Python 3.11 removed inspect.getargspec, but pymorphy2 still uses it.
# Provide a compatible shim early, before importing pymorphy2.
if not hasattr(inspect, "getargspec"):
    from collections import namedtuple

    _ArgSpec = namedtuple("ArgSpec", ["args", "varargs", "keywords",
"defaults"])

    def _getargspec(func):  # type: ignore[override]
        spec = inspect.getfullargspec(func)
```

```
        return _ArgSpec(spec.args, spec.varargs, spec.varkw, spec.defaults)

    inspect.getargspec = _getargspec  # type: ignore[attr-defined]

try:
    import pymorphy2
except ImportError:
    pymorphy2 = None

# Ініціалізуємо stanza pipeline для морфологічного аналізу
nlp = stanza.Pipeline(lang='uk', processors='tokenize,mwt,pos,lemma',
download_method=None)
MORPH_ANALYZER = pymorphy2.MorphAnalyzer(lang='uk') if pymorphy2 else None

CASE_MAP = {
    'Nom': 'nomn',
    'Gen': 'gent',
    'Dat': 'datv',
    'Acc': 'accs',
    'Ins': 'ablt',
    'Loc': 'loct',
    'Voc': 'voct'
}
NUMBER_MAP = {
    'Sing': 'sing',
    'Plur': 'plur'
}
GENDER_MAP = {
    'Masc': 'masc',
    'Fem': 'femn',
    'Neut': 'neut'
}
TENSE_MAP = {
    'Past': 'past',
    'Pres': 'pres',
    'Fut': 'futr'
}
PERSON_MAP = {
    '1': '1per',
    '2': '2per',
    '3': '3per'
}

def parse_morph_features(feats: str) -> Dict[str, str]:
    """Парсинг морфологічних ознак з формату stanza"""
    morph_features = {}
    if feats:
        for feat in feats.split('|'):
            if '=' in feat:
                key, value = feat.split('=', 1)
                morph_features[key] = value
    return morph_features

def get_word_morphology(word: str) -> Optional[Dict]:
    """Отримання морфологічної інформації для слова"""
    try:
        doc = nlp(word)
        if doc.sentences and doc.sentences[0].words:
            word_obj = doc.sentences[0].words[0]
            return {
```

```
                'lemma': word_obj.lemma.lower(),
                'upos': word_obj.upos,
                'morph_features': parse_morph_features(word_obj.feats or '')
            }
    except Exception as e:
        print(f"Помилка аналізу морфології для '{word}': {e}")
    return None

def validate_pos_compatibility(original_upos: str, replacement_word: str) ->
bool:
    """
    Перевіряє чи сумісні частини мови оригінального слова та заміни

    Args:
        original_upos: частина мови оригінального слова (напр. "VERB")
        replacement_word: слово-заміна для аналізу

    Returns:
        True якщо частини мови сумісні
    """
    # If replacement is a phrase (multi-word) or contains punctuation markup, we
treat it as compatible.
    # This is used for "explanation-style" replacements where strict POS
matching is not meaningful.
    if isinstance(replacement_word, str):
        candidate = replacement_word.strip()
        if any(ch.isspace() for ch in candidate) or any(ch in candidate for ch
in ("(", ")", "—", "-")):
            return True

    replacement_morph = get_word_morphology(replacement_word)
    if not replacement_morph:
        print(f"⚠️  Не вдалося визначити частину мови для заміни:
{replacement_word}")
        return False

    replacement_upos = replacement_morph['upos']

    # Прямі співпадіння
    if original_upos == replacement_upos:
        return True

    # Сумісні групи частин мови
    compatible_groups = [
        {'NOUN', 'PROPN'},  # Іменники та власні назви
        {'ADJ', 'ADV'},     # Прикметники та прислівники (частково)
        {'VERB', 'AUX'},    # Дієслова основні та допоміжні
    ]

    for group in compatible_groups:
        if original_upos in group and replacement_upos in group:
            return True

    print(f"❌ Несумісні частини мови: {original_upos} → {replacement_upos}
({replacement_word})")
    return False

def get_compatible_replacements(replacements: list, original_upos: str) -> list:
    """
    Фільтрує заміни залишаючи тільки сумісні з оригінальною частиною мови
```

```
    Args:
        replacements: список слів-замін
        original_upos: частина мови оригінального слова

    Returns:
        відфільтрований список сумісних замін
    """
    compatible = []

    for replacement in replacements:
        if validate_pos_compatibility(original_upos, replacement):
            compatible.append(replacement)
        else:
            print(f"🔄 Відкидаємо несумісну заміну: {replacement} (для
{original_upos})")

    return compatible

def inflect_word_to_match(replacement_lemma: str, target_morph: Dict) -> str:
    """
    Відмінює слово заміну згідно з морфологією цільового слова

    Args:
        replacement_lemma: лема слова-заміни (напр. "поганий")
        target_morph: морфологічні ознаки цільового слова

    Returns:
        відмінене слово заміни
    """
    try:
        # Отримуємо морфологічну інформацію про заміну
        replacement_info = get_word_morphology(replacement_lemma)
        if not replacement_info:
            return replacement_lemma

        # Перевіряємо чи частини мови співпадають
        target_upos = target_morph.get('upos', '')
        replacement_upos = replacement_info['upos']

        # Якщо частини мови не співпадають, повертаємо лему без змін
        if target_upos != replacement_upos:
            print(f"Частини мови не співпадають: {target_upos} vs
{replacement_upos}")
            return replacement_lemma

        target_features = target_morph.get('morph_features', {})

        if MORPH_ANALYZER:
            pymorphy_form = generate_with_pymorphy(replacement_lemma,
target_upos, target_features)
            if pymorphy_form:
                return pymorphy_form

        # Fallback до попередніх евристик
        if target_upos in ['NOUN', 'ADJ']:
            test_forms = generate_test_forms(replacement_lemma, target_features)
            best_form = find_best_form(test_forms, target_features)
            if best_form:
                return best_form
```

```
            # Fallback до леми
            return replacement_lemma

        except Exception as e:
            print(f"Помилка відмінювання '{replacement_lemma}': {e}")
            return replacement_lemma

    def generate_test_forms(lemma: str, target_features: Dict) -> list:
        """Генерує тестові форми слова для різних відмінків"""
        test_forms = []  # Спочатку порожній список

        # Орудний відмінок
        if target_features.get('Case') == 'Ins':
            if target_features.get('Gender') == 'Masc':
                # Чоловічий рід прикметники: поганий -> поганим
                if lemma.endswith('ий'):
                    test_forms.append(lemma[:-2] + 'им')
                elif lemma.endswith('ій'):
                    test_forms.append(lemma[:-2] + 'ім')
                # Іменники: капець -> капцем
                test_forms.extend([lemma + 'ем', lemma + 'ом', lemma + 'им'])

            elif target_features.get('Gender') == 'Neut':
                # Середній рід: пояснення -> поясненням
                if lemma.endswith('ння'):
                    test_forms.append(lemma[:-2] + 'ям')
                elif lemma.endswith('я'):
                    test_forms.append(lemma[:-1] + 'ям')
                test_forms.extend([lemma + 'м', lemma + 'ям'])

            elif target_features.get('Gender') == 'Fem':
                # Жіночий рід: погана -> поганою
                if lemma.endswith('а'):
                    test_forms.append(lemma[:-1] + 'ою')
                elif lemma.endswith('я'):
                    test_forms.append(lemma[:-1] + 'ею')

        # Знахідний відмінок
        elif target_features.get('Case') == 'Acc':
            if target_features.get('Gender') == 'Masc':
                if lemma.endswith('ий'):
                    test_forms.append(lemma[:-2] + 'ого')
            elif target_features.get('Gender') == 'Fem':
                if lemma.endswith('а'):
                    test_forms.append(lemma[:-1] + 'у')

        # Родовий відмінок
        elif target_features.get('Case') == 'Gen':
            if target_features.get('Gender') == 'Masc':
                if lemma.endswith('ий'):
                    test_forms.append(lemma[:-2] + 'ого')
                test_forms.extend([lemma + 'а', lemma + 'у'])
            elif target_features.get('Gender') == 'Fem':
                if lemma.endswith('а'):
                    test_forms.append(lemma[:-1] + 'ої')

        # Додаємо лему як fallback в кінець
        test_forms.append(lemma)
```

```
    return list(dict.fromkeys(test_forms))  # Унікальні форми зі збереженням
порядку

def find_best_form(test_forms: list, target_features: Dict) -> Optional[str]:
    """Знаходить найкращу форму серед тестових"""
    print(f"  Тестові форми: {test_forms}")

    for i, form in enumerate(test_forms):
        print(f"  Тестуємо форму #{i}: '{form}'")
        form_info = get_word_morphology(form)
        if form_info:
            print(f"    Морфологія: {form_info['morph_features']}")
            if forms_match(form_info['morph_features'], target_features):
                print(f"    ✅ Збіг! Вибрано: '{form}'")
                return form
            else:
                print(f"    ❌ Не збігається")
        else:
            print(f"    ❌ Не вдалося проаналізувати")

    # Якщо точний збіг не знайдено, повертаємо першу не-лема форму, або лему
    for form in test_forms[:-1]:  # Всі крім останньої (леми)
        if form != test_forms[-1]:  # Якщо це не лема
            print(f"  Fallback до першої відміненої форми: '{form}'")
            return form

    # Якщо нічого кращого немає, повертаємо лему
    result = test_forms[-1] if test_forms else None
    print(f"  Fallback до леми: '{result}'")
    return result

def forms_match(form_features: Dict, target_features: Dict) -> bool:
    """Перевіряє чи форми відповідають за ключовими ознаками"""
    key_features = ['Case', 'Gender', 'Number']

    for feature in key_features:
        if feature in target_features and feature in form_features:
            if form_features[feature] != target_features[feature]:
                return False
    return True

def build_pymorphy_tags(target_features: Dict) -> Set[str]:
    tags = set()
    case = target_features.get('Case')
    if case:
        tags.add(CASE_MAP.get(case, case.lower()))
    number = target_features.get('Number')
    if number:
        tags.add(NUMBER_MAP.get(number, number.lower()))
    gender = target_features.get('Gender')
    if gender:
        tags.add(GENDER_MAP.get(gender, gender.lower()))
    tense = target_features.get('Tense')
    if tense:
        tags.add(TENSE_MAP.get(tense, tense.lower()))
    person = target_features.get('Person')
    if person:
        tags.add(PERSON_MAP.get(person, person.lower()))
    return tags
```

```
def generate_with_pymorphy(lemma: str, upos: str, target_features: Dict) ->
Optional[str]:
    if not MORPH_ANALYZER:
        return None
    tags = build_pymorphy_tags(target_features)
    if not tags:
        return None
    try:
        parses = MORPH_ANALYZER.parse(lemma)
        if not parses:
            return None
        for parse in parses:
            word_tags = set(parse.tag.cyr_repr.split())
            if upos == 'ADJ' and 'ADJF' not in word_tags and 'ADJS' not in
word_tags:
                continue
            if upos in ['NOUN', 'PROPN'] and 'NOUN' not in word_tags:
                continue
            if upos == 'VERB' and 'VERB' not in word_tags and 'INFN' not in
word_tags:
                continue
            inflected = parse.inflect(tags)
            if inflected:
                return inflected.word
    except Exception as e:
        print(f"Помилка pymorphy при відмінюванні '{lemma}': {e}")
    return None

def apply_morphological_agreement(original_word: str, original_morph: Dict,
replacement: str) -> str:
    """
    Основна функція для застосування морфологічного узгодження

    Args:
        original_word: оригінальне слово що замінюється
        original_morph: морфологічна інформація оригінального слова
        replacement: слово-заміна (лема)

    Returns:
        морфологічно узгоджена заміна
    """
    print(f"Застосовуємо морфологічне узгодження:")
    print(f"  Оригінал: {original_word}")
    print(f"  Морфологія: {original_morph}")
    print(f"  Заміна: {replacement}")

    result = inflect_word_to_match(replacement, original_morph)
    print(f"  Результат: {result}")

    return result
```

alhoritm/utils/replacement_service.py

```
"""
Сервіс для роботи зі словниками замін (лексичний рівень)
Винесений з Flask-роутера, щоб надалі додати LLM-фолбек та кешування
"""
from __future__ import annotations

import json
```

```
import os
import re
from collections import OrderedDict
from datetime import datetime
from pathlib import Path
from typing import Dict, List, Optional, Tuple

from config.experiment import ENABLE_LLM_BY_KEY_DEFAULT

# Optional imports: kept for backwards compatibility, but runtime LLM-by-key is
disabled by default.
try:
    from .llm_clients import GemmaLLMClient, GeminiFlashLLMClient  # type:
ignore
except Exception:  # pragma: no cover
    GemmaLLMClient = None  # type: ignore
    GeminiFlashLLMClient = None  # type: ignore


class ReplacementService:
    """Завантаження, кешування та пошук замін у словниках"""

    GROUP_FILES = {
        'profanity_group': ['profanity.json', 'vulg.json', 'obscene.json'],
        'obsolete_group': ['obsolete.json', 'arch.json'],
        'complex_group': ['complex.json'],
        'rare': ['rare.json'],
        'slang': ['slang.json'],
        'abbr': ['abbr.json'],
        'terms': ['terms.json']
    }

    TYPE_TO_GROUP = {
        'profanity': 'profanity_group',
        'vulgar': 'profanity_group',
        'obscene': 'profanity_group',
        'bad': 'profanity_group',
        'complex': 'complex_group',
        'obsolete': 'obsolete_group',
        'archaic': 'obsolete_group',
        'rare_word': 'rare',
        'rare': 'rare',
        'term': 'terms',
        'terms': 'terms',
        'slang': 'slang',
        'abbreviation': 'abbr',
        'abbr': 'abbr'
    }

    SIMPLE_TYPES = {
        'slang',
        'rare',
        'rare_word',
        'abbreviation',
        'abbr',
        'profanity',
        'vulgar',
        'obscene',
        'bad',
        # Тимчасово покриваємо складні категорії Gemma LLM
```

```
        'complex',
        'obsolete',
        'archaic',
        'term',
        'terms'
    }

    LLM_CATEGORY_MAP = {
        'slang': 'slang',
        'rare': 'rare',
        'rare_word': 'rare',
        'abbreviation': 'abbreviations',
        'abbr': 'abbreviations',
        'profanity': 'profanity',
        'vulgar': 'profanity',
        'obscene': 'profanity',
        'bad': 'profanity',
        'complex': 'complex',
        'obsolete': 'obsolete',
        'archaic': 'obsolete',
        'term': 'terms',
        'terms': 'terms'
    }

    TEMPORARY_COMPLEX_TYPES = {
        'complex',
        'obsolete',
        'archaic',
        'term',
        'terms'
    }

    def __init__(
        self,
        replacements_dir: Optional[Path] = None,
        llm_client: Optional[object] = None,
        advanced_llm_client: Optional[object] = None,
        *,
        enable_substring_heuristic: bool = False,
    ) -> None:
        self.base_dir = replacements_dir or
Path(__file__).resolve().parent.parent / "data" / "replacements"
        self.source_dir = Path(__file__).resolve().parent.parent / "data" /
"source"
        self._dictionaries_cache: Dict[str, Dict] = {}
        self._word_forms_index_cache: Dict[str, Dict[str, str]] = {}
        self._lookup_cache: "OrderedDict[str, Optional[Dict]]" = OrderedDict()
        self._vulg_form_to_nominative_cache: Optional[Dict[str, str]] = None
        self._rare_form_to_nominative_cache: Optional[Dict[str, str]] = None
        self._slang_form_to_nominative_cache: Optional[Dict[str, str]] = None
        self._abbr_form_to_nominative_cache: Optional[Dict[str, str]] = None
        self.llm_client = llm_client
        self.advanced_llm_client = advanced_llm_client
        self._lookup_cache_max = 10_000
        self.enable_substring_heuristic = bool(enable_substring_heuristic)
        # LLM-by-key is opt-in and must be explicitly enabled via flag/env.
        self.enable_llm_by_key = (
            os.environ.get("ENABLE_LLM_BY_KEY",
str(ENABLE_LLM_BY_KEY_DEFAULT)).strip().lower()
            in {"1", "true", "yes", "on"}
```

```
        )

    def clear_cache(self) -> None:
        """Очищує кеш словників (наприклад, після оновлення файлів)"""
        self._dictionaries_cache.clear()
        self._word_forms_index_cache.clear()
        self._lookup_cache.clear()

    @staticmethod
    def _clean_word(word: str) -> str:
        """Очищує слово від пунктуації та приводить до нижнього регістру.

        Важливо: підтримує як кирилицю, так і латиницю (абревіатури типу `LLM`,
`API`, `JSON`),
        а також цифри (`F1`).
        """
        w = word.lower()
        return re.sub(r"^[^0-9a-za-яіїєґ'’]+|[^0-9a-za-яіїєґ'’]+$", "", w)

    def _lookup_cache_get(self, key: str) -> Optional[Dict]:
        if key not in self._lookup_cache:
            return None
        value = self._lookup_cache.pop(key)
        self._lookup_cache[key] = value
        return value

    def _lookup_cache_set(self, key: str, value: Optional[Dict]) -> None:
        if key in self._lookup_cache:
            self._lookup_cache.pop(key)
        self._lookup_cache[key] = value
        if len(self._lookup_cache) > self._lookup_cache_max:
            self._lookup_cache.popitem(last=False)

    def _load_file(self, filename: str) -> Dict:
        file_path = self.base_dir / filename
        with open(file_path, 'r', encoding='utf-8') as f:
            data = json.load(f)
        return data.get('replacements', {})

    def _load_vulg_form_to_nominative(self) -> Dict[str, str]:
        """
        Build a map of vulg surface forms -> nominative anchor from
`data/source/vulg.json`.
        Used when `replacements/vulg.json` is keyed by nominative (lemma-like)
rather than wordforms.
        """
        if self._vulg_form_to_nominative_cache is not None:
            return self._vulg_form_to_nominative_cache

        path = self.source_dir / "vulg.json"
        try:
            with open(path, "r", encoding="utf-8") as f:
                rows = json.load(f)
        except Exception:
            self._vulg_form_to_nominative_cache = {}
            return self._vulg_form_to_nominative_cache

        out: Dict[str, str] = {}
        if isinstance(rows, list):
            for row in rows:
```

```
                if not isinstance(row, dict):
                    continue
                w = row.get("word")
                n = row.get("nominative") or row.get("word")
                if not (isinstance(w, str) and w.strip() and isinstance(n, str)
and n.strip()):
                    continue
                cw = self._clean_word(w)
                cn = self._clean_word(n)
                if cw and cn:
                    out[cw] = cn

        self._vulg_form_to_nominative_cache = out
        return out

    def _load_rare_form_to_nominative(self) -> Dict[str, str]:
        if self._rare_form_to_nominative_cache is not None:
            return self._rare_form_to_nominative_cache
        path = self.source_dir / "rare.json"
        try:
            with open(path, "r", encoding="utf-8") as f:
                rows = json.load(f)
        except Exception:
            self._rare_form_to_nominative_cache = {}
            return self._rare_form_to_nominative_cache
        out: Dict[str, str] = {}
        if isinstance(rows, list):
            for row in rows:
                if not isinstance(row, dict):
                    continue
                w = row.get("word")
                n = row.get("nominative") or row.get("word")
                if not (isinstance(w, str) and w.strip() and isinstance(n, str)
and n.strip()):
                    continue
                cw = self._clean_word(w)
                cn = self._clean_word(n)
                if cw and cn:
                    out[cw] = cn
        self._rare_form_to_nominative_cache = out
        return out

    def _load_slang_form_to_nominative(self) -> Dict[str, str]:
        if self._slang_form_to_nominative_cache is not None:
            return self._slang_form_to_nominative_cache
        path = self.source_dir / "slang.json"
        try:
            with open(path, "r", encoding="utf-8") as f:
                rows = json.load(f)
        except Exception:
            self._slang_form_to_nominative_cache = {}
            return self._slang_form_to_nominative_cache
        out: Dict[str, str] = {}
        if isinstance(rows, list):
            for row in rows:
                if not isinstance(row, dict):
                    continue
                w = row.get("word")
                n = row.get("nominative") or row.get("word")
                if not (isinstance(w, str) and w.strip() and isinstance(n, str)
```

```
and n.strip()):
                    continue
                cw = self._clean_word(w)
                cn = self._clean_word(n)
                if cw and cn:
                    out[cw] = cn
        self._slang_form_to_nominative_cache = out
        return out

    def _load_abbr_form_to_nominative(self) -> Dict[str, str]:
        """
        Build a map of abbreviation surface forms -> nominative anchor from
`data/source/abbr.json`.
        This lets us keep `replacements/abbr.json` lemma-keyed (nominatives)
while still matching inflected forms.
        """
        if self._abbr_form_to_nominative_cache is not None:
            return self._abbr_form_to_nominative_cache
        path = self.source_dir / "abbr.json"
        try:
            with open(path, "r", encoding="utf-8") as f:
                rows = json.load(f)
        except Exception:
            self._abbr_form_to_nominative_cache = {}
            return self._abbr_form_to_nominative_cache

        out: Dict[str, str] = {}
        if isinstance(rows, list):
            for row in rows:
                if not isinstance(row, dict):
                    continue
                w = row.get("word")
                n = row.get("nominative") or row.get("word")
                if not (isinstance(w, str) and w.strip() and isinstance(n, str)
and n.strip()):
                    continue
                cw = self._clean_word(w)
                cn = self._clean_word(n)
                if cw and cn:
                    out[cw] = cn
        self._abbr_form_to_nominative_cache = out
        return out

    def _build_word_forms_index(self, dictionary: Dict) -> Dict[str, str]:
        index: Dict[str, str] = {}
        for _base_word, data in dictionary.items():
            if not isinstance(data, dict):
                continue
            word_forms = data.get('word_forms', {})
            if not isinstance(word_forms, dict):
                continue
            for form, replacement in word_forms.items():
                if not isinstance(form, str) or not isinstance(replacement,
str):
                    continue
                clean_form = self._clean_word(form)
                if not clean_form:
                    continue
                index.setdefault(clean_form, replacement)
        return index
```

```
    def load_group_dictionary(self, group_type: str) -> Dict:
        """Завантажує та кешує словник для конкретної логічної групи"""
        if group_type not in self._dictionaries_cache:
            combined_dict: Dict = {}
            for filename in self.GROUP_FILES.get(group_type, []):
                try:
                    replacements = self._load_file(filename)
                    combined_dict.update(replacements)
                except Exception as exc:
                    print(f"Помилка завантаження {filename}: {exc}")
            self._dictionaries_cache[group_type] = combined_dict
            self._word_forms_index_cache[group_type] =
self._build_word_forms_index(combined_dict)
        return self._dictionaries_cache[group_type]

    def find_word_replacement(self, word: str, dictionary: Dict) ->
Optional[Dict]:
        """Шукає конкретне слово у переданому словнику"""
        clean_word = self._clean_word(word)

        if clean_word in dictionary:
            return dictionary[clean_word]

        # Fast path: use prebuilt word_forms index (O(1) lookup instead of
scanning the whole dictionary).
        for group_type, dict_obj in self._dictionaries_cache.items():
            if dict_obj is dictionary:
                form_index = self._word_forms_index_cache.get(group_type, {})
                repl = form_index.get(clean_word)
                if repl:
                    return {
                        'priority_replacements': [
                            {
                                'replacement': repl,
                                'priority': 1,
                                'context': 'word_form',
                                'formality': 'neutral'
                            }
                        ],
                        'word_forms': {}
                    }
                break

        # Substring heuristic is disabled by default for accuracy.
        if self.enable_substring_heuristic:
            for key, data in dictionary.items():
                if key in clean_word or clean_word in key:
                    return data

        return None

    def get_group_for_type(self, word_type: str) -> str:
        """Повертає логічну групу для категорії слова"""
        return self.TYPE_TO_GROUP.get(word_type, 'profanity_group')

    def get_dictionary_replacement(
        self,
        word: str,
        word_type: str,
```

```
        *,
        lemma: Optional[str] = None,
    ) -> Tuple[Optional[Dict], str]:
        """Повертає пару (replacement_data, group_type) для подальшої обробки"""
        group_type = self.get_group_for_type(word_type)
        clean_word = self._clean_word(word)
        cache_key = f"{group_type}:{clean_word}"

        cached = self._lookup_cache_get(cache_key)
        if cache_key in self._lookup_cache:
            return cached, group_type

        dictionary = self.load_group_dictionary(group_type)
        # Lemma-first lookup can improve recall for inflected forms without
resorting to substring heuristics.
        replacement_data: Optional[Dict] = None
        if lemma:
            clean_lemma = self._clean_word(lemma)
            if clean_lemma and clean_lemma in dictionary:
                replacement_data = dictionary[clean_lemma]
        # For vulg, replacements can be stored as nominatives (lemmas), while
detection still happens on wordforms.
        # Try surface-form -> nominative mapping as a fallback before scanning
word_forms/substring heuristics.
        if replacement_data is None:
            try:
                if group_type == "profanity_group":
                    form_to_nom = self._load_vulg_form_to_nominative()
                    nom = form_to_nom.get(clean_word)
                    if nom and nom in dictionary:
                        replacement_data = dictionary[nom]
                elif group_type == "rare":
                    form_to_nom = self._load_rare_form_to_nominative()
                    nom = form_to_nom.get(clean_word)
                    if nom and nom in dictionary:
                        replacement_data = dictionary[nom]
                elif group_type == "slang":
                    form_to_nom = self._load_slang_form_to_nominative()
                    nom = form_to_nom.get(clean_word)
                    if nom and nom in dictionary:
                        replacement_data = dictionary[nom]
                elif group_type == "abbr":
                    form_to_nom = self._load_abbr_form_to_nominative()
                    nom = form_to_nom.get(clean_word)
                    if nom and nom in dictionary:
                        replacement_data = dictionary[nom]
            except Exception:
                pass
        if replacement_data is None:
            replacement_data = self.find_word_replacement(word, dictionary)
        if replacement_data is not None:
            metadata = replacement_data.get('metadata', {})
            if metadata.get('isVerify') is False:
                replacement_data = None
        self._lookup_cache_set(cache_key, replacement_data)
        return replacement_data, group_type

    def get_all_dictionaries(self) -> Dict[str, Dict]:
        """Використовується для статистики: завантажує всі групи"""
        all_dicts = {}
```

```
    for group_type in self.GROUP_FILES.keys():
        all_dicts[group_type] = self.load_group_dictionary(group_type)
    return all_dicts

def should_use_llm(self, word_type: str) -> bool:
    """Перевіряє, чи входить категорія до простих (для Gemma)"""
    return self.enable_llm_by_key and word_type in self.SIMPLE_TYPES

def _get_primary_filename(self, group_type: str) -> Optional[str]:
    files = self.GROUP_FILES.get(group_type, [])
    return files[0] if files else None

def _get_source_filename(self, group_type: str) -> Optional[str]:
    filename = self._get_primary_filename(group_type)
    if not filename:
        return None
    return filename

@staticmethod
def _group_to_category(group_type: str) -> str:
    if group_type == "profanity_group":
        return "profanity"
    if group_type == "obsolete_group":
        return "obsolete"
    if group_type == "complex_group":
        return "complex"
    return group_type

def add_source_entry(self, word: str, group_type: str) -> bool:
    filename = self._get_source_filename(group_type)
    if not filename:
        return False

    file_path = self.source_dir / filename
    if file_path.exists():
        with open(file_path, 'r', encoding='utf-8') as f:
            data = json.load(f)
    else:
        data = []

    if not isinstance(data, list):
        return False

    clean_word = self._clean_word(word)
    for item in data:
        existing = self._clean_word(item.get('word', ''))
        if existing == clean_word:
            return True

    data.append({
        "word": word,
        "nominative": word,
        "pos": "",
        "form": "",
        "info": "",
        "category": self._group_to_category(group_type),
    })

    with open(file_path, 'w', encoding='utf-8') as f:
        json.dump(data, f, ensure_ascii=False, indent=2)
```

```
        self.clear_cache()
        return True

    def _filter_synonyms(self, word: str, synonyms: List[str]) -> List[str]:
        """Видаляє повтори та оригінал, повертає максимум 3 синоніми"""
        clean_original = self._clean_word(word)
        seen = set()
        filtered: List[str] = []

        for synonym in synonyms:
            candidate = synonym.strip()
            if not candidate:
                continue
            normalized = self._clean_word(candidate)
            if not normalized or normalized == clean_original:
                continue
            if normalized in seen:
                continue
            filtered.append(candidate)
            seen.add(normalized)
            if len(filtered) >= 3:
                break

        # Повертаємо лише якщо є хоча б 2 різні варіанти
        return filtered if len(filtered) >= 2 else []

    def _get_llm_client_for_type(self, word_type: str):
        """Повертає відповідний LLM клієнт залежно від категорії"""
        if not self.enable_llm_by_key:
            return None
        if GemmaLLMClient is None and GeminiFlashLLMClient is None:
            return None

        if word_type in self.TEMPORARY_COMPLEX_TYPES:
            if self.advanced_llm_client is None:
                if GeminiFlashLLMClient is None:
                    return None
                self.advanced_llm_client = GeminiFlashLLMClient()
            return self.advanced_llm_client

        if self.llm_client is None:
            if GemmaLLMClient is None:
                return None
            self.llm_client = GemmaLLMClient()
        return self.llm_client

    def generate_llm_replacement_entry(self, word: str, word_type: str, context:
Optional[str] = None) -> Optional[Dict]:
        """Запитує відповідний LLM та повертає запис для словника"""
        if not self.should_use_llm(word_type):
            return None

        llm_category = self.LLM_CATEGORY_MAP.get(word_type, 'slang')
        client = self._get_llm_client_for_type(word_type)
        if client is None:
            return None
        synonyms = client.get_synonyms(word, llm_category, context=context)
        filtered = self._filter_synonyms(word, synonyms)
```

```
        if not filtered:
            return None

        priority_items = []
        for idx, synonym in enumerate(filtered, start=1):
            priority_items.append({
                'replacement': synonym,
                'priority': idx,
                'context': llm_category,
                'formality': 'neutral'
            })

        source_label = 'gemini_flash_llm' if word_type in
self.TEMPORARY_COMPLEX_TYPES else 'gemma_llm'

        metadata = {
            'source': source_label,
            'generated_at': datetime.utcnow().isoformat() + 'Z',
            'isVerify': False,
            'model': getattr(client, 'model_id', None),
            'original_category': word_type,
            'temporary_solution': word_type in self.TEMPORARY_COMPLEX_TYPES,
            'needs_review': word_type in self.TEMPORARY_COMPLEX_TYPES
        }

        return {
            'priority_replacements': priority_items,
            'word_forms': {},
            'metadata': metadata
        }

    def save_replacement_entry(self, word: str, entry: Dict, group_type: str) ->
bool:
        """Зберігає запис у відповідний JSON словника"""
        filename = self._get_primary_filename(group_type)
        if not filename:
            return False

        file_path = self.base_dir / filename
        if file_path.exists():
            with open(file_path, 'r', encoding='utf-8') as f:
                data = json.load(f)
        else:
            data = {
                'metadata': {
                    'category': group_type,
                    'version': '1.0',
                    'last_updated': '',
                    'total_words': 0
                },
                'replacements': {}
            }

        replacements = data.setdefault('replacements', {})
        replacements[word] = entry

        metadata = data.setdefault('metadata', {})
        metadata['total_words'] = len(replacements)
        metadata['last_updated'] = datetime.utcnow().isoformat() + 'Z'
```

```
        with open(file_path, 'w', encoding='utf-8') as f:
            json.dump(data, f, ensure_ascii=False, indent=2)

        # Очищаємо кеш для групи, щоб наступний запит підхопив оновлення
        if group_type in self._dictionaries_cache:
            del self._dictionaries_cache[group_type]
        if group_type in self._word_forms_index_cache:
            del self._word_forms_index_cache[group_type]
        # Кеш пошуку може містити застарілі "не знайдено" / результати —
очищаємо.
        self._lookup_cache.clear()

        return True

    def generate_and_store_llm_replacement(self, word: str, word_type: str,
context: Optional[str] = None) -> Tuple[Optional[Dict], Optional[str]]:
        """
        Викликає Gemma для простих категорій і додає результат у словник.
        Повертає (entry, group_type) якщо успішно.
        """
        if not self.enable_llm_by_key:
            return None, None

        entry = self.generate_llm_replacement_entry(word, word_type,
context=context)
        if not entry:
            return None, None

        group_type = self.get_group_for_type(word_type)
        saved = self.save_replacement_entry(word, entry, group_type)
        if not saved:
            return None, None

        return entry, group_type

    def list_pending_entries(self) -> List[Dict]:
        """Повертає список записів, які ще не перевірені (isVerify=False)"""
        pending: List[Dict] = []
        for group_type in self.GROUP_FILES.keys():
            dictionary = self.load_group_dictionary(group_type)
            for word_key, entry in dictionary.items():
                metadata = entry.get('metadata', {})
                if metadata.get('isVerify', True) is False:
                    pending.append({
                        'word': word_key,
                        'group': group_type,
                        'replacements': entry.get('priority_replacements', []),
                        'metadata': metadata
                    })
        return pending

    def set_entry_verification(self, word: str, group_type: str, is_verified:
bool, *, replacements: Optional[List[str]] = None) -> bool:
        """Позначає запис перевіреним / неперевіреним"""
        filename = self._get_primary_filename(group_type)
        if not filename:
            return False

        file_path = self.base_dir / filename
        if not file_path.exists():
```

```
            return False

        with open(file_path, 'r', encoding='utf-8') as f:
            data = json.load(f)

        replacements_map = data.get('replacements', {})
        if word not in replacements_map:
            return False

        entry = replacements_map[word]
        if replacements is not None:
            cleaned = []
            seen = set()
            for item in replacements:
                if not isinstance(item, str):
                    continue
                candidate = item.strip()
                if not candidate:
                    continue
                normalized = self._clean_word(candidate)
                if not normalized or normalized in seen:
                    continue
                seen.add(normalized)
                cleaned.append(candidate)

            if cleaned:
                metadata = entry.get('metadata', {})
                context = metadata.get('original_category') or
self._group_to_category(group_type)
                entry['priority_replacements'] = [
                    {
                        'replacement': synonym,
                        'priority': idx,
                        'context': context,
                        'formality': 'neutral',
                    }
                    for idx, synonym in enumerate(cleaned, start=1)
                ]
        metadata = entry.setdefault('metadata', {})
        metadata['isVerify'] = bool(is_verified)
        if is_verified:
            metadata['verified_at'] = datetime.utcnow().isoformat() + 'Z'
        else:
            metadata.pop('verified_at', None)

        with open(file_path, 'w', encoding='utf-8') as f:
            json.dump(data, f, ensure_ascii=False, indent=2)

        if group_type in self._dictionaries_cache:
            del self._dictionaries_cache[group_type]
        self._lookup_cache.clear()

        if is_verified:
            self.add_source_entry(word, group_type)

        return True

    def delete_entry(self, word: str, group_type: str) -> bool:
        """Видаляє запис із словника (наприклад, після відхилення)"""
        filename = self._get_primary_filename(group_type)
```

```
        if not filename:
            return False

        file_path = self.base_dir / filename
        if not file_path.exists():
            return False

        with open(file_path, 'r', encoding='utf-8') as f:
            data = json.load(f)

        replacements = data.get('replacements', {})
        if word not in replacements:
            return False

        del replacements[word]

        metadata = data.setdefault('metadata', {})
        metadata['total_words'] = len(replacements)
        metadata['last_updated'] = datetime.utcnow().isoformat() + 'Z'

        with open(file_path, 'w', encoding='utf-8') as f:
            json.dump(data, f, ensure_ascii=False, indent=2)

        if group_type in self._dictionaries_cache:
            del self._dictionaries_cache[group_type]

        return True
```

alhoritm/utils/schema.py

```
from __future__ import annotations

from typing import Any, Dict, Type, TypeVar

from pydantic import BaseModel, ValidationError

from utils.errors import ApiError

T = TypeVar("T", bound=BaseModel)


def parse_json(payload: Any, schema: Type[T]) -> T:
    if not isinstance(payload, dict):
        raise ApiError("Request body must be a JSON object.", status_code=400)
    try:
        return schema.model_validate(payload)
    except ValidationError as exc:
        raise ApiError("Schema validation failed.", status_code=422,
details=exc.errors())
```

alhoritm/utils/syntax_improvement_service.py

```
"""
Rule-based syntactic simplification service for iterative improvements.
Focuses on sentence structure: splitting long sentences, isolating clauses,
and normalising tricky constructions without relying on external LLMs.
"""
from __future__ import annotations

from dataclasses import dataclass
import re
```

```
import statistics
from typing import Dict, List, Optional, Sequence, Tuple

import stanza

TARGET_SENTENCE_LENGTH = 25
TARGET_SENTENCE_VARIATION = 12

def _safe_int(value):
    try:
        return int(str(value).split("-")[0])
    except (ValueError, TypeError, AttributeError):
        return 0


@dataclass
class SimplificationResult:
    """Stores information about a single simplification iteration."""

    strategy: str
    original_sentence: str
    transformed_sentence: str
    score_before: float
    score_after: float
    sentence_word_count_before: int
    new_sentences: List[str]


class SyntaxImprovementService:
    """Performs deterministic syntactic simplifications in iterations."""

    CONNECTOR_PATTERNS: Sequence[Tuple[str, str]] = (
        ("тому що", "reason"),
        ("оскільки", "reason"),
        ("через те що", "reason"),
        ("бо", "reason"),
        ("адже", "reason"),
        ("хоча", "contrast"),
        ("проте", "contrast"),
        ("але", "contrast"),
        ("якщо", "condition"),
        ("коли", "time"),
        ("доки", "time"),
        ("поки", "time"),
        ("аби", "condition"),
        ("щоб", "purpose"),
        ("наче", "comparison"),
        ("ніби", "comparison"),
        ("мов", "comparison"),
        ("немов", "comparison"),
    )

    GERUND_PATTERN = re.compile(
        r",\s*(?P<phrase>[^,]*(?:ючи|ючися|ачи|аючи|вши|ши)\b[^,]*)",
re.IGNORECASE
    )
    RELATIVE_PATTERN = re.compile(

r",\s*(?P<rel>який|яка|яке|які|котрий|котра|котре|котрі|що|де|куди|чому)\b",
        re.IGNORECASE,
```

```
    )
    INTRODUCTORY_PATTERNS = (
        "враховуючи",
        "зважаючи на",
        "беручи до уваги",
        "незважаючи на",
        "попри",
        "з огляду на",
        "по-перше",
        "по-друге",
        "по-третє",
        "у разі",
        "у випадку",
        "під час",
        "після",
        "перед",
        "з метою",
        "при цьому",
    )
    PASSIVE_PARTICIPLE_SUFFIXES = (
        "но",
        "то",
        "ено",
        "єно",
        "ано",
        "ене",
        "ений",
        "ена",
        "ені",
        "ене",
        "йно",
    )
    PASSIVE_SUFFIX_REPLACEMENTS = (
        ("єно", "ли"),
        ("ено", "ли"),
        ("ано", "ли"),
        ("то", "ли"),
        ("но", "ли"),
    )

    def __init__(
        self,
        nlp: Optional[stanza.Pipeline] = None,
        target_sentence_length: int = TARGET_SENTENCE_LENGTH,
        target_sentence_variation: int = TARGET_SENTENCE_VARIATION,
    ) -> None:
        self.nlp = nlp or stanza.Pipeline(
            lang="uk", processors="tokenize,mwt,pos,lemma,depparse",
tokenize_no_ssplit=True
        )
        self.target_sentence_length = target_sentence_length
        self.target_sentence_variation = target_sentence_variation

    def improve_sentence_structure(
        self,
        text: str,
        max_iterations: int = 3,
        min_improvement: float = 0.01,
    ) -> Dict:
        """Iteratively splits complex sentences and tracks score
```

```
improvements."""
        current_text = text
        iterations: List[SimplificationResult] = []

        doc, sentences_data = self._analyze_text(current_text)
        sentence_lengths = [item["word_count"] for item in sentences_data if
item["word_count"] > 0]
        initial_score = self._sentence_structure_score(sentence_lengths)
        current_score = initial_score

        for _ in range(max_iterations):
            candidate = self._select_sentence(sentences_data)
            if not candidate:
                break

            transformation = self._transform_sentence(candidate)
            if not transformation:
                break

            updated_text = self._apply_transformation(
                current_text, candidate["start_char"], candidate["end_char"],
transformation["new_sentences"]
            )

            new_doc, new_sentences_data = self._analyze_text(updated_text)
            new_lengths = [item["word_count"] for item in new_sentences_data if
item["word_count"] > 0]
            new_score = self._sentence_structure_score(new_lengths)
            improvement = new_score - current_score

            if improvement < min_improvement:
                # Stop if no tangible gain
                break

            iterations.append(
                SimplificationResult(
                    strategy=transformation["strategy"],
                    original_sentence=candidate["text"],
                    transformed_sentence="
".join(transformation["new_sentences"]),
                    score_before=round(current_score, 3),
                    score_after=round(new_score, 3),
                    sentence_word_count_before=candidate["word_count"],
                    new_sentences=transformation["new_sentences"],
                )
            )

            current_text = updated_text
            current_score = new_score
            doc, sentences_data = new_doc, new_sentences_data

        return {
            "initial_sentence_structure_score": round(initial_score, 3),
            "final_sentence_structure_score": round(current_score, 3),
            "iterations": [iteration.__dict__ for iteration in iterations],
            "improved_text": current_text,
            "applied_iterations": len(iterations),
            "status": "ok" if iterations else "unchanged",
        }
```

```
    def _analyze_text(self, text: str) -> Tuple[stanza.Document, List[Dict]]:
        doc = self.nlp(text)
        sentences_data: List[Dict] = []
        for sentence in doc.sentences:
            tokens = sentence.tokens
            if not tokens:
                continue
            start_char = tokens[0].start_char or 0
            end_char = tokens[-1].end_char or start_char
            text = sentence.text.strip()
            lowered = text.lower()
            has_intro = any(lowered.startswith(phrase) for phrase in
self.INTRODUCTORY_PATTERNS)
            passive_candidate = self._find_passive_verb(sentence)
            sentences_data.append(
                {
                    "sentence": sentence,
                    "text": text,
                    "word_count": len(tokens),
                    "comma_count": text.count(","),
                    "semicolon_count": text.count(";"),
                    "has_intro": has_intro,
                    "has_passive": passive_candidate is not None,
                    "start_char": start_char,
                    "end_char": end_char,
                }
            )
        return doc, sentences_data

    def _sentence_structure_score(self, lengths: List[int]) -> float:
        if not lengths:
            return 1.0
        avg_length = sum(lengths) / len(lengths)
        length_stdev = statistics.pstdev(lengths) if len(lengths) > 1 else 0.0

        length_penalty = 0.7 * min(avg_length / self.target_sentence_length,
1.0)
        length_penalty += 0.3 * min(length_stdev /
self.target_sentence_variation, 1.0)

        return max(0.0, 1 - length_penalty)

    def _select_sentence(self, sentences: List[Dict]) -> Optional[Dict]:
        if not sentences:
            return None
        for sentence in sentences:
            if sentence.get("has_passive"):
                return sentence
        def score(item: Dict) -> float:
            return (
                item["word_count"]
                + item.get("comma_count", 0) * 2
                + item.get("semicolon_count", 0) * 4
                + (3 if item.get("has_intro") else 0)
            )
        candidate = max(sentences, key=score)
        if (
            candidate["word_count"] < self.target_sentence_length
            and candidate["comma_count"] < 3
            and candidate.get("semicolon_count", 0) == 0
```

```
            and not candidate.get("has_intro")
        ):
            return None
        return candidate

    def _transform_sentence(self, sentence_data: Dict) -> Optional[Dict]:
        transformers = (
            self._split_by_connector,
            self._split_introductory_phrase,
            self._split_relative_clause,
            self._split_semicolon_blocks,
            self._split_by_gerund,
            self._split_by_comma,
            self._convert_passive_to_active,
        )
        for transformer in transformers:
            transformed = transformer(sentence_data)
            if transformed:
                return transformed
        return None

    def _split_by_connector(self, sentence_data: Dict) -> Optional[Dict]:
        text = sentence_data["text"]
        for connector, label in self.CONNECTOR_PATTERNS:
            pattern = re.compile(rf",\s+{connector}\b", re.IGNORECASE)
            match = pattern.search(text)
            if not match:
                continue
            before = text[: match.start()]
            after = text[match.end():]
            if len(before.split()) < 3 or len(after.split()) < 3:
                continue

            first_sentence = self._ensure_sentence(before)
            second_sentence = self._build_second_sentence(
                after, sentence_data["sentence"], connector, label
            )
            if not first_sentence or not second_sentence:
                continue

            return {
                "strategy": f"connector_split:{connector}",
                "new_sentences": [first_sentence, second_sentence],
            }
        return None

    def _split_introductory_phrase(self, sentence_data: Dict) -> Optional[Dict]:
        text = sentence_data["text"]
        lowered = text.lower()
        for phrase in self.INTRODUCTORY_PATTERNS:
            if lowered.startswith(phrase):
                delimiter_index = None
                for delimiter in [",", "—", "-"]:
                    idx = text.find(delimiter)
                    if idx != -1:
                        delimiter_index = idx
                        break
                if delimiter_index is None:
                    continue
                intro = text[:delimiter_index].strip()
```

```
                remainder = self._normalize_fragment(text[delimiter_index + 1
:])
                if len(intro.split()) < 2 or len(remainder.split()) < 3:
                    continue
                first_sentence = self._ensure_sentence(intro)
                second_sentence = self._ensure_sentence(remainder)
                if not first_sentence or not second_sentence:
                    continue
                return {
                    "strategy": f"intro_split:{phrase}",
                    "new_sentences": [first_sentence, second_sentence],
                }
        return None

    def _split_by_comma(self, sentence_data: Dict) -> Optional[Dict]:
        text = sentence_data["text"]
        commas = [match.start() for match in re.finditer(",", text)]
        if not commas:
            return None
        midpoint = len(text) / 2
        comma_index = min(commas, key=lambda idx: abs(idx - midpoint))
        before = text[:comma_index]
        after = text[comma_index + 1 :]
        if len(before.split()) < 3 or len(after.split()) < 3:
            return None
        first_sentence = self._ensure_sentence(before)
        after_clean = self._normalize_fragment(after.strip())
        second_sentence = self._ensure_sentence(after_clean)
        if not first_sentence or not second_sentence:
            return None
        return {
            "strategy": "comma_split",
            "new_sentences": [first_sentence, second_sentence],
        }

    def _split_by_gerund(self, sentence_data: Dict) -> Optional[Dict]:
        text = sentence_data["text"]
        match = self.GERUND_PATTERN.search(text)
        if not match:
            return None
        phrase = match.group("phrase").strip()
        before = text[: match.start()].rstrip()
        after = text[match.end() :].lstrip()
        if not phrase or len(phrase.split()) < 2:
            return None
        first_sentence = self._ensure_sentence(" ".join([before,
after]).strip())
        second_sentence = self._ensure_sentence(phrase)
        if not first_sentence or not second_sentence:
            return None
        return {
            "strategy": "gerund_isolation",
            "new_sentences": [first_sentence, second_sentence],
        }

    def _split_relative_clause(self, sentence_data: Dict) -> Optional[Dict]:
        text = sentence_data["text"]
        match = self.RELATIVE_PATTERN.search(text)
        if not match:
            return None
```

```
        before = text[: match.start()]
        after = self._normalize_fragment(text[match.end():])
        if len(before.split()) < 2 or len(after.split()) < 2:
            return None
        first_sentence = self._ensure_sentence(before)
        second_sentence = self._ensure_sentence(f"Це {after}")
        if not first_sentence or not second_sentence:
            return None
        return {
            "strategy": f"relative_split:{match.group('rel').lower()}",
            "new_sentences": [first_sentence, second_sentence],
        }

    def _split_semicolon_blocks(self, sentence_data: Dict) -> Optional[Dict]:
        text = sentence_data["text"]
        if ";" not in text:
            return None
        parts = [self._normalize_fragment(part) for part in text.split(";")]
        parts = [part for part in parts if len(part.split()) >= 3]
        if len(parts) < 2:
            return None
        new_sentences = []
        for part in parts:
            sentence = self._ensure_sentence(part)
            if not sentence:
                return None
            new_sentences.append(sentence)
        return {
            "strategy": "semicolon_split",
            "new_sentences": new_sentences,
        }

    def _convert_passive_to_active(self, sentence_data: Dict) -> Optional[Dict]:
        sentence = sentence_data["sentence"]
        if sentence is None:
            return None
        passive_verb = self._find_passive_verb(sentence)
        if not passive_verb:
            return None
        patient = self._find_patient(sentence, passive_verb)
        if not patient:
            return None
        agent = self._find_agent(sentence, passive_verb)
        agent_phrase, agent_ids = self._extract_phrase(sentence, agent) if agent
else ("Фахівці", set())
        patient_phrase, patient_ids = self._extract_phrase(sentence, patient)
        if not patient_phrase:
            return None
        lemma = (passive_verb.lemma or passive_verb.text).lower()
        active_verb = self._to_active_form(passive_verb.text.lower(), lemma)
        if not active_verb:
            return None
        verb_id = _safe_int(passive_verb.id)
        used_ids = set(agent_ids) | set(patient_ids) | {verb_id}
        used_ids |= {
            _safe_int(word.id)
            for word in sentence.words
            if (word.deprel or "").startswith("aux") and _safe_int(word.head) ==
verb_id
        }
```

```
        rest_tokens = [
            word.text
            for word in sentence.words
            if _safe_int(word.id) not in used_ids and word.upos != "PUNCT"
        ]
        rest_text = self._clean_spacing(" ".join(rest_tokens))
        if agent and self._has_case(agent, "Ins") and len(agent_phrase.split())
== 1:
            agent_phrase = self._guess_nominative(agent)
        agent_phrase = self._capitalize_phrase(agent_phrase)
        new_sentence = f"{agent_phrase} {active_verb} {patient_phrase}"
        if rest_text:
            new_sentence = f"{new_sentence} {rest_text}"
        new_sentence = self._ensure_sentence(new_sentence)
        return {
            "strategy": "passive_to_active",
            "new_sentences": [new_sentence] if new_sentence else [],
        }

    def _build_second_sentence(
        self,
        fragment: str,
        sentence: stanza.models.common.doc.Sentence,
        connector: str,
        label: str,
    ) -> Optional[str]:
        fragment = self._normalize_fragment(fragment)
        if not fragment:
            return None
        sentence_text = self._ensure_sentence(fragment)
        if label == "reason":
            sentence_text = f"Причина:
{sentence_text[0].lower()}{sentence_text[1:]}" if sentence_text else
sentence_text
        return sentence_text

    def _ensure_sentence(self, fragment: str) -> Optional[str]:
        fragment = fragment.strip()
        if not fragment:
            return None
        if fragment[-1] not in ".!?…":
            fragment += "."
        return fragment[0].upper() + fragment[1:]

    def _normalize_fragment(self, fragment: str) -> str:
        fragment = fragment.strip()
        fragment = re.sub(r"^(і|й|та|але|проте|тож|а|причому)\b\s*", "",
fragment, flags=re.IGNORECASE)
        return fragment.strip()

    def _apply_transformation(
        self,
        text: str,
        start_char: int,
        end_char: int,
        new_sentences: List[str],
    ) -> str:
        replacement = " ".join(new_sentences)
        return f"{text[:start_char]}{replacement}{text[end_char:]}"
```

```
    @staticmethod
    def _extract_subject(sentence: stanza.models.common.doc.Sentence) ->
Optional[str]:
        for word in sentence.words:
            deprel = (word.deprel or "").split(":")[0]
            if deprel == "nsubj":
                return word.text.capitalize()
        return None

    def _to_active_form(self, surface: str, lemma: str) -> Optional[str]:
        lemma = lemma.strip()
        if lemma.endswith("ести"):
            return (lemma[:-4] + "ели").strip()
        if lemma.endswith("ти"):
            return (lemma[:-2] + "ли").strip()
        for suffix, replacement in self.PASSIVE_SUFFIX_REPLACEMENTS:
            if surface.endswith(suffix):
                base = surface[: -len(suffix)]
                return (base + replacement).strip()
        return None

    @staticmethod
    def _has_case(word, case: str) -> bool:
        if not word.feats:
            return False
        return f"Case={case}" in word.feats

    def _guess_nominative(self, word) -> str:
        base = (word.lemma or word.text or "").strip()
        if not base:
            return ""
        feats = word.feats or ""
        if "Number=Plur" in feats:
            if base.endswith("к"):
                base = base[:-1] + "ки"
            elif base.endswith("а"):
                base = base[:-1] + "и"
            elif base.endswith("я"):
                base = base[:-1] + "ї"
            else:
                base = base + "и"
        return base.capitalize()

    def _find_passive_verb(self, sentence: stanza.models.common.doc.Sentence):
        for word in sentence.words:
            feats = word.feats or ""
            if word.upos in {"VERB", "AUX"} and "Voice=Pass" in feats:
                return word
            if word.upos == "VERB" and
self._looks_passive_participle(word.text.lower()):
                return word
        return None

    def _find_patient(self, sentence: stanza.models.common.doc.Sentence, verb) -
> Optional[stanza.models.common.doc.Word]:
        verb_id = _safe_int(verb.id)
        for word in sentence.words:
            head = _safe_int(word.head)
            if head != verb_id:
                continue
```

```
            deprel = (word.deprel or "").split(":")[0]
            if deprel in {"obj", "nsubj", "nsubjpass", "nsubj:pass"}:
                return word
        return None

    def _find_agent(self, sentence: stanza.models.common.doc.Sentence, verb):
        verb_id = _safe_int(verb.id)
        for word in sentence.words:
            head = _safe_int(word.head)
            if head != verb_id:
                continue
            deprel = (word.deprel or "")
            if deprel.startswith("obl"):
                if self._has_case(word, "Ins") or
word.text.lower().startswith(("за", "від", "через", "під")):
                    return word
        return None

    def _extract_phrase(self, sentence, word):
        if not word:
            return "", set()
        ids = self._collect_subtree_ids(sentence, _safe_int(word.id))
        words = [w for w in sentence.words if _safe_int(w.id) in ids and w.upos
!= "PUNCT"]
        words.sort(key=lambda w: _safe_int(w.id))
        phrase = " ".join(w.text for w in words)
        phrase = self._clean_spacing(phrase)
        return phrase, ids

    def _collect_subtree_ids(self, sentence, root_id):
        ids = set()
        stack = [root_id]
        while stack:
            current = stack.pop()
            ids.add(current)
            for word in sentence.words:
                head = _safe_int(word.head)
                wid = _safe_int(word.id)
                if head == current and wid not in ids:
                    stack.append(wid)
        return ids

    @staticmethod
    def _clean_spacing(text: str) -> str:
        text = re.sub(r"\s+([,.!?;:])", r"\1", text)
        text = re.sub(r"\(\s+", "(", text)
        text = re.sub(r"\s+\)", ")", text)
        return text.strip()

    @staticmethod
    def _capitalize_phrase(text: str) -> str:
        text = text.strip()
        if not text:
            return text
        return text[0].upper() + text[1:]

    def _looks_passive_participle(self, text: str) -> bool:
        return text.endswith(self.PASSIVE_PARTICIPLE_SUFFIXES)
```

alhoritm/utils/term_explainer.py

```
from __future__ import annotations

import json
import re
from pathlib import Path
from typing import Dict, Optional


_TOKEN_RE = re.compile(
    r"[a-zA-Zґ]+(?:['’`ʼ-][a-zA-Zґ]+)*|[а-яіїєґ]+(?:['’`ʼ][а-яіїєґ]+)*",
    re.IGNORECASE,
)
_DEFINITIONS_CACHE: Optional[Dict[str, str]] = None


def _default_definitions_path() -> Path:
    return Path(__file__).resolve().parents[1] / "data" / "replacements" /
"term_definitions.json"


def load_term_definitions(path: Optional[Path] = None) -> Dict[str, str]:
    global _DEFINITIONS_CACHE
    if _DEFINITIONS_CACHE is not None:
        return _DEFINITIONS_CACHE

    p = path or _default_definitions_path()
    try:
        raw = json.loads(p.read_text(encoding="utf-8"))
    except Exception:
        raw = {}

    out: Dict[str, str] = {}
    if isinstance(raw, dict):
        for k, v in raw.items():
            if not isinstance(k, str) or not isinstance(v, str):
                continue
            key = k.strip().lower()
            val = v.strip()
            if key and val:
                out[key] = val

    _DEFINITIONS_CACHE = out
    return out


def _tokens(text: str) -> list[str]:
    if not isinstance(text, str) or not text.strip():
        return []
    return [m.group(0).lower() for m in _TOKEN_RE.finditer(text)]


def _normalize_token_for_lookup(token: str) -> list[str]:
    """
    Cheap, offline normalizations for common Ukrainian inflections in technical
terms.
    Returns candidates in priority order (first match wins).
    """
    tok = token.strip().lower()
    if not tok:
```

```
        return []

    candidates = [tok]

    # Very common: "...ції" (Gen/Loc) -> "...ція" (Nom), e.g. "класифікації" ->
"класифікація".
    if tok.endswith("ції") and len(tok) > 4:
        candidates.append(tok[:-3] + "ція")

    # Common: "...ості" -> "...ість", e.g. "доступності" -> "доступність".
    if tok.endswith("ості") and len(tok) > 4:
        candidates.append(tok[:-3] + "ість")

    # De-dup while preserving order.
    out: list[str] = []
    seen = set()
    for c in candidates:
        if c not in seen:
            out.append(c)
            seen.add(c)
    return out


def explain_term(
    surface: str,
    *,
    lemma: Optional[str] = None,
    definitions: Optional[Dict[str, str]] = None,
) -> Optional[str]:
    """
    Returns a short in-text explanation for a detected term, or None if we can't
explain it offline.

    Intended output format: "<surface> (тобто <simple explanation>)"
    """
    if not isinstance(surface, str) or not surface.strip():
        return None

    defs = definitions if definitions is not None else load_term_definitions()
    if not defs:
        return None

    # Tokenize surface first. For multi-word terms we must prefer the surface
representation:
    # the lemma we receive might belong to only one token (often not the head).
    surface_tokens = _tokens(surface)
    tokens = surface_tokens

    # For single-word terms, lemma-first matching can help with inflections.
    if len(surface_tokens) <= 1 and isinstance(lemma, str) and lemma.strip():
        lemma_tokens = _tokens(lemma)
        if lemma_tokens:
            tokens = lemma_tokens
    elif len(surface_tokens) > 1 and isinstance(lemma, str) and lemma.strip():
        # For multi-word terms, keep surface tokens, but also consider lemma as
an extra lookup candidate.
        lemma_tokens = _tokens(lemma)
        if lemma_tokens and len(lemma_tokens) == 1:
            tokens = lemma_tokens + surface_tokens
```

```
    if not tokens:
        return None

    # Prefer the head word (right-most token) for multi-word terms.
    definition = None
    for tok in reversed(tokens):
        for candidate in _normalize_token_for_lookup(tok):
            definition = defs.get(candidate)
            if definition:
                break
        if definition:
            break

    if not definition:
        return None

    return f"{surface} (тобто {definition})"
```

alhoritm/utils/textNornalize.py

```
import os
import PyPDF2
import docx2txt

LATEST_FILE_PATH = "/tmp/latest_uploaded_file"
LATEST_FILE_PATH_FULL = None

def save_latest_file(file_bytes, original_filename):
    global LATEST_FILE_PATH_FULL
    ext = os.path.splitext(original_filename)[-1]
    LATEST_FILE_PATH_FULL = os.path.join(f"{LATEST_FILE_PATH}{ext}")

    with open(LATEST_FILE_PATH_FULL, 'wb') as f:
        f.write(file_bytes)
    return f"file://{LATEST_FILE_PATH_FULL}"

def extract_text_from_file(filepath):
    if filepath.startswith('file://'):
        filepath = filepath.replace('file://', '', 1)
    ext = filepath.rsplit('.', 1)[-1].lower() if '.' in filepath else 'txt'

    if ext == 'txt':
        with open(filepath, 'r', encoding='utf-8', errors='ignore') as f:
            return f.read()
    elif ext == 'pdf':
        with open(filepath, 'rb') as f:
            reader = PyPDF2.PdfReader(f)
            text = ""
            for page in reader.pages:
                page_text = page.extract_text()
                if page_text:
                    text += page_text
            if not text:
                print("Warning: No extractable text found in PDF. It may be
scanned or image-based.")
            return text
    elif ext in ['doc', 'docx']:
        text = docx2txt.process(filepath)
        if not text:
            print("Warning: No extractable text found in DOCX.")
```

```
        return text
    else:
        return ""

def get_latest_file_text():
    if LATEST_FILE_PATH_FULL is None:
        return None

    candidate = f"{LATEST_FILE_PATH_FULL}"

    if os.path.exists(candidate):
        return extract_text_from_file(candidate)
    return None
```

alhoritm/utils/utilResponse.py

```
from flask import jsonify
from utils.textNornalize import get_latest_file_text

def check_response(request):
    data = request.get_json()
    if not data:
        return jsonify({"error": "No 'text' field provided."}), 400

    # Якщо явно вказано використовувати останній файл
    if data.get("isLatestText", False):
        text = get_latest_file_text()
    else:
        # Інакше використовуємо текст з запиту
        if "text" not in data:
            return jsonify({"error": "No 'text' field provided."}), 400
        text = data["text"]
    return text
```

alhoritm/repositories/base.py

```
from __future__ import annotations

import json
from pathlib import Path
from typing import Any


class JsonRepository:
    def __init__(self, base_dir: Path):
        self.base_dir = base_dir

    def read_json(self, relative_path: str, default: Any):
        path = self.base_dir / relative_path
        if not path.exists():
            return default
        try:
            return json.loads(path.read_text(encoding="utf-8"))
        except json.JSONDecodeError:
            return default

    def write_json(self, relative_path: str, data: Any, *, indent: int = 2) ->
None:
        path = self.base_dir / relative_path
        path.parent.mkdir(parents=True, exist_ok=True)
        path.write_text(json.dumps(data, ensure_ascii=False, indent=indent),
```

```
encoding="utf-8")
```

react-frontend-app/src/main.tsx

```
import { StrictMode } from 'react';
import { createRoot } from 'react-dom/client';
import App from './App';
import './i18n';

const container = document.getElementById('root');
if (!container) {
  throw new Error('Root element not found');
}

createRoot(container).render(
  <StrictMode>
    <App />
  </StrictMode>
);
```

react-frontend-app/src/App.tsx

```
import { useCallback, useEffect, useRef, useState } from 'react';
import { useTranslation } from 'react-i18next';
import styles from './App.module.css';
import { Editor, Footer, Header, LeftAside, RightAside, ErrorBoundary,
ErrorNotificationContainer } from './components';
import { batchAnalyze } from './services/batch-analysis';
import { getTransformText } from './services/text-transformation';
import { AppProviders, useTextAnalysis, useCurrentFile, useAppActions,
useUISettings, useDebugSettings } from './context/AppProviders';
import { BatchAnalyzeResponse, DetectedWords, DetectionType } from
'./types/api';
import { ExpertValidationModal } from './components/expert-
validation/ExpertValidationModal';

function hashStringDjb2(input: string): string {
  let hash = 5381;
  for (let i = 0; i < input.length; i += 1) {
    hash = ((hash << 5) + hash) ^ input.charCodeAt(i);
  }
  // Unsigned 32-bit
  return (hash >>> 0).toString(16);
}

function isAbortError(error: unknown): boolean {
  const anyError = error as { name?: string; code?: string; message?: string } |
null;
  if (!anyError) return false;
  if (anyError.name === 'AbortError' || anyError.code === 'ABORT_ERR') return
true;
  return anyError.message === 'signal is aborted without reason';
}

function AppContent(): JSX.Element {
  const { t } = useTranslation();
  const headerBoxRef = useRef<HTMLDivElement>(null);
  const { score, detected, wordTypes, accessibilityData, loading, error } =
useTextAnalysis();
  const currentFile = useCurrentFile();
  const { settings } = useUISettings();
```

```
  const { options: debugOptions, setOption: setDebugOption } =
useDebugSettings();
  const { setAnalysisLoading, setAnalysisSuccess, setAnalysisError,
setAccessibilityData, setCurrentFile } = useAppActions();
  const analysisCacheRef = useRef<Map<string, BatchAnalyzeResponse>>(new Map());
  const [expertModalOpen, setExpertModalOpen] = useState<boolean>(false);

  // Use content from uploaded file or fallback to default text
  const text = currentFile.content || '';
  const textToEditor = getTransformText(text);

  const applyAnalysisResult = useCallback((result: BatchAnalyzeResponse): void
=> {
    // Для підсвітки
    const detectedWords: DetectedWords = {};
    type DetectionKey = Exclude<keyof BatchAnalyzeResponse, 'scope' | 'cai'>;
    const detectionTypes: DetectionKey[] = ['profanity', 'abbr', 'slang',
'terms', 'obsolete', 'complex', 'rare', 'flesch'];

    for (const type of detectionTypes) {
      const block = result[type];
      if (block?.words) {
        detectedWords[type] = block.words;
      } else {
        detectedWords[type] = [];
      }
    }

    // Для FeedbackPanel: формуємо flat-об'єкт з found_count
    const scoreFlat: Record<string, number> = {};
    for (const type of detectionTypes) {
      const block = result[type];
      scoreFlat[type] = block?.found_count || 0;
    }

    const accessibilityResult = result.cai ?? null;

    const syntacticFactors =
accessibilityResult?.components?.syntactic?.factors;
    if (syntacticFactors) {
      scoreFlat.sentence_structure = syntacticFactors.sentence_structure_score
?? 0;
      scoreFlat.clause_simplicity = syntacticFactors.clause_score ?? 0;
      scoreFlat.word_order = syntacticFactors.svo_ratio ?? 0;
    }

    setAnalysisSuccess(scoreFlat, detectedWords);
    setAccessibilityData(accessibilityResult);
  }, [setAnalysisSuccess, setAccessibilityData]);

  const getProfanityScoreHandler = useCallback(async (customTextOrAnalysis?:
string | BatchAnalyzeResponse): Promise<void> => {
    const providedAnalysis =
      typeof customTextOrAnalysis === 'object' && customTextOrAnalysis !== null
&& 'scope' in customTextOrAnalysis
        ? (customTextOrAnalysis as BatchAnalyzeResponse)
        : null;

    if (!providedAnalysis && loading) return; // Prevent multiple concurrent
requests (only for network calls)
```

```
    try {
      setAnalysisLoading(true);
      const result = providedAnalysis ?? (await batchAnalyze(
        { text: (customTextOrAnalysis as string) || textToEditor, include_cai:
true },
        { abortKey: 'analyze:global', dedupe: true }
      ));

      applyAnalysisResult(result);

    } catch (error) {
      if (isAbortError(error)) {
        return;
      }
      const errorMessage = error instanceof Error ? error.message : 'Unknown
error occurred';
      setAnalysisError(errorMessage);
      console.error('Error in text analysis:', error);
    }
  }, [textToEditor, loading, setAnalysisLoading, setAnalysisError,
applyAnalysisResult]);

  useEffect(() => {
    const text = textToEditor;
    if (!text?.trim()) return;

    const mode = settings.analysisScopeMode;
    const { pageScopeStart, pageScopeEnd } = currentFile;

    const canUsePage = mode === 'page' && pageScopeEnd > pageScopeStart;
    const scope = canUsePage ? { start: pageScopeStart, end: pageScopeEnd } :
undefined;
    if (mode === 'page' && !scope) return; // wait for Editor to provide valid
page scope

    const cacheKey =
      mode === 'page' && scope
        ?
`page:${scope.start}:${scope.end}:${hashStringDjb2(text.slice(scope.start,
scope.end))}`
        : `document:0:${text.length}:${hashStringDjb2(text)}`;

    const cached = analysisCacheRef.current.get(cacheKey);
    if (cached) {
      applyAnalysisResult(cached);
      return;
    }

    void (async () => {
      try {
        setAnalysisLoading(true);
        const result = await batchAnalyze(
          { text, scope, include_cai: true },
          { abortKey: 'analyze:auto', dedupe: true }
        );
        analysisCacheRef.current.set(cacheKey, result);
        // Evict oldest entry to cap memory usage during long sessions.
        if (analysisCacheRef.current.size > 50) {
          const oldestKey = analysisCacheRef.current.keys().next().value;
```

```
          if (oldestKey !== undefined)
analysisCacheRef.current.delete(oldestKey);
        }
        applyAnalysisResult(result);
      } catch (error) {
        if (isAbortError(error)) {
          return;
        }
        const errorMessage = error instanceof Error ? error.message : 'Unknown
error occurred';
        setAnalysisError(errorMessage);
        console.error('Error in auto analysis:', error);
      }
    })();
  }, [
    textToEditor,
    settings.analysisScopeMode,
    currentFile.pageScopeStart,
    currentFile.pageScopeEnd,
    setAnalysisLoading,
    setAnalysisError,
    applyAnalysisResult,
  ]);

  const handleEditorPageContextChange = useCallback((ctx: { pageIndex: number;
totalPages: number; scope: { start: number; end: number } }) => {
    setCurrentFile({
      page: ctx.pageIndex + 1,
      pageIndex: ctx.pageIndex,
      pageScopeStart: ctx.scope.start,
      pageScopeEnd: ctx.scope.end,
    });
  }, [setCurrentFile]);

  const handleEditorCursorContextChange = useCallback((ctx: { cursorIndex:
number | null }) => {
    setCurrentFile({
      cursorIndex: ctx.cursorIndex,
    });
  }, [setCurrentFile]);

  useEffect(() => {
    const headerBox = headerBoxRef.current;
    if (!headerBox) return;

    const setVar = (): void => {
      const h = headerBox.getBoundingClientRect().height;
      document.documentElement.style.setProperty('--app-header-height',
`${Math.round(h)}px`);
    };

    setVar();

    let ro: ResizeObserver | null = null;
    if (typeof ResizeObserver !== 'undefined') {
      ro = new ResizeObserver(setVar);
      ro.observe(headerBox);
    } else {
      window.addEventListener('resize', setVar);
    }
```

```
    return () => {
      if (ro) {
        ro.disconnect();
      } else {
        window.removeEventListener('resize', setVar);
      }
    };
  }, []);

  useEffect(() => {
    if (typeof window === 'undefined' || typeof document === 'undefined')
return;
    const media = window.matchMedia ? window.matchMedia('(prefers-contrast:
high)') : null;
    const apply = (): void => {
      const prefersHigh = media?.matches ?? false;
      const enabled = Boolean(settings.highContrast) || prefersHigh;
      if (enabled) {
        document.documentElement.setAttribute('data-contrast', 'high');
      } else {
        document.documentElement.removeAttribute('data-contrast');
      }
    };
    apply();
    if (media?.addEventListener) {
      media.addEventListener('change', apply);
      return () => media.removeEventListener('change', apply);
    }
    if (media?.addListener) {
      media.addListener(apply);
      return () => media.removeListener(apply);
    }
    return undefined;
  }, [settings.highContrast]);

  useEffect(() => {
    if (debugOptions.expertValidation) {
      setExpertModalOpen(true);
    }
  }, [debugOptions.expertValidation]);

  return (
    <>
      <a href="#main-content" className={styles.skipLink}>
        Пропустити до тексту
      </a>
      <ErrorBoundary>
        <div ref={headerBoxRef}>
          <Header t={t} />
        </div>
      </ErrorBoundary>
      <div className={styles.container}>
        <div className={styles.mainBlock}>
          <ErrorBoundary>
            <LeftAside
              t={t}
              fileName={currentFile.name}
              page={currentFile.page}
              simplificationLevel={currentFile.simplificationLevel}
```

```
              dispatch={setCurrentFile}
              currentFile={currentFile}
            />
          </ErrorBoundary>
          <main id="main-content">
            <ErrorBoundary>
              <Editor
                initialText={textToEditor}
                detected={detected}
                wordTypes={wordTypes}
                activeRange={currentFile.ttsActiveRange}
                onPageContextChange={handleEditorPageContextChange}
                onCursorContextChange={handleEditorCursorContextChange}
              />
            </ErrorBoundary>
          </main>
          <ErrorBoundary>
            <RightAside t={t} score={score}
accessibilityData={accessibilityData} />
          </ErrorBoundary>
        </div>
      </div>
      <ErrorBoundary>
        <Footer t={t} />
      </ErrorBoundary>
      {error && (
        <div className={styles.errorNotification}>
          Error: {error}
        </div>
      )}
      <ErrorNotificationContainer />
      <ExpertValidationModal
        isOpen={debugOptions.expertValidation && expertModalOpen}
        onClose={() => {
          setExpertModalOpen(false);
          setDebugOption('expertValidation', false);
        }}
      />
    </>
  );
}

function App(): JSX.Element {
  return (
    <AppProviders>
      <AppContent />
    </AppProviders>
  );
}

export default App;
```

react-frontend-app/src/i18n.ts

```
import i18n from "i18next";
import { initReactI18next } from "react-i18next";
import HttpBackend from 'i18next-http-backend';

const getInitialLanguage = (): string => {
    if (typeof window === 'undefined') return 'ua';
```

```
    try {
        const raw = window.localStorage.getItem('ui_settings');
        if (!raw) return 'ua';
        const parsed = JSON.parse(raw);
        return typeof parsed?.language === 'string' ? parsed.language : 'ua';
    } catch {
        return 'ua';
    }
};

const toHtmlLang = (lng: string): string => {
    if (!lng) return 'uk';
    if (lng.toLowerCase() === 'ua') return 'uk';
    return lng;
};

i18n
    .use(HttpBackend)
    .use(initReactI18next) // passes i18n down to react-i18next
    .init({
        fallbackLng: 'en', // Default language
        lng: getInitialLanguage(), // Initial language
        debug: true, // Turn off in production
        backend: {
            loadPath: '/locales/{{lng}}/translations.json',
        },
        interpolation: {
            escapeValue: false // react already safes from xss
        }
    })
    .then(() => {
        if (typeof document !== 'undefined') {
            document.documentElement.lang = toHtmlLang(i18n.language);
        }
    });

i18n.on('languageChanged', (lng: string) => {
    if (typeof document !== 'undefined') {
        document.documentElement.lang = toHtmlLang(lng);
    }
});

export default i18n;
```

react-frontend-app/src/context/AppProviders.tsx

```
import { ReactNode } from 'react';
import { AppStateProvider } from './AppStateContext';
import { UISettingsProvider } from './UISettingsContext';
import { ErrorBoundary } from '../components';

interface AppProvidersProps {
  children: ReactNode;
}

/**
 * Main application providers wrapper
 * Combines all context providers with proper error boundaries
 */
export function AppProviders({ children }: AppProvidersProps) {
```

```
  return (
    <ErrorBoundary>
      <UISettingsProvider>
        <ErrorBoundary>
          <AppStateProvider>
            {children}
          </AppStateProvider>
        </ErrorBoundary>
      </UISettingsProvider>
    </ErrorBoundary>
  );
}

// Re-export all hooks for convenience
export {
  // App State hooks
  useAppState,
  useTextAnalysis,
  useFileUpload,
  useCurrentFile,
  useAppActions,
  useFileUploadActions,
} from './AppStateContext';

export {
  // UI Settings hooks
  useUISettings,
  useWordMode,
  useHighlightMode,
  useSimplifyScopeMode,
  useAnalysisScopeMode,
  useTheme,
  useAppLanguage,
  useDebugSettings,
  useSettings, // Backward compatibility
} from './UISettingsContext';

// Legacy export for backward compatibility
export { UISettingsProvider as SettingsProvider } from './UISettingsContext';
```

react-frontend-app/src/context/AppStateContext.tsx

```
import { createContext, useContext, useReducer, ReactNode, useMemo, type
Dispatch } from 'react';
import { DetectedWords, WordType, AccessibilityData, LastSimplifyResult,
SimplifyRunStatus } from '../types/api';

// Application State Types
interface AppState {
  // Text Analysis State
  score: Record<string, number>;
  detected: DetectedWords;
  wordTypes: Array<WordType | null> | null;
  accessibilityData: AccessibilityData | null;

  // UI State
  loading: {
    textAnalysis: boolean;
    fileUpload: boolean;
  };
```

```
  // Error State
  errors: {
    textAnalysis: string | null;
    fileUpload: string | null;
  };

  // File State
  currentFile: {
    name: string;
    page: number;
    pageIndex: number;
    pageScopeStart: number;
    pageScopeEnd: number;
    cursorIndex: number | null;
    ttsActiveRange: { start: number; end: number } | null;
    lastSimplify: LastSimplifyResult | null;
    lastSimplifyStatus: SimplifyRunStatus;
    simplificationLevel: number;
    content: string;
    originalFileName?: string;
    fileSize?: number;
    fileType?: string;
    uploadedAt?: number;
  };
}

// Action Types
type AppAction =
  | { type: 'SET_ANALYSIS_LOADING'; payload: boolean }
  | { type: 'SET_ANALYSIS_SUCCESS'; payload: { score: Record<string, number>;
detected: DetectedWords } }
  | { type: 'SET_ANALYSIS_ERROR'; payload: string }
  | { type: 'SET_WORD_TYPES'; payload: Array<WordType | null> | null }
  | { type: 'SET_ACCESSIBILITY_DATA'; payload: AccessibilityData | null }
  | { type: 'SET_FILE_UPLOAD_LOADING'; payload: boolean }
  | { type: 'SET_FILE_UPLOAD_SUCCESS'; payload: { content: string; fileName:
string; fileSize: number; fileType: string } }
  | { type: 'SET_FILE_UPLOAD_ERROR'; payload: string | null }
  | { type: 'SET_CURRENT_FILE'; payload: Partial<AppState['currentFile']> }
  | { type: 'RESET_ERRORS' };

// Initial State
const initialState: AppState = {
  score: {},
  detected: {},
  wordTypes: null,
  accessibilityData: null,
  loading: {
    textAnalysis: false,
    fileUpload: false,
  },
  errors: {
    textAnalysis: null,
    fileUpload: null,
  },
  currentFile: {
    name: 'example.txt',
    page: 1,
    pageIndex: 0,
```

```
    pageScopeStart: 0,
    pageScopeEnd: 0,
    cursorIndex: null,
    ttsActiveRange: null,
    lastSimplify: null,
    lastSimplifyStatus: 'idle',
    simplificationLevel: 1,
    content: '',
  },
};

// Reducer
function appStateReducer(state: AppState, action: AppAction): AppState {
  switch (action.type) {
    case 'SET_ANALYSIS_LOADING':
      return {
        ...state,
        loading: { ...state.loading, textAnalysis: action.payload },
        errors: { ...state.errors, textAnalysis: null },
      };

    case 'SET_ANALYSIS_SUCCESS':
      return {
        ...state,
        score: action.payload.score,
        detected: action.payload.detected,
        loading: { ...state.loading, textAnalysis: false },
        errors: { ...state.errors, textAnalysis: null },
      };

    case 'SET_ANALYSIS_ERROR':
      return {
        ...state,
        loading: { ...state.loading, textAnalysis: false },
        errors: { ...state.errors, textAnalysis: action.payload },
      };

    case 'SET_WORD_TYPES':
      return {
        ...state,
        wordTypes: action.payload,
      };

    case 'SET_ACCESSIBILITY_DATA':
      return {
        ...state,
        accessibilityData: action.payload,
      };

    case 'SET_FILE_UPLOAD_LOADING':
      return {
        ...state,
        loading: { ...state.loading, fileUpload: action.payload },
        errors: { ...state.errors, fileUpload: null },
      };

    case 'SET_FILE_UPLOAD_SUCCESS':
      const newState = {
        ...state,
        loading: { ...state.loading, fileUpload: false },
```

```
        errors: { ...state.errors, fileUpload: null },
        currentFile: {
          ...state.currentFile,
          content: action.payload.content,
          originalFileName: action.payload.fileName,
          fileSize: action.payload.fileSize,
          fileType: action.payload.fileType,
          uploadedAt: Date.now(),
          name: action.payload.fileName,
          page: 1, // Reset to first page when new file is loaded
          pageIndex: 0,
          pageScopeStart: 0,
          pageScopeEnd: 0,
          cursorIndex: null,
          ttsActiveRange: null,
          lastSimplify: null,
        },
      };
      return newState;

    case 'SET_FILE_UPLOAD_ERROR':
      return {
        ...state,
        loading: { ...state.loading, fileUpload: false },
        errors: { ...state.errors, fileUpload: action.payload },
      };

    case 'SET_CURRENT_FILE':
      return {
        ...state,
        currentFile: { ...state.currentFile, ...action.payload },
      };

    case 'RESET_ERRORS':
      return {
        ...state,
        errors: {
          textAnalysis: null,
          fileUpload: null,
        },
      };

    default:
      return state;
  }
}

// Context
interface AppStateContextValue {
  state: AppState;
  dispatch: Dispatch<AppAction>;
}

const AppStateContext = createContext<AppStateContextValue |
undefined>(undefined);

// Provider
interface AppStateProviderProps {
  children: ReactNode;
}
```

```
export function AppStateProvider({ children }: AppStateProviderProps) {
  const [state, dispatch] = useReducer(appStateReducer, initialState);

  const value = useMemo(() => ({ state, dispatch }), [state]);

  return (
    <AppStateContext.Provider value={value}>
      {children}
    </AppStateContext.Provider>
  );
}

// Hook
export function useAppState(): AppStateContextValue {
  const context = useContext(AppStateContext);
  if (context === undefined) {
    throw new Error('useAppState must be used within an AppStateProvider');
  }
  return context;
}

// Selector Hooks for Performance
export function useTextAnalysis() {
  const { state } = useAppState();
  return useMemo(() => ({
    score: state.score,
    detected: state.detected,
    wordTypes: state.wordTypes,
    accessibilityData: state.accessibilityData,
    loading: state.loading.textAnalysis,
    error: state.errors.textAnalysis,
  }), [state.score, state.detected, state.wordTypes, state.accessibilityData,
state.loading.textAnalysis, state.errors.textAnalysis]);
}

export function useFileUpload() {
  const { state } = useAppState();
  return useMemo(() => ({
    loading: state.loading.fileUpload,
    error: state.errors.fileUpload,
  }), [state.loading.fileUpload, state.errors.fileUpload]);
}

export function useCurrentFile() {
  const { state } = useAppState();
  return useMemo(() => state.currentFile, [state.currentFile]);
}

// Action Creators
export function useAppActions() {
  const { dispatch } = useAppState();

  return useMemo(() => ({
    setAnalysisLoading: (loading: boolean) =>
      dispatch({ type: 'SET_ANALYSIS_LOADING', payload: loading }),

    setAnalysisSuccess: (score: Record<string, number>, detected: DetectedWords)
=>
      dispatch({ type: 'SET_ANALYSIS_SUCCESS', payload: { score, detected } }),
```

```
    setAnalysisError: (error: string) =>
      dispatch({ type: 'SET_ANALYSIS_ERROR', payload: error }),

    setWordTypes: (wordTypes: Array<WordType | null> | null) =>
      dispatch({ type: 'SET_WORD_TYPES', payload: wordTypes }),

    setAccessibilityData: (data: AccessibilityData | null) =>
      dispatch({ type: 'SET_ACCESSIBILITY_DATA', payload: data }),

    setFileUploadLoading: (loading: boolean) =>
      dispatch({ type: 'SET_FILE_UPLOAD_LOADING', payload: loading }),

    setFileUploadSuccess: (content: string, fileName: string, fileSize: number,
fileType: string) => {
      dispatch({ type: 'SET_FILE_UPLOAD_SUCCESS', payload: { content, fileName,
fileSize, fileType } });
    },

    setFileUploadError: (error: string | null) =>
      dispatch({ type: 'SET_FILE_UPLOAD_ERROR', payload: error }),

    setCurrentFile: (file: Partial<AppState['currentFile']>) =>
      dispatch({ type: 'SET_CURRENT_FILE', payload: file }),

    resetErrors: () =>
      dispatch({ type: 'RESET_ERRORS' }),
  }), [dispatch]);
}

// File upload specific actions
export function useFileUploadActions() {
  const { dispatch } = useAppState();

  return useMemo(() => ({
    setFileUploadLoading: (loading: boolean) =>
      dispatch({ type: 'SET_FILE_UPLOAD_LOADING', payload: loading }),

    setFileUploadSuccess: (content: string, fileName: string, fileSize: number,
fileType: string) =>
      dispatch({ type: 'SET_FILE_UPLOAD_SUCCESS', payload: { content, fileName,
fileSize, fileType } }),

    setFileUploadError: (error: string | null) =>
      dispatch({ type: 'SET_FILE_UPLOAD_ERROR', payload: error }),
  }), [dispatch]);
}
```

react-frontend-app/src/context/UISettingsContext.tsx

```
import { createContext, useContext, useState, useEffect, ReactNode, useMemo,
useCallback } from 'react';

// UI Settings State
interface DebugSettings {
  widgets: boolean;
  camera: boolean;
  llm: boolean;
  expertValidation: boolean;
}
```

```
interface UISettings {
  wordMode: boolean;
  highlightMode: boolean;
  ttsEnabled: boolean;
  highContrast: boolean;
  simplifyScopeMode: 'document' | 'page' | 'cursor';
  analysisScopeMode: 'document' | 'page';
  theme: 'light' | 'dark';
  language: string;
  debug: DebugSettings;
}

// Context Value Type
interface UISettingsContextValue {
  settings: UISettings;
  updateWordMode: (enabled: boolean) => void;
  updateHighlightMode: (enabled: boolean) => void;
  updateTtsEnabled: (enabled: boolean) => void;
  updateHighContrast: (enabled: boolean) => void;
  updateSimplifyScopeMode: (mode: UISettings['simplifyScopeMode']) => void;
  updateAnalysisScopeMode: (mode: UISettings['analysisScopeMode']) => void;
  updateTheme: (theme: 'light' | 'dark') => void;
  updateLanguage: (language: string) => void;
  toggleWordMode: () => void;
  toggleHighlightMode: () => void;
  toggleTtsEnabled: () => void;
  toggleHighContrast: () => void;
  updateDebugOption: (key: keyof DebugSettings, enabled: boolean) => void;
  toggleDebugOption: (key: keyof DebugSettings) => void;
  resetSettings: () => void;
}

const STORAGE_KEY = 'ui_settings';

// Default Settings
const defaultSettings: UISettings = {
  wordMode: false,
  highlightMode: false,
  ttsEnabled: false,
  highContrast: false,
  simplifyScopeMode: 'page',
  analysisScopeMode: 'page',
  theme: 'light',
  language: 'ua',
  debug: {
    widgets: false,
    camera: false,
    llm: false,
    expertValidation: false,
  },
};

// Utility Functions
function loadSettingsFromStorage(): UISettings {
  try {
    const saved = localStorage.getItem(STORAGE_KEY);
    if (saved) {
      const parsed = JSON.parse(saved);
      const simplifyScopeMode =
```

```
        parsed.simplifyScopeMode === 'document' || parsed.simplifyScopeMode ===
'page' || parsed.simplifyScopeMode === 'cursor'
          ? parsed.simplifyScopeMode
          : defaultSettings.simplifyScopeMode;
      const analysisScopeMode =
        parsed.analysisScopeMode === 'document' || parsed.analysisScopeMode ===
'page'
          ? parsed.analysisScopeMode
          : defaultSettings.analysisScopeMode;
      return {
        wordMode: typeof parsed.wordMode === 'boolean' ? parsed.wordMode :
defaultSettings.wordMode,
        highlightMode: typeof parsed.highlightMode === 'boolean' ?
parsed.highlightMode : defaultSettings.highlightMode,
        ttsEnabled: typeof parsed.ttsEnabled === 'boolean' ? parsed.ttsEnabled :
defaultSettings.ttsEnabled,
        highContrast: typeof parsed.highContrast === 'boolean' ?
parsed.highContrast : defaultSettings.highContrast,
        simplifyScopeMode,
        analysisScopeMode,
        theme: parsed.theme === 'light' || parsed.theme === 'dark' ?
parsed.theme : defaultSettings.theme,
        language: typeof parsed.language === 'string' ? parsed.language :
defaultSettings.language,
        debug: {
          widgets: Boolean(parsed?.debug?.widgets),
          camera: Boolean(parsed?.debug?.camera),
          llm: Boolean(parsed?.debug?.llm),
          expertValidation: Boolean(parsed?.debug?.expertValidation),
        },
      };
    }
  } catch (error) {
    console.warn('Failed to load UI settings from localStorage:', error);
  }
  return defaultSettings;
}

function saveSettingsToStorage(settings: UISettings): void {
  try {
    localStorage.setItem(STORAGE_KEY, JSON.stringify(settings));
  } catch (error) {
    console.warn('Failed to save UI settings to localStorage:', error);
  }
}

// Context
const UISettingsContext = createContext<UISettingsContextValue |
undefined>(undefined);

// Provider
interface UISettingsProviderProps {
  children: ReactNode;
}

export function UISettingsProvider({ children }: UISettingsProviderProps) {
  const [settings, setSettings] = useState<UISettings>(loadSettingsFromStorage);

  // Save to localStorage when settings change
  useEffect(() => {
```

```
    saveSettingsToStorage(settings);
  }, [settings]);

  // Memoized update functions
  const updateWordMode = useCallback((enabled: boolean) => {
    setSettings(prev => ({ ...prev, wordMode: enabled }));
  }, []);

  const updateHighlightMode = useCallback((enabled: boolean) => {
    setSettings(prev => ({ ...prev, highlightMode: enabled }));
  }, []);

  const updateTtsEnabled = useCallback((enabled: boolean) => {
    setSettings(prev => ({ ...prev, ttsEnabled: enabled }));
  }, []);

  const updateHighContrast = useCallback((enabled: boolean) => {
    setSettings(prev => ({ ...prev, highContrast: enabled }));
  }, []);

  const updateSimplifyScopeMode = useCallback((mode:
UISettings['simplifyScopeMode']) => {
    setSettings(prev => ({ ...prev, simplifyScopeMode: mode }));
  }, []);

  const updateAnalysisScopeMode = useCallback((mode:
UISettings['analysisScopeMode']) => {
    setSettings(prev => ({ ...prev, analysisScopeMode: mode }));
  }, []);

  const updateTheme = useCallback((theme: 'light' | 'dark') => {
    setSettings(prev => ({ ...prev, theme }));
  }, []);

  const updateLanguage = useCallback((language: string) => {
    setSettings(prev => ({ ...prev, language }));
  }, []);

  const toggleWordMode = useCallback(() => {
    setSettings(prev => ({ ...prev, wordMode: !prev.wordMode }));
  }, []);

  const toggleHighlightMode = useCallback(() => {
    setSettings(prev => ({ ...prev, highlightMode: !prev.highlightMode }));
  }, []);

  const toggleTtsEnabled = useCallback(() => {
    setSettings(prev => ({ ...prev, ttsEnabled: !prev.ttsEnabled }));
  }, []);

  const toggleHighContrast = useCallback(() => {
    setSettings(prev => ({ ...prev, highContrast: !prev.highContrast }));
  }, []);

  const updateDebugOption = useCallback((key: keyof DebugSettings, enabled:
boolean) => {
    setSettings(prev => ({
      ...prev,
      debug: {
        ...prev.debug,
```

```
        [key]: enabled,
      },
    }));
  }, []);

  const toggleDebugOption = useCallback((key: keyof DebugSettings) => {
    setSettings(prev => ({
      ...prev,
      debug: {
        ...prev.debug,
        [key]: !prev.debug[key],
      },
    }));
  }, []);

  const resetSettings = useCallback(() => {
    setSettings(defaultSettings);
  }, []);

  // Memoized context value
  const value = useMemo((): UISettingsContextValue => ({
    settings,
    updateWordMode,
    updateHighlightMode,
    updateTtsEnabled,
    updateHighContrast,
    updateSimplifyScopeMode,
    updateAnalysisScopeMode,
    updateTheme,
    updateLanguage,
    toggleWordMode,
    toggleHighlightMode,
    toggleTtsEnabled,
    toggleHighContrast,
    updateDebugOption,
    toggleDebugOption,
    resetSettings,
  }), [settings, updateWordMode, updateHighlightMode, updateTtsEnabled,
updateHighContrast, updateSimplifyScopeMode, updateAnalysisScopeMode,
updateTheme, updateLanguage, toggleWordMode, toggleHighlightMode,
toggleTtsEnabled, toggleHighContrast, updateDebugOption, toggleDebugOption,
resetSettings]);

  return (
    <UISettingsContext.Provider value={value}>
      {children}
    </UISettingsContext.Provider>
  );
}

// Main Hook
export function useUISettings(): UISettingsContextValue {
  const context = useContext(UISettingsContext);
  if (context === undefined) {
    throw new Error('useUISettings must be used within a UISettingsProvider');
  }
  return context;
}

// Selector Hooks for Performance
```

```
export function useWordMode() {
  const { settings, updateWordMode, toggleWordMode } = useUISettings();
  return useMemo(() => ({
    enabled: settings.wordMode,
    toggle: toggleWordMode,
    set: updateWordMode,
  }), [settings.wordMode, toggleWordMode, updateWordMode]);
}

export function useHighlightMode() {
  const { settings, updateHighlightMode, toggleHighlightMode } =
useUISettings();
  return useMemo(() => ({
    enabled: settings.highlightMode,
    toggle: toggleHighlightMode,
    set: updateHighlightMode,
  }), [settings.highlightMode, toggleHighlightMode, updateHighlightMode]);
}

export function useSimplifyScopeMode() {
  const { settings, updateSimplifyScopeMode } = useUISettings();
  return useMemo(() => ({
    current: settings.simplifyScopeMode,
    set: updateSimplifyScopeMode,
  }), [settings.simplifyScopeMode, updateSimplifyScopeMode]);
}

export function useAnalysisScopeMode() {
  const { settings, updateAnalysisScopeMode } = useUISettings();
  return useMemo(() => ({
    current: settings.analysisScopeMode,
    set: updateAnalysisScopeMode,
  }), [settings.analysisScopeMode, updateAnalysisScopeMode]);
}

export function useTheme() {
  const { settings, updateTheme } = useUISettings();
  return useMemo(() => ({
    current: settings.theme,
    set: updateTheme,
    isDark: settings.theme === 'dark',
  }), [settings.theme, updateTheme]);
}

export function useAppLanguage() {
  const { settings, updateLanguage } = useUISettings();
  return useMemo(() => ({
    current: settings.language,
    set: updateLanguage,
  }), [settings.language, updateLanguage]);
}

export function useDebugSettings() {
  const { settings, updateDebugOption, toggleDebugOption } = useUISettings();
  return useMemo(() => ({
    options: settings.debug,
    setOption: updateDebugOption,
    toggleOption: toggleDebugOption,
  }), [settings.debug, updateDebugOption, toggleDebugOption]);
}
```

```
// Backward compatibility hook
export function useSettings() {
  const { settings, updateWordMode, updateHighlightMode, updateTtsEnabled,
updateHighContrast, updateSimplifyScopeMode, updateAnalysisScopeMode } =
useUISettings();
  return useMemo(() => ({
    wordMode: settings.wordMode,
    setWordMode: updateWordMode,
    highlightMode: settings.highlightMode,
    setHighlightMode: updateHighlightMode,
    ttsEnabled: settings.ttsEnabled,
    setTtsEnabled: updateTtsEnabled,
    highContrast: settings.highContrast,
    setHighContrast: updateHighContrast,
    simplifyScopeMode: settings.simplifyScopeMode,
    setSimplifyScopeMode: updateSimplifyScopeMode,
    analysisScopeMode: settings.analysisScopeMode,
    setAnalysisScopeMode: updateAnalysisScopeMode,
  }), [settings.wordMode, settings.highlightMode, settings.ttsEnabled,
settings.highContrast, settings.simplifyScopeMode, settings.analysisScopeMode,
updateWordMode, updateHighlightMode, updateTtsEnabled, updateHighContrast,
updateSimplifyScopeMode, updateAnalysisScopeMode]);
}
```

react-frontend-app/src/services/api-client.ts

```
import { baseTextAPIPath } from '../enums/global.config';
import { ApiError } from '../types/api';
import { errorHandler } from './error-handler';

// Request/Response Interceptor Types
interface RequestInterceptor {
  onRequest?: (config: RequestConfig) => RequestConfig | Promise<RequestConfig>;
  onRequestError?: (error: Error) => Error | Promise<Error>;
}

interface ResponseInterceptor {
  onResponse?: (response: Response) => Response | Promise<Response>;
  onResponseError?: (error: Error) => Error | Promise<Error>;
}

interface RequestConfig extends RequestInit {
  url?: string;
  timeout?: number;
  retries?: number;
  retryDelay?: number;
  skipCache?: boolean;
  dedupe?: boolean;
  abortKey?: string;
}

interface CacheEntry {
  data: any;
  timestamp: number;
  ttl: number;
}

// API Client Class
export class ApiClient {
```

```
private baseURL: string;
private requestInterceptors: RequestInterceptor[] = [];
private responseInterceptors: ResponseInterceptor[] = [];
private cache = new Map<string, CacheEntry>();
private pendingRequests = new Map<string, Promise<any>>();
private abortControllers = new Map<string, AbortController>();

constructor(baseURL: string = baseTextAPIPath) {
  this.baseURL = baseURL;
  this.setupDefaultInterceptors();
}

private normalizeHeaders(init?: HeadersInit): Record<string, string> {
  if (!init) return {};
  if (init instanceof Headers) {
    const obj: Record<string, string> = {};
    init.forEach((value, key) => {
      obj[key] = value;
    });
    return obj;
  }
  if (Array.isArray(init)) {
    return Object.fromEntries(init);
  }
  return { ...init };
}

// Setup default interceptors
private setupDefaultInterceptors(): void {
  // Request logging interceptor
  this.addRequestInterceptor({
    onRequest: (config) => {
      return config;
    },
    onRequestError: (error) => {
      return error;
    },
  });

  // Response logging interceptor
  this.addResponseInterceptor({
    onResponse: (response) => {
      return response;
    },
    onResponseError: (error) => {
      return error;
    },
  });

  // Error handling interceptor with enhanced error handler
  this.addResponseInterceptor({
    onResponseError: async (error) => {
      // Use enhanced error handler for better categorization and logging
      const enhancedError = errorHandler.handleError(error, {
        timestamp: Date.now(),
        userAgent: navigator.userAgent,
        url: window.location.href,
      });

      // Return original error to maintain compatibility
```

```
        return error;
      },
    });
  }

  // Interceptor management
  addRequestInterceptor(interceptor: RequestInterceptor): number {
    return this.requestInterceptors.push(interceptor) - 1;
  }

  addResponseInterceptor(interceptor: ResponseInterceptor): number {
    return this.responseInterceptors.push(interceptor) - 1;
  }

  removeRequestInterceptor(id: number): void {
    this.requestInterceptors.splice(id, 1);
  }

  removeResponseInterceptor(id: number): void {
    this.responseInterceptors.splice(id, 1);
  }

  // Cache management
  private getCacheKey(url: string, config: RequestConfig): string {
    const { method = 'GET', body } = config;
    const bodyKey =
      typeof body === 'string'
        ? body
        : body
          ? '[non-string-body]'
          : '';
    return `${method}:${url}:${bodyKey}`;
  }

  private getCachedData(key: string): any | null {
    const entry = this.cache.get(key);
    if (!entry) return null;

    const now = Date.now();
    if (now > entry.timestamp + entry.ttl) {
      this.cache.delete(key);
      return null;
    }

    return entry.data;
  }

  private setCachedData(key: string, data: any, ttl: number = 5 * 60 * 1000):
void {
    this.cache.set(key, {
      data,
      timestamp: Date.now(),
      ttl,
    });
  }

  clearCache(): void {
    this.cache.clear();
  }
```

```
  // Request deduplication
  private getDuplicateRequest(key: string): Promise<any> | null {
    return this.pendingRequests.get(key) || null;
  }

  private setPendingRequest(key: string, promise: Promise<any>): void {
    this.pendingRequests.set(key, promise);
    promise.finally(() => {
      this.pendingRequests.delete(key);
    });
  }

  abort(abortKey: string): void {
    const controller = this.abortControllers.get(abortKey);
    if (!controller) return;
    controller.abort();
    this.abortControllers.delete(abortKey);
  }

  // Retry logic with exponential backoff
  private async executeWithRetry<T>(
    operation: () => Promise<T>,
    retries: number = 3,
    delay: number = 1000
  ): Promise<T> {
    try {
      return await operation();
    } catch (error) {
      if (retries > 0 && this.shouldRetry(error)) {
        await this.delay(delay);
        return this.executeWithRetry(operation, retries - 1, delay * 2);
      }
      throw error;
    }
  }

  private shouldRetry(error: any): boolean {
    // Retry on network errors or specific HTTP status codes
    if (error instanceof TypeError && error.message.includes('fetch')) {
      return true; // Network error
    }
    if (error instanceof ApiError) {
      return error.status === 408 || error.status === 429 || error.status >=
500;
    }
    return false;
  }

  private delay(ms: number): Promise<void> {
    return new Promise(resolve => setTimeout(resolve, ms));
  }

  // Timeout implementation
  private createTimeoutPromise(timeout: number): Promise<never> {
    return new Promise((_, reject) => {
      setTimeout(() => {
        reject(new ApiError({
          message: `Request timeout after ${timeout}ms`,
          code: 'TIMEOUT',
        }));
```

```
      }, timeout);
    });
  }

  // Main request method
  async request<T = any>(endpoint: string, config: RequestConfig = {}):
Promise<T> {
    const url = endpoint.startsWith('http') ? endpoint :
`${this.baseURL}/${endpoint}`;

    const normalizedHeaders = this.normalizeHeaders(config.headers);
    const hasContentTypeHeader = Object.keys(normalizedHeaders).some((k) =>
k.toLowerCase() === 'content-type');
    const shouldDefaultJsonContentType =
      !hasContentTypeHeader &&
      (typeof config.body === 'string' || typeof config.body === 'undefined');

    const headers: Record<string, string> = {
      ...(shouldDefaultJsonContentType ? { 'Content-Type': 'application/json' }
: {}),
      ...normalizedHeaders,
    };

    // If we're sending FormData, ensure we don't force a JSON Content-Type.
    if (typeof FormData !== 'undefined' && config.body instanceof FormData) {
      for (const key of Object.keys(headers)) {
        if (key.toLowerCase() === 'content-type') delete headers[key];
      }
    }

    const finalConfig: RequestConfig = {
      method: 'GET',
      timeout: 30000,  // Збільшуємо до 30 секунд для AI операцій
      retries: 3,
      retryDelay: 1000,
      ...config,
      url,
      headers,
    };

    // Apply request interceptors
    let processedConfig = finalConfig;
    for (const interceptor of this.requestInterceptors) {
      if (interceptor.onRequest) {
        try {
          processedConfig = await interceptor.onRequest(processedConfig);
        } catch (error) {
          if (interceptor.onRequestError) {
            throw await interceptor.onRequestError(error as Error);
          }
          throw error;
        }
      }
    }

    const cacheKey = this.getCacheKey(url, processedConfig);
    const method = (processedConfig.method || 'GET').toUpperCase();

    const isJsonBody = typeof processedConfig.body === 'string' &&
processedConfig.body.trim().length > 0;
```

```
    const dedupeEligible =
      processedConfig.dedupe !== false &&
      (method === 'GET' || method === 'POST') &&
      (method === 'GET' || isJsonBody);

    // Cache for GET requests (unchanged behavior)
    if (!processedConfig.skipCache && method === 'GET') {
      const cachedData = this.getCachedData(cacheKey);
      if (cachedData) return cachedData;
    }

    // Deduplicate in-flight requests (GET + JSON POST)
    if (dedupeEligible) {
      const duplicateRequest = this.getDuplicateRequest(cacheKey);
      if (duplicateRequest) return duplicateRequest;
    }

    // Create the request promise
    const requestPromise = this.executeWithRetry(async () => {
      const controller = new AbortController();
      const upstreamSignal = processedConfig.signal;

      if (upstreamSignal) {
        if (upstreamSignal.aborted) controller.abort();
        else upstreamSignal.addEventListener('abort', () => controller.abort(),
{ once: true });
      }

      if (processedConfig.abortKey) {
        // Abort previous request with the same key.
        this.abort(processedConfig.abortKey);
        this.abortControllers.set(processedConfig.abortKey, controller);
      }

      const fetchPromise = fetch(url, { ...processedConfig, signal:
controller.signal });

      let response: Response;
      if (processedConfig.timeout) {
        response = await Promise.race([
          fetchPromise,
          this.createTimeoutPromise(processedConfig.timeout),
        ]);
      } else {
        response = await fetchPromise;
      }

      // Apply response interceptors
      let processedResponse = response;
      for (const interceptor of this.responseInterceptors) {
        if (interceptor.onResponse) {
          try {
            processedResponse = await interceptor.onResponse(processedResponse);
          } catch (error) {
            if (interceptor.onResponseError) {
              throw await interceptor.onResponseError(error as Error);
            }
            throw error;
          }
        }
```

```
      }

      if (!processedResponse.ok) {
        const error = new ApiError({
          message: `HTTP error! status: ${processedResponse.status}`,
          status: processedResponse.status,
        });

        // Apply error interceptors
        for (const interceptor of this.responseInterceptors) {
          if (interceptor.onResponseError) {
            await interceptor.onResponseError(error);
          }
        }

        throw error;
      }

      const data = await processedResponse.json();

      // Cache successful GET requests
      if (!processedConfig.skipCache && method === 'GET') {
        this.setCachedData(cacheKey, data);
      }

      return data;
    }, processedConfig.retries, processedConfig.retryDelay);

    // Store pending request for deduplication
    if (dedupeEligible) {
      this.setPendingRequest(cacheKey, requestPromise);
    }

    if (processedConfig.abortKey) {
      const abortKey = processedConfig.abortKey;
      const controller = this.abortControllers.get(abortKey);
      requestPromise.finally(() => {
        if (controller && this.abortControllers.get(abortKey) === controller) {
          this.abortControllers.delete(abortKey);
        }
      });
    }

    return requestPromise;
  }

  // Convenience methods
  async get<T = any>(endpoint: string, config?: Omit<RequestConfig, 'method'>):
Promise<T> {
    return this.request<T>(endpoint, { ...config, method: 'GET' });
  }

  async post<T = any>(endpoint: string, data?: any, config?: Omit<RequestConfig,
'method' | 'body'>): Promise<T> {
    return this.request<T>(endpoint, {
      ...config,
      method: 'POST',
      body: data ? JSON.stringify(data) : undefined,
    });
  }
```

```
  async put<T = any>(endpoint: string, data?: any, config?: Omit<RequestConfig,
'method' | 'body'>): Promise<T> {
    return this.request<T>(endpoint, {
      ...config,
      method: 'PUT',
      body: data ? JSON.stringify(data) : undefined,
    });
  }

  async delete<T = any>(endpoint: string, config?: Omit<RequestConfig,
'method'>): Promise<T> {
    return this.request<T>(endpoint, { ...config, method: 'DELETE' });
  }

  // File upload method
  async uploadFile<T = any>(endpoint: string, file: File, config?:
Omit<RequestConfig, 'method' | 'body' | 'headers'>): Promise<T> {
    const formData = new FormData();
    formData.append('file', file);

    return this.request<T>(endpoint, {
      ...config,
      method: 'POST',
      body: formData,
      headers: {
        // Remove Content-Type to let browser set it with boundary
        ...config?.headers,
      },
      skipCache: true, // Don't cache file uploads
    });
  }
}

// Create default client instance
export const apiClient = new ApiClient();

// Export convenience functions that use the default client
export const api = {
  get: <T = any>(endpoint: string, config?: Omit<RequestConfig, 'method'>) =>
    apiClient.get<T>(endpoint, config),

  post: <T = any>(endpoint: string, data?: any, config?: Omit<RequestConfig,
'method' | 'body'>) =>
    apiClient.post<T>(endpoint, data, config),

  put: <T = any>(endpoint: string, data?: any, config?: Omit<RequestConfig,
'method' | 'body'>) =>
    apiClient.put<T>(endpoint, data, config),

  delete: <T = any>(endpoint: string, config?: Omit<RequestConfig, 'method'>) =>
    apiClient.delete<T>(endpoint, config),

  uploadFile: <T = any>(endpoint: string, file: File, config?:
Omit<RequestConfig, 'method' | 'body' | 'headers'>) =>
    apiClient.uploadFile<T>(endpoint, file, config),

  abort: (abortKey: string) => apiClient.abort(abortKey),

  clearCache: () => apiClient.clearCache(),
```

```
  addRequestInterceptor: (interceptor: RequestInterceptor) =>
    apiClient.addRequestInterceptor(interceptor),

  addResponseInterceptor: (interceptor: ResponseInterceptor) =>
    apiClient.addResponseInterceptor(interceptor),
};
```

react-frontend-app/src/services/batch-analysis.ts

```
import { api } from './api-client';
import { BatchAnalyzeResponse } from '../types/api';

export async function batchAnalyze(params: {
  text: string;
  scope?: { start: number; end: number };
  include_cai?: boolean;
  confidence_threshold?: number;
}, config?: { signal?: AbortSignal; abortKey?: string; dedupe?: boolean }):
Promise<BatchAnalyzeResponse> {
  return api.post<BatchAnalyzeResponse>('analyze', params, {
    timeout: 120000,
    ...config,
  });
}
```

react-frontend-app/src/services/error-handler.ts

```
import { ApiError } from '../types/api';

// Error categories
export enum ErrorCategory {
  NETWORK = 'NETWORK',
  API = 'API',
  VALIDATION = 'VALIDATION',
  PERMISSION = 'PERMISSION',
  UNKNOWN = 'UNKNOWN',
  TIMEOUT = 'TIMEOUT',
}

// Error severity levels
export enum ErrorSeverity {
  LOW = 'LOW',
  MEDIUM = 'MEDIUM',
  HIGH = 'HIGH',
  CRITICAL = 'CRITICAL',
}

// Enhanced error interface
export interface EnhancedError extends Error {
  category: ErrorCategory;
  severity: ErrorSeverity;
  timestamp: number;
  context?: Record<string, any>;
  userMessage?: string;
  code?: string;
  retryable?: boolean;
}

// Error handler class
export class ErrorHandler {
```

```
  private static instance: ErrorHandler;
  private errorLog: EnhancedError[] = [];
  private maxLogSize = 100;
  private errorListeners: Array<(error: EnhancedError) => void> = [];

  static getInstance(): ErrorHandler {
    if (!ErrorHandler.instance) {
      ErrorHandler.instance = new ErrorHandler();
    }
    return ErrorHandler.instance;
  }

  // Add error listener
  addErrorListener(listener: (error: EnhancedError) => void): () => void {
    this.errorListeners.push(listener);
    return () => this.removeErrorListener(listener);
  }

  // Remove error listener
  removeErrorListener(listener: (error: EnhancedError) => void): void {
    const index = this.errorListeners.indexOf(listener);
    if (index > -1) {
      this.errorListeners.splice(index, 1);
    }
  }

  // Enhanced error handling
  handleError(error: any, context?: Record<string, any>): EnhancedError {
    const enhancedError = this.enhanceError(error, context);

    // Log error
    this.logError(enhancedError);

    // Notify listeners
    this.notifyListeners(enhancedError);

    // Log to console based on severity
    this.logToConsole(enhancedError);

    return enhancedError;
  }

  private enhanceError(error: any, context?: Record<string, any>): EnhancedError
{
    let category = ErrorCategory.UNKNOWN;
    let severity = ErrorSeverity.MEDIUM;
    let userMessage = 'An unexpected error occurred';
    let retryable = false;

    // Categorize and analyze error
    if (error instanceof ApiError) {
      category = ErrorCategory.API;
      retryable = error.status ? this.isRetryableStatus(error.status) : false;

      switch (error.status) {
        case 400:
          category = ErrorCategory.VALIDATION;
          severity = ErrorSeverity.LOW;
          userMessage = 'Invalid request. Please check your input.';
          break;
```

```
        case 401:
          category = ErrorCategory.PERMISSION;
          severity = ErrorSeverity.HIGH;
          userMessage = 'Authentication required. Please sign in.';
          break;
        case 403:
          category = ErrorCategory.PERMISSION;
          severity = ErrorSeverity.HIGH;
          userMessage = 'Access denied. You don\'t have permission to perform
this action.';
          break;
        case 404:
          category = ErrorCategory.API;
          severity = ErrorSeverity.LOW;
          userMessage = 'Resource not found.';
          break;
        case 408:
        case 504:
          category = ErrorCategory.TIMEOUT;
          severity = ErrorSeverity.MEDIUM;
          userMessage = 'Request timed out. Please try again.';
          retryable = true;
          break;
        case 429:
          category = ErrorCategory.API;
          severity = ErrorSeverity.MEDIUM;
          userMessage = 'Too many requests. Please wait a moment and try
again.';
          retryable = true;
          break;
        case 500:
        case 502:
        case 503:
          category = ErrorCategory.API;
          severity = ErrorSeverity.HIGH;
          userMessage = 'Server error. Please try again later.';
          retryable = true;
          break;
      }
    } else if (error instanceof TypeError && error.message.includes('fetch')) {
      category = ErrorCategory.NETWORK;
      severity = ErrorSeverity.HIGH;
      userMessage = 'Network connection error. Please check your internet
connection.';
      retryable = true;
    } else if (error.code === 'TIMEOUT') {
      category = ErrorCategory.TIMEOUT;
      severity = ErrorSeverity.MEDIUM;
      userMessage = 'Request timed out. Please try again.';
      retryable = true;
    }

    const enhancedError: EnhancedError = {
      name: error.name || 'Error',
      message: error.message || 'Unknown error',
      category,
      severity,
      timestamp: Date.now(),
      context,
      userMessage,
```

```
    code: error.code,
    retryable,
    stack: error.stack,
  };

  return enhancedError;
}

private isRetryableStatus(status: number): boolean {
  return [408, 429, 500, 502, 503, 504].includes(status);
}

private logError(error: EnhancedError): void {
  this.errorLog.unshift(error);

  // Maintain log size
  if (this.errorLog.length > this.maxLogSize) {
    this.errorLog = this.errorLog.slice(0, this.maxLogSize);
  }
}

private notifyListeners(error: EnhancedError): void {
  this.errorListeners.forEach(listener => {
    try {
      listener(error);
    } catch (err) {
      console.error('Error in error listener:', err);
    }
  });
}

private logToConsole(error: EnhancedError): void {
  const logMethod = this.getLogMethod(error.severity);
  const timestamp = new Date(error.timestamp).toISOString();

  logMethod(
    `[${timestamp}] ${error.category}/${error.severity}:`,
    error.message,
    error.context ? '\nContext:' : '',
    error.context || '',
    error.stack ? '\nStack:' : '',
    error.stack || ''
  );
}

private getLogMethod(severity: ErrorSeverity): typeof console.log {
  switch (severity) {
    case ErrorSeverity.LOW:
      return console.info;
    case ErrorSeverity.MEDIUM:
      return console.warn;
    case ErrorSeverity.HIGH:
    case ErrorSeverity.CRITICAL:
      return console.error;
    default:
      return console.log;
  }
}

// Get error log
```

```
  getErrorLog(): EnhancedError[] {
    return [...this.errorLog];
  }

  // Get errors by category
  getErrorsByCategory(category: ErrorCategory): EnhancedError[] {
    return this.errorLog.filter(error => error.category === category);
  }

  // Get errors by severity
  getErrorsBySeverity(severity: ErrorSeverity): EnhancedError[] {
    return this.errorLog.filter(error => error.severity === severity);
  }

  // Clear error log
  clearErrorLog(): void {
    this.errorLog = [];
  }

  // Check if error is retryable
  static isRetryableError(error: any): boolean {
    if (error instanceof TypeError && error.message.includes('fetch')) {
      return true;
    }
    if (error instanceof ApiError) {
      return error.status ? [408, 429, 500, 502, 503,
504].includes(error.status) : false;
    }
    if (error.code === 'TIMEOUT') {
      return true;
    }
    return false;
  }

  // Format user-friendly error message
  static formatUserMessage(error: any): string {
    if (error instanceof ApiError) {
      switch (error.status) {
        case 400:
          return 'Invalid request. Please check your input.';
        case 401:
          return 'Authentication required. Please sign in.';
        case 403:
          return 'Access denied. You don\'t have permission to perform this
action.';
        case 404:
          return 'Resource not found.';
        case 408:
        case 504:
          return 'Request timed out. Please try again.';
        case 429:
          return 'Too many requests. Please wait a moment and try again.';
        case 500:
        case 502:
        case 503:
          return 'Server error. Please try again later.';
        default:
          return 'An error occurred while processing your request.';
      }
    }
```

```
    if (error instanceof TypeError && error.message.includes('fetch')) {
      return 'Network connection error. Please check your internet connection.';
    }

    if (error.code === 'TIMEOUT') {
      return 'Request timed out. Please try again.';
    }

    return 'An unexpected error occurred. Please try again.';
  }
}

// Export singleton instance
export const errorHandler = ErrorHandler.getInstance();

// Convenience function for handling errors
export const handleError = (error: any, context?: Record<string, any>):
EnhancedError => {
  return errorHandler.handleError(error, context);
};

// Hook for React components to subscribe to errors
export const useErrorHandler = () => {
  const addListener = (listener: (error: EnhancedError) => void) => {
    errorHandler.addErrorListener(listener);
    return () => errorHandler.removeErrorListener(listener);
  };

  return {
    handleError,
    addListener,
    getErrorLog: () => errorHandler.getErrorLog(),
    clearErrorLog: () => errorHandler.clearErrorLog(),
    formatUserMessage: ErrorHandler.formatUserMessage,
    isRetryableError: ErrorHandler.isRetryableError,
  };
};
```

react-frontend-app/src/services/iterative-simplification.ts

```
/**
 * Служба для ітераційного спрощення тексту
 * Аналізує текст, створює мапу проблем та пропонує покращення
 */

import { detection, getCognitiveAccessibilityIndex } from './text-detection';
import { baseTextAPIPath } from '../enums/global.config';

// Словники замін тепер обробляються через API

// Types
export interface TextProblem {
  id: string;
  type: ProblemType;
  word: string;
  position: {
    start: number;
    end: number;
    sentence?: number;
  };
```

```
  severity: number; // 1-10, higher = more impactful for CAI
  caiImpact: number; // Estimated CAI improvement if fixed
  suggestion?: string;
  replacements?: string[];
  morphInfo?: {
    lemma?: string;
    pos?: string;
    upos?: string;
    morph_features?: {[key: string]: string};
    category?: string;
    score?: number;
  };
}

export type ProblemType =
  | 'rare_word'
  | 'term'
  | 'slang'
  | 'abbreviation'
  | 'obsolete'
  | 'profanity'
  | 'obscene'
  | 'vulgar'
  | 'complex'
  | 'archaic'
  | 'long_sentence'
  | 'complex_syntax';

export interface ProblemMap {
  problems: TextProblem[];
  totalProblems: number;
  prioritizedProblems: TextProblem[]; // Sorted by CAI impact
  currentCAI: number;
  potentialCAI: number; // If all problems fixed
}

export interface SyntaxImprovementIteration {
  strategy: string;
  original_sentence: string;
  transformed_sentence: string;
  score_before: number;
  score_after: number;
  sentence_word_count_before: number;
  new_sentences: string[];
}

export interface SyntaxImprovementResponse {
  improvedText: string;
  appliedIterations: number;
  iterations: SyntaxImprovementIteration[];
  status: string;
  initialScore?: number;
  finalScore?: number;
}

export interface SyntaxImprovementOptions {
  maxIterations?: number;
  minImprovement?: number;
}
```

```
export interface SimplificationOptions {
  highlightedWords?: string[]; // If highlight mode is enabled
  maxChangesPerIteration?: number; // Default: 3-5
  minCAIImprovement?: number; // Stop if improvement < this threshold
}

// Типи для нової структури словників замін
export interface PriorityReplacement {
  replacement: string;
  priority: number;
  context: string;
  formality: 'formal' | 'neutral' | 'informal';
}

export interface ReplacementData {
  priority_replacements: PriorityReplacement[];
  word_forms: Record<string, string>;
}

const UI_SETTINGS_STORAGE_KEY = 'ui_settings';

function isLLMStopEnabled(): boolean {
  try {
    if (typeof window === 'undefined' || typeof window.localStorage ===
'undefined') {
      return false;
    }
    const raw = localStorage.getItem(UI_SETTINGS_STORAGE_KEY);
    if (!raw) {
      return false;
    }
    const parsed = JSON.parse(raw);
    return Boolean(parsed?.debug?.llm);
  } catch (error) {
    console.warn('Не вдалося прочитати налаштування Дебаг ЛЛМ:', error);
    return false;
  }
}

function emulateLLMReplacements(word: string, type: ProblemType): string[] {
  const normalized = word || 'текст';
  const variants = [
    `${normalized} ✅`,
    `${normalized} спрощено`,
    `${type}-debug`,
  ];
  console.log(`[Дебаг ЛЛМ] Емулюємо заміни для "${normalized}" (тип: ${type}) -
>`, variants);
  return variants;
}

// Пріоритети категорій будуть оброблятися на бекенді через API

// Fallback словник видалено - використовуємо тільки backend API

/**
 * Створює мапу проблем для заданого тексту
 */
export async function createProblemMap(
  text: string,
```

```
  options: SimplificationOptions = {}
): Promise<ProblemMap> {
  const { highlightedWords, maxChangesPerIteration = 5 } = options;

  try {
    // 1. Отримуємо базовий аналіз тексту через існуючі API
    const detectionResult = await detection({ text });
    console.log('🔍 DETECTION RESULT:', detectionResult);

    // 2. Розбиваємо текст на речення та слова для позиційного аналізу
    const sentences = splitIntoSentences(text);
    const words = extractWordsWithPositions(text);

    // 3. Створюємо проблеми на основі детекції
    const problems: TextProblem[] = [];

    // Додаємо лексичні проблеми
    addLexicalProblems(problems, detectionResult, words, highlightedWords);

    // Додаємо синтаксичні проблеми
    addSyntacticProblems(problems, sentences);

    // 4. Розраховуємо CAI impact для кожної проблеми
    await calculateCAIImpacts(problems, text);

    // 5. Пріоритизуємо проблеми
    const prioritizedProblems = prioritizeProblems(problems,
maxChangesPerIteration);

    // 6. Розраховуємо поточний та потенційний CAI
    const currentCAI = await getCurrentCAI(text);
    const potentialCAI = estimatePotentialCAI(currentCAI, problems);

    return {
      problems,
      totalProblems: problems.length,
      prioritizedProblems,
      currentCAI,
      potentialCAI
    };

  } catch (error) {
    console.error('Помилка створення мапи проблем:', error);
    throw new Error('Не вдалося проаналізувати текст для спрощення');
  }
}

/**
 * Розбиває текст на речення
 */
function splitIntoSentences(text: string): Array<{text: string, start: number,
end: number}> {
  // Простий алгоритм розбиття на речення
  const sentences: Array<{text: string, start: number, end: number}> = [];
  const sentenceRegex = /[.!?]+\s*/g;

  let lastIndex = 0;
  let match;

  while ((match = sentenceRegex.exec(text)) !== null) {
```

```
    const sentenceText = text.slice(lastIndex, match.index +
match[0].length).trim();
    if (sentenceText) {
      sentences.push({
        text: sentenceText,
        start: lastIndex,
        end: match.index + match[0].length
      });
    }
    lastIndex = match.index + match[0].length;
  }

  // Додаємо останнє речення, якщо воно не закінчується розділовим знаком
  if (lastIndex < text.length) {
    const remainingText = text.slice(lastIndex).trim();
    if (remainingText) {
      sentences.push({
        text: remainingText,
        start: lastIndex,
        end: text.length
      });
    }
  }

  return sentences;
}

/**
 * Витягує слова з їх позиціями в тексті
 */
function extractWordsWithPositions(text: string): Array<{word: string, start:
number, end: number}> {
  const words: Array<{word: string, start: number, end: number}> = [];
  const wordRegex = /[\p{L}\p{N}''\-]+/gu;

  let match;
  while ((match = wordRegex.exec(text)) !== null) {
    words.push({
      word: match[0],
      start: match.index,
      end: match.index + match[0].length
    });
  }

  return words;
}

/**
 * Додає лексичні проблеми на основі результатів детекції
 */
// Пошук слів тепер виконується через API на бекенді

function addLexicalProblems(
  problems: TextProblem[],
  detectionResult: any,
  words: Array<{word: string, start: number, end: number}>,
  highlightedWords?: string[]
) {
  console.log('=== DIAGNOSTIC INFO ===');
  console.log('DetectionResult keys:', Object.keys(detectionResult));
```

```
  console.log('Words found:', words.length);
  console.log('First 10 words:', words.slice(0, 10).map(w => w.word));
  // Мап API категорій до типів проблем (оновлено для нової структури)
  const problemTypes: Array<{key: keyof typeof detectionResult, type:
ProblemType, severity: number}> = [
    { key: 'profanity', type: 'profanity', severity: 10 },
    { key: 'obsolete', type: 'obsolete', severity: 9 },
    { key: 'complex', type: 'complex', severity: 8 },
    { key: 'rare', type: 'rare_word', severity: 7 },
    { key: 'terms', type: 'term', severity: 6 },
    { key: 'slang', type: 'slang', severity: 5 },
    { key: 'abbr', type: 'abbreviation', severity: 4 }
  ];

  // Додаткові категорії тепер обробляються API detection на бекенді

  console.log('Final problemTypes:', problemTypes.map(p =>
`${p.key}(${p.type})`));
  console.log('=== END DIAGNOSTIC ===');

  // Обробляємо всі типи проблем
  problemTypes.forEach(({ key, type, severity }) => {
    const detectedWords = detectionResult[key]?.words || [];
    console.log(`Processing ${key}(${type}): ${detectedWords.length} words`);

    detectedWords.forEach((detectedWord: any) => {
      // Обробляємо як старий формат (строка) так і новий (об'єкт з морфологією)
      let wordText: string;
      let wordPosition: {start: number, end: number} | undefined;
      let morphInfo: any = {};

      if (typeof detectedWord === 'string') {
        // Старий формат - просто строка
        wordText = detectedWord;
        // Шукаємо позицію в тексті
        wordPosition = words.find(w =>
          w.word.toLowerCase() === wordText.toLowerCase()
        );
      } else {
        // Новий формат - об'єкт з детальною інформацією
        wordText = detectedWord.word || detectedWord.text || detectedWord.lemma
|| '';

        // Використовуємо позицію з API, якщо доступна
        if (detectedWord.start !== undefined && detectedWord.end !== undefined)
{
          wordPosition = {
            start: detectedWord.start,
            end: detectedWord.end
          };
        } else {
          // Fallback - шукаємо позицію в тексті
          wordPosition = words.find(w =>
            w.word.toLowerCase() === wordText.toLowerCase()
          );
        }

        // Зберігаємо морфологічну інформацію
        morphInfo = {
          lemma: detectedWord.lemma,
```

```
          pos: detectedWord.pos,
          upos: detectedWord.upos,
          morph_features: detectedWord.morph_features,
          category: detectedWord.category,
          score: detectedWord.score
        };
      }

      if (!wordText || !wordPosition) return;

      // Якщо є highlighted words, перевіряємо чи це слово підсвічене
      if (highlightedWords &&
!highlightedWords.includes(wordText.toLowerCase())) {
        return; // Пропускаємо не підсвічені слова
      }

      problems.push({
        id: `${type}_${wordPosition.start}_${Date.now()}`,
        type,
        word: wordText,
        position: {
          start: wordPosition.start,
          end: wordPosition.end
        },
        severity,
        caiImpact: 0, // Буде розраховано пізніше
        replacements: [], // Буде заповнено при генерації замін
        morphInfo // Додаємо морфологічну інформацію для використання при заміні
      });
    });
  });
}

/**
 * Додає синтаксичні проблеми (довгі речення, складний синтаксис)
 */
function addSyntacticProblems(
  problems: TextProblem[],
  sentences: Array<{text: string, start: number, end: number}>
) {
  sentences.forEach((sentence, index) => {
    const wordCount = sentence.text.split(/\s+/).length;

    // Довгі речення (>25 слів)
    if (wordCount > 25) {
      problems.push({
        id: `long_sentence_${sentence.start}_${Date.now()}`,
        type: 'long_sentence',
        word: sentence.text.substring(0, 50) + '...', // Показуємо початок
речення
        position: {
          start: sentence.start,
          end: sentence.end,
          sentence: index
        },
        severity: Math.min(10, Math.floor(wordCount / 5)), // Чим довше, тим
гірше
        caiImpact: 0,
        suggestion: `Розбити речення (${wordCount} слів)`
      });
```

```
    }

    // Складний синтаксис (багато ком)
    const commaCount = (sentence.text.match(/,/g) || []).length;
    if (commaCount > 3) {
      problems.push({
        id: `complex_syntax_${sentence.start}_${Date.now()}`,
        type: 'complex_syntax',
        word: sentence.text.substring(0, 50) + '...',
        position: {
          start: sentence.start,
          end: sentence.end,
          sentence: index
        },
        severity: Math.min(8, commaCount * 2),
        caiImpact: 0,
        suggestion: `Спростити структуру (${commaCount} ком)`
      });
    }
  });
}

/**
 * Розраховує CAI impact для кожної проблеми
 */
async function calculateCAIImpacts(problems: TextProblem[], originalText:
string) {
  // Реальні ваги впливу на CAI (отримані експериментально)
  const impactMultipliers: Record<ProblemType, number> = {
    'profanity': 0.024,      // НАЙВИЩИЙ вплив (експериментально підтверджено)
    'obsolete': 0.020,       // ВИСОКИЙ вплив (експериментально підтверджено)
    'archaic': 0.020,        // Архаїчні слова як застарілі
    'obscene': 0.019,        // Вульгарні слова майже як лайка
    'vulgar': 0.017,         // Вульгарні слова
    'term': 0.012,           // Терміни помірно впливають
    'slang': 0.010,          // Сленг менше впливає
    'complex': 0.009,        // СЕРЕДНІЙ вплив (експериментально знижено)
    'abbreviation': 0.008,   // Абревіатури помірно впливають
    'rare_word': 0.001,      // МІНІМАЛЬНИЙ вплив (експериментально знижено)
    'long_sentence': 0.025,  // Довгі речення сильно впливають на синтаксичний
компонент
    'complex_syntax': 0.020  // Складний синтаксис також значно впливає
  };

  problems.forEach(problem => {
    const baseImpact = impactMultipliers[problem.type] || 0.01;

    // Розраховуємо impact з урахуванням severity (1-10)
    problem.caiImpact = baseImpact * (problem.severity / 10);

    // Додаткові коефіцієнти для окремих типів
    if (problem.type === 'long_sentence') {
      // Чим довше речення, тим більший impact
      const wordCount = problem.word.split(/\s+/).length;
      if (wordCount > 40) {
        problem.caiImpact *= 1.5; // Дуже довгі речення мають ще більший impact
      }
    }

    // Обмежуємо максимальний impact однієї проблеми
```

```
    problem.caiImpact = Math.min(problem.caiImpact, 0.05);
  });

  // TODO: У майбутньому можна додати більш точний розрахунок,
  // тестуючи CAI до та після виправлення кожної проблеми,
  // але це буде повільно для великих текстів
}

/**
 * Пріоритизує проблеми за CAI impact
 */
function prioritizeProblems(problems: TextProblem[], maxChanges: number):
TextProblem[] {
  return problems
    .sort((a, b) => b.caiImpact - a.caiImpact) // Сортуємо за спаданням CAI
impact
    .slice(0, maxChanges); // Беремо тільки топ N проблем
}

/**
 * Отримує поточний CAI тексту
 */
async function getCurrentCAI(text: string): Promise<number> {
  try {
    const caiData = await getCognitiveAccessibilityIndex({ text });
    return caiData?.cai || 0.5; // Fallback до 0.5 якщо API не працює
  } catch (error) {
    console.warn('Не вдалося отримати CAI, використовуємо fallback:', error);
    return 0.5;
  }
}

/**
 * Оцінює потенційний CAI після виправлення всіх проблем
 */
function estimatePotentialCAI(currentCAI: number, problems: TextProblem[]):
number {
  const totalImprovement = problems.reduce((sum, p) => sum + p.caiImpact, 0);
  return Math.min(1.0, currentCAI + totalImprovement);
}

/**
 * Генерує заміни для проблем
 */
export async function generateReplacements(problems: TextProblem[]):
Promise<TextProblem[]> {
  console.log('=== GENERATE REPLACEMENTS START ===');
  console.log(`Отримано ${problems.length} проблем для генерації замін`);

  const problemsWithReplacements = [...problems];

  for (const problem of problemsWithReplacements) {
    console.log(`🔄 Генеруємо заміни для: "${problem.word}" (тип:
${problem.type})`);
    console.log(`📍 Позиція: ${problem.position.start}-
${problem.position.end}`);
    console.log(`🧬 Морфологія:`, problem.morphInfo);

    try {
      const replacements = await generateReplacementsForWord(problem.word,
```

```
problem.type, problem.morphInfo);
      problem.replacements = replacements;

      console.log(`✅ Згенеровано ${replacements.length} замін для
"${problem.word}": [${replacements.join(', ')}]`);
    } catch (error) {
      console.warn(`❌ Не вдалося згенерувати заміни для "${problem.word}":`,
error);
      problem.replacements = [];
    }
  }

  console.log('=== GENERATE REPLACEMENTS END ===');
  console.log(`Результат: ${problemsWithReplacements.filter(p => p.replacements
&& p.replacements.length > 0).length} проблем з замінами`);

  return problemsWithReplacements;
}

// Функція завантаження словників видалена - використовуємо тільки backend API

/**
 * Генерує заміни для конкретного слова з морфологічною інформацією
 */
async function generateReplacementsForWord(word: string, type: ProblemType,
morphInfo?: any): Promise<string[]> {
  console.log(`Генеруємо заміни для слова "${word}" типу "${type}"`);

  if (isLLMStopEnabled()) {
    return emulateLLMReplacements(word, type);
  }

  try {
    // Викликаємо API бекенду для отримання замін з морфологічною інформацією
    const requestBody: any = {
      word: word,
      type: type
    };

    // Додаємо морфологічну інформацію, якщо доступна
    if (morphInfo && morphInfo.morph_features) {
      requestBody.morph_info = {
        lemma: morphInfo.lemma,
        upos: morphInfo.upos,
        morph_features: morphInfo.morph_features
      };
      console.log(`Передаємо морфологічну інформацію для "${word}":`,
requestBody.morph_info);
    }

    const response = await fetch(`${baseTextAPIPath}/replace-word`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(requestBody)
    });

    if (response.ok) {
      const data = await response.json();
```

```
      console.log(`API відповідь для "${word}":`, data);

      if (data.found && data.replacements && data.replacements.length > 0) {
        let finalReplacements = data.replacements;

        // Якщо доступне морфологічне узгодження, використовуємо його
        if (data.morphologically_adjusted && data.morphologically_adjusted_word)
{
          console.log(`Використовуємо морфологічно узгоджену заміну для
"${word}": "${data.morphologically_adjusted_word}"`);
          finalReplacements = [data.morphologically_adjusted_word,
...data.replacements];
        }

        console.log(`✅ Знайдено заміни для "${word}":`, finalReplacements);
        return finalReplacements.slice(0, 3); // Максимум 3 заміни
      } else {
        console.log(`❌ Слово "${word}" не знайдено в словнику backend`);
        return [];
      }
    } else {
      console.warn(`API помилка для "${word}": ${response.status}`);
    }
  } catch (error) {
    console.warn(`Помилка виклику API для "${word}":`, error);
  }

  // Якщо API недоступне або не знайдено заміну - повертаємо порожній масив
  console.log(`❌ Заміну не знайдено для "${word}" (тип: ${type})`);
  return [];
}

// Fallback функція видалена - використовуємо тільки backend API

/**
 * Виконує rule-based синтаксичне спрощення на бекенді
 */
export async function applySyntaxImprovement(
  text: string,
  options: SyntaxImprovementOptions = {}
): Promise<SyntaxImprovementResponse> {
  const { maxIterations = 3, minImprovement = 0.01 } = options;
  try {
    const response = await fetch(`${baseTextAPIPath}/syntax-improvement`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        text,
        max_iterations: maxIterations,
        min_improvement: minImprovement,
      }),
    });

    if (!response.ok) {
      throw new Error(`Syntax improvement failed with status
${response.status}`);
    }

    const data = await response.json();
    return {
```

```
      improvedText: data?.improved_text ?? text,
      appliedIterations: data?.applied_iterations ?? 0,
      iterations: data?.iterations ?? [],
      status: data?.status ?? 'unknown',
      initialScore: data?.initial_sentence_structure_score,
      finalScore: data?.final_sentence_structure_score,
    };
  } catch (error) {
    console.warn('Не вдалося застосувати синтаксичне спрощення:', error);
    return {
      improvedText: text,
      appliedIterations: 0,
      iterations: [],
      status: 'error',
    };
  }
}

/**
 * Застосовує заміни до тексту
 */
export function applyReplacements(originalText: string, problems:
TextProblem[]): string {
  console.log('=== APPLY REPLACEMENTS START ===');
  console.log(`📝 Оригінальний текст: "${originalText}"`);
  console.log(`🔢 Кількість проблем: ${problems.length}`);

  // Детальний аналіз проблем
  problems.forEach((p, i) => {
    console.log(`📋 Проблема ${i+1}: "${p.word}" (${p.type}) позиція
${p.position.start}-${p.position.end}`);
    console.log(`   Заміни: ${p.replacements ? `[${p.replacements.join(', ')}]`
: 'НЕМАЄ'}`);
  });

  let modifiedText = originalText;

  // Сортуємо проблеми за позицією в зворотному порядку
  // щоб зміни не впливали на позиції наступних замін
  const sortedProblems = problems
    .filter(p => p.replacements && p.replacements.length > 0)
    .sort((a, b) => b.position.start - a.position.start);

  console.log(`🔄 Проблеми з замінами: ${sortedProblems.length} з
${problems.length}`);

  if (sortedProblems.length === 0) {
    console.log('⚠️  НЕМАЄ ПРОБЛЕМ З ЗАМІНАМИ! Повертаємо оригінальний текст');
    return modifiedText;
  }

  for (const problem of sortedProblems) {
    if (!problem.replacements || problem.replacements.length === 0) {
      console.log(`⏭️  Пропускаємо проблему "${problem.word}" - немає замін`);
      continue;
    }

    // Беремо першу заміну (найкращу)
    const replacement = problem.replacements[0];
```

```
    console.log(`🔄 Застосовуємо заміну: "${problem.word}"
(${problem.position.start}-${problem.position.end}) -> "${replacement}"`);

    // Перевіряємо чи слово все ще на тій самій позиції
    const currentWordAtPosition = modifiedText.substring(problem.position.start,
problem.position.end);
    console.log(`🔍 Слово на позиції зараз: "${currentWordAtPosition}"`);

    // Застосовуємо заміну
    const before = modifiedText.substring(0, problem.position.start);
    const after = modifiedText.substring(problem.position.end);
    const newText = before + replacement + after;

    console.log(`📝 Текст до заміни: "${modifiedText}"`);
    console.log(`📝 Текст після заміни: "${newText}"`);

    modifiedText = newText;
  }

  console.log('=== APPLY REPLACEMENTS END ===');
  console.log(`✅ Фінальний текст: "${modifiedText}"`);
  console.log(`📊 Змінений: ${originalText !== modifiedText ? 'ТАК' : 'НІ'}`);

  return modifiedText;
}
```

react-frontend-app/src/services/iterative-simplify-backend.ts

```
import { api } from './api-client';
import { BatchAnalyzeResponse } from '../types/api';

export interface SimplifyScope {
  start: number;
  end: number;
}

export interface AppliedChange {
  type: string;
  start: number;
  end: number;
  from: string;
  to: string;
}

export interface IterativeSimplifyResponse {
  algorithm_id?: string;
  algorithm_version?: number;
  mode?: 'document' | 'page' | 'cursor' | 'custom';
  scope: SimplifyScope;
  limits_used?: { max_changes: number; steps: number };
  timings_ms?: { total: number; analysis: number; iterations: number[] };
  steps_requested?: number;
  steps_performed?: number;
  applied_changes: AppliedChange[];
  applied_count: number;
  total_applied_count?: number;
  iterations?: Array<{ applied_count: number; applied_changes: AppliedChange[]
}>;
  text: string;
  analysis?: BatchAnalyzeResponse | null;
```

```
}

export type IterativeSimplifyConfig = { signal?: AbortSignal; abortKey?: string;
dedupe?: boolean };

export async function iterativeSimplifyDocument(params: {
  text: string;
  max_changes?: number;
  steps?: number;
  allow_llm?: boolean;
  include_analysis?: boolean;
  include_cai?: boolean;
  syntax?: { enabled?: boolean; max_iterations?: number; min_improvement?:
number };
}, config?: IterativeSimplifyConfig): Promise<IterativeSimplifyResponse> {
  return api.post<IterativeSimplifyResponse>('iterative-simplify/document',
params, config);
}

export async function iterativeSimplifyPage(params: {
  text: string;
  scope: SimplifyScope;
  max_changes?: number;
  steps?: number;
  allow_llm?: boolean;
  include_analysis?: boolean;
  include_cai?: boolean;
  syntax?: { enabled?: boolean; max_iterations?: number; min_improvement?:
number };
}, config?: IterativeSimplifyConfig): Promise<IterativeSimplifyResponse> {
  return api.post<IterativeSimplifyResponse>('iterative-simplify/page', params,
config);
}

export async function iterativeSimplifyCursor(params: {
  text: string;
  cursor_index: number;
  max_changes?: number;
  steps?: number;
  allow_llm?: boolean;
  include_analysis?: boolean;
  include_cai?: boolean;
  syntax?: { enabled?: boolean; max_iterations?: number; min_improvement?:
number };
}, config?: IterativeSimplifyConfig): Promise<IterativeSimplifyResponse> {
  return api.post<IterativeSimplifyResponse>('iterative-simplify/cursor',
params, config);
}

// Backward-compatible generic endpoint (still supported on backend).
export async function iterativeSimplify(params: {
  text: string;
  scope?: SimplifyScope;
  max_changes?: number;
  steps?: number;
  allow_llm?: boolean;
  include_analysis?: boolean;
  include_cai?: boolean;
  syntax?: { enabled?: boolean; max_iterations?: number; min_improvement?:
number };
```

```
}, config?: IterativeSimplifyConfig): Promise<IterativeSimplifyResponse> {
  return api.post<IterativeSimplifyResponse>('iterative-simplify', params,
config);
}
```

react-frontend-app/src/services/stable-emotion.service.ts

```
import { timeDetectionStableEmotionMS } from '../enums/global.config';

let timeEmotion = Date.now();
let currentEmotion = '';
let stableEmotion: string | undefined;

export const getStableEmotion = (emotion: string): string | undefined => {
  if (currentEmotion !== emotion) {
    timeEmotion = Date.now();
  }

  if (currentEmotion === emotion && (Date.now() - timeEmotion >
timeDetectionStableEmotionMS)) {
    stableEmotion = emotion;
  }

  currentEmotion = emotion;

  return stableEmotion;
};
```

react-frontend-app/src/services/text-detection.ts

```
import { api } from './api-client';
import { DetectionResponse, DetectionResult, AccessibilityData } from
'../types/api';

const sanitizeText = (text: any): string => {
  if (typeof text === 'string') {
    return text;
  }
  if (text && typeof text === 'object') {
    if (text.toString && typeof text.toString === 'function') {
      return String(text.toString());
    }
    return String(text);
  }
  return String(text || '');
};

export const detection = async ({ text }: { text: any }):
Promise<DetectionResponse> => {
  const result: DetectionResponse = {};
  const sanitizedText = sanitizeText(text);

  try {
    result.obsolete = await getObsoleteScore({ text: sanitizedText });
    result.rare = await getRareScore({ text: sanitizedText });
    result.abbr = await getAbbrScore({ text: sanitizedText });
    result.terms = await getTermScore({ text: sanitizedText });
    result.profanity = await getProfanityScore({ text: sanitizedText });
    result.complex = await getComplexScore({ text: sanitizedText });
    result.flesch = await getFleschScore({ text: sanitizedText });
    result.smog = await getSMOGScore({ text: sanitizedText });
```

```
    result.dlevel = await getDLevelScore({ text: sanitizedText });
    result.slang = await getSlangScore({ text: sanitizedText });
    // Якщо потрібно — додати інші сервіси

    return result;
  } catch (error) {
    console.error('Error in detection:', error);
    throw error;
  }
};

const getObsoleteScore = async ({ text }: { text: string }):
Promise<DetectionResult> => {
  try {
    const res = await api.post('obsolete-score', { text });
    return { words: res.words || [], found_count: res.found_count || 0 };
  } catch (error) {
    console.error('Error getting obsolete score:', error);
    return { words: [], found_count: 0 };
  }
};

const getRareScore = async ({ text }: { text: string }):
Promise<DetectionResult> => {
  try {
    const res = await api.post('rare-score', { text });
    return { words: res.words || [], found_count: res.found_count || 0 };
  } catch (error) {
    console.error('Error getting rare score:', error);
    return { words: [], found_count: 0 };
  }
};

const getTermScore = async ({ text }: { text: string }):
Promise<DetectionResult> => {
  try {
    const res = await api.post('term-score', { text });
    return { words: res.words || [], found_count: res.found_count || 0 };
  } catch (error) {
    console.error('Error getting term score:', error);
    return { words: [], found_count: 0 };
  }
};

const getAbbrScore = async ({ text }: { text: string }):
Promise<DetectionResult> => {
  try {
    const res = await api.post('abbr-score', { text });
    return { words: res.words || [], found_count: res.found_count || 0 };
  } catch (error) {
    console.error('Error getting abbr score:', error);
    return { words: [], found_count: 0 };
  }
};

const getSlangScore = async ({ text }: { text: string }):
Promise<DetectionResult> => {
  try {
    const res = await api.post('slang-score', { text});
    return { words: res.words || [], found_count: res.found_count || 0 };
```

```
  } catch (error) {
    console.error('Error getting slang score:', error);
    return { words: [], found_count: 0 };
  }
};

export const getProfanityScore = async ({ text }: { text: string }):
Promise<DetectionResult> => {
  try {
    const res = await api.post('profanity-score', { text });
    return { words: res.words || [], found_count: res.found_count || 0 };
  } catch (error) {
    console.error('Error getting profanity score:', error);
    return { words: [], found_count: 0 };
  }
};

const getComplexScore = async ({ text }: { text: string }):
Promise<DetectionResult> => {
  try {
    const res = await api.post('complex-score', { text });
    return { words: res.words || [], found_count: res.found_count || 0 };
  } catch (error) {
    console.error('Error getting complex score:', error);
    return { words: [], found_count: 0 };
  }
};

const getFleschScore = async ({ text }: { text: string }):
Promise<DetectionResult> => {
  try {
    const res = await api.post('flesch-kincaid', { text });
    // Для Flesch-Kincaid використовуємо калібрований показник складності,
    // а не класичний Flesch Reading Ease.
    return {
      words: [],
      found_count: res.flesch_kincaid?.grade_level || 0,
      flesch_data: res.flesch_kincaid // Додаткові дані для можливого
використання
    };
  } catch (error) {
    console.error('Error getting flesch score:', error);
    return { words: [], found_count: 0 };
  }
};

const getSMOGScore = async ({ text }: { text: string }):
Promise<DetectionResult> => {
  try {
    const res = await api.post('smog', { text });
    // Для SMOG повертаємо smog_grade як found_count
    return {
      words: [],
      found_count: res.smog?.smog_grade || 0,
      smog_data: res.smog // Додаткові дані для можливого використання
    };
  } catch (error) {
    console.error('Error getting SMOG score:', error);
    return { words: [], found_count: 0 };
  }
```

```
};

const getDLevelScore = async ({ text }: { text: string }):
Promise<DetectionResult> => {
  try {
    const res = await api.post('d-level', { text });
    // Для D-Level повертаємо d_level_score як found_count
    return {
      words: [],
      found_count: res.d_level?.d_level_score || 0,
      dlevel_data: res.d_level // Додаткові дані для можливого використання
    };
  } catch (error) {
    console.error('Error getting D-Level score:', error);
    return { words: [], found_count: 0 };
  }
};

export const getCognitiveAccessibilityIndex = async ({ text }: { text: any }):
Promise<AccessibilityData | null> => {
  try {
    const sanitizedText = sanitizeText(text);
    const res = await api.post('cognitive-accessibility-index', { text:
sanitizedText });
    return res as AccessibilityData;
  } catch (error) {
    console.error('Error getting Cognitive Accessibility Index:', error);
    return null;
  }
};
```

react-frontend-app/src/services/text-transformation.ts

```
import data from '../mockdata/texts.json';

export const getTransformText = (text?: string): string => {
  let texts: string;

  if (!text) {
    // Fallback to mock data only if no text is provided
    texts = Object.values(data)[0] as string;
  } else {
    texts = text;
  }

  return texts;
};
```

react-frontend-app/src/hooks/useErrorNotification.ts

```
import { useEffect, useState } from 'react';
import { errorHandler, EnhancedError, ErrorSeverity } from '../services/error-
handler';

interface ErrorNotification {
  id: string;
  message: string;
  severity: ErrorSeverity;
  timestamp: number;
  dismissed: boolean;
}
```

```
interface UseErrorNotificationReturn {
  notifications: ErrorNotification[];
  dismissNotification: (id: string) => void;
  dismissAllNotifications: () => void;
  clearNotifications: () => void;
}

export const useErrorNotification = (
  maxNotifications: number = 5,
  autoDismissAfter: number = 5000
): UseErrorNotificationReturn => {
  const [notifications, setNotifications] = useState<ErrorNotification[]>([]);

  useEffect(() => {
    const unsubscribe = errorHandler.addErrorListener((error: EnhancedError) =>
{
      // Only show user-facing errors
      if (error.severity === ErrorSeverity.LOW) return;

      const notification: ErrorNotification = {
        id: `${error.timestamp}-${Math.random()}`,
        message: error.userMessage || error.message,
        severity: error.severity,
        timestamp: error.timestamp,
        dismissed: false,
      };

      setNotifications(prev => {
        const updated = [notification, ...prev.slice(0, maxNotifications - 1)];
        return updated;
      });

      // Auto-dismiss non-critical errors
      if (error.severity !== ErrorSeverity.CRITICAL && autoDismissAfter > 0) {
        setTimeout(() => {
          dismissNotification(notification.id);
        }, autoDismissAfter);
      }
    });

    return unsubscribe;
  }, [maxNotifications, autoDismissAfter]);

  const dismissNotification = (id: string) => {
    setNotifications(prev =>
      prev.map(notification =>
        notification.id === id
          ? { ...notification, dismissed: true }
          : notification
      )
    );

    // Remove dismissed notifications after animation
    setTimeout(() => {
      setNotifications(prev =>
        prev.filter(notification => notification.id !== id)
      );
    }, 300);
  };
```

```
  const dismissAllNotifications = () => {
    setNotifications(prev =>
      prev.map(notification => ({ ...notification, dismissed: true }))
    );

    setTimeout(() => {
      setNotifications([]);
    }, 300);
  };

  const clearNotifications = () => {
    setNotifications([]);
  };

  return {
    notifications: notifications.filter(n => !n.dismissed),
    dismissNotification,
    dismissAllNotifications,
    clearNotifications,
  };
};
```

react-frontend-app/src/hooks/useFetchEmotion.tsx

```
import { useState, useEffect, useRef } from 'react';
import {
  defaultDelay,
  basePath,
  negativeConsensusWindowMS,
  consensusHoldDurationMS,
} from '../enums/global.config';
import { getStableEmotion } from '../services/stable-emotion.service';
import axios from 'axios';
import { EmotionResponse } from '../types/api';
import { useDebugSettings } from '../context/AppProviders';

export interface EmotionStatus {
  value: string;
  fastState?: string | null;
  slowState?: string | null;
  negativeConsensus: boolean;
  isConsensusStable: boolean;
  consensusDurationMs: number;
  fastAlert: boolean;
  consensusActive: boolean;
}

const initialStatus: EmotionStatus = {
  value: '',
  negativeConsensus: false,
  isConsensusStable: false,
  consensusDurationMs: 0,
  fastState: null,
  slowState: null,
  fastAlert: false,
  consensusActive: false,
};

const useFetchEmotion = (): EmotionStatus => {
  const [status, setStatus] = useState<EmotionStatus>(initialStatus);
```

```
  const consensusStartRef = useRef<number | null>(null);
  const fastAlertRef = useRef<number | null>(null);
  const consensusHoldRef = useRef<number | null>(null);
  const { options: debugOptions } = useDebugSettings();
  const stopCamera = debugOptions.camera;

  useEffect(() => {
    if (stopCamera) {
      consensusStartRef.current = null;
      fastAlertRef.current = null;
      consensusHoldRef.current = null;
      setStatus(() => ({ ...initialStatus }));
      return;
    }

    const fetchEmotion = async (): Promise<void> => {
      try {
        const response = await
axios.get<EmotionResponse>(`${basePath}/emotion`);
        const negativeStates = new Set(['confused', 'overloaded']);
        const slowState = response.data.stateSlow ?? response.data.state ?? '';
        const fastState = response.data.stateFast ?? '';
        const slowNegative = negativeStates.has(slowState);
        const fastNegative = negativeStates.has(fastState);

        const now = Date.now();
        let consensusDuration = 0;
        if (response.data.negativeConsensus) {
          if (!consensusStartRef.current) {
            consensusStartRef.current = now;
          }
          consensusDuration = now - consensusStartRef.current;
        } else {
          consensusStartRef.current = null;
          consensusDuration = 0;
        }

        // Якщо повільна гілка бачить стрес, а швидка ні — вважаємо, що сигнал
ще не підтверджений.
        let fastAlertActive = false;
        if (fastNegative) {
          if (!fastAlertRef.current) {
            fastAlertRef.current = now;
          }
          fastAlertActive = now - fastAlertRef.current >=
negativeConsensusWindowMS;
        } else {
          fastAlertRef.current = null;
        }

        const detectedEmotion = response.data.emotionSlow ??
response.data.emotion;
        const stableEmotion = detectedEmotion ?
getStableEmotion(detectedEmotion) : undefined;
        const isConsensusStable =
          Boolean(response.data.negativeConsensus) && consensusDuration >=
negativeConsensusWindowMS;

        let consensusActive = false;
        if (isConsensusStable) {
```

```
          consensusHoldRef.current = now;
          consensusActive = true;
        } else if (
          consensusHoldRef.current &&
          now - consensusHoldRef.current <= consensusHoldDurationMS
        ) {
          consensusActive = true;
        } else {
          consensusHoldRef.current = null;
          consensusActive = false;
        }

        setStatus(prev => ({
          value: stableEmotion ?? prev.value,
          fastState,
          slowState,
          negativeConsensus: Boolean(response.data.negativeConsensus),
          isConsensusStable,
          consensusDurationMs: consensusDuration,
          fastAlert: fastAlertActive,
          consensusActive,
        }));
      } catch (error) {
        console.error('Error fetching emotion:', error);
      }
    };

    fetchEmotion(); // run immediately so we don't wait for the first interval
tick
    const interval = setInterval(fetchEmotion, defaultDelay);
    return () => clearInterval(interval);
  }, [stopCamera]);

  return status;
};

export default useFetchEmotion;
```

react-frontend-app/src/hooks/useSurveyForm.ts

```
import { useEffect, useRef, useState, type FormEvent } from 'react';
import { sendSurveyApi } from '../utils/api';

interface UseSurveyFormProps {
  t: (key: string) => string;
  fileName: string;
  page: number;
  simplificationLevel: number;
}

interface SurveyFormState {
  rating: number;
  reasons: string[];
  loading: boolean;
  submitted: boolean;
  error: string | null;
  REASONS: string[];
  handleStarClick: (value: number) => void;
  handleReasonChange: (reason: string) => void;
  handleSubmit: (e?: FormEvent) => Promise<void>;
```

```
}

export function useSurveyForm({ t, fileName, page, simplificationLevel }:
UseSurveyFormProps): SurveyFormState {
  const [rating, setRating] = useState<number>(0);
  const [reasons, setReasons] = useState<string[]>([]);
  const [loading, setLoading] = useState<boolean>(false);
  const [submitted, setSubmitted] = useState<boolean>(false);
  const [error, setError] = useState<string | null>(null);
  const resetTimer = useRef<ReturnType<typeof setTimeout> | null>(null);

  const REASONS: string[] = [
    'Не якісне покращення',
    'Не зрозумілий текст',
    'Довге очікування',
    'Втрата фокусу при зміні тексту',
  ];

  const handleStarClick = (value: number): void => {
    setRating(value);
    if (value >= 7) setReasons([]);
  };

  const handleReasonChange = (reason: string): void => {
    setReasons((prev) =>
      prev.includes(reason)
        ? prev.filter((r) => r !== reason)
        : [...prev, reason]
    );
  };

  const handleSubmit = async (e?: FormEvent): Promise<void> => {
    e?.preventDefault();
    setLoading(true);
    setError(null);

    try {
      await sendSurveyApi({
        date: new Date().toISOString(),
        rating,
        reasons,
        fileName,
        page,
        simplificationLevel,
      }, t);

      setSubmitted(true);
      if (resetTimer.current) {
        clearTimeout(resetTimer.current);
      }
      resetTimer.current = setTimeout(() => {
        setRating(0);
        setReasons([]);
        setSubmitted(false);
        resetTimer.current = null;
      }, 3000);
    } catch (error) {
      console.error(t('Error submitting survey:'), error);
      setSubmitted(false);
      setError(t('Error submitting survey. Please try again.'));
```

```
    } finally {
      setLoading(false);
    }
  };

  useEffect(() => {
    return () => {
      if (resetTimer.current) {
        clearTimeout(resetTimer.current);
      }
    };
  }, []);

  return {
    rating,
    reasons,
    loading,
    submitted,
    error,
    REASONS,
    handleStarClick,
    handleReasonChange,
    handleSubmit,
  };
}
```

react-frontend-app/src/hooks/useTtsPlayer.ts

```
import { useCallback, useEffect, useRef, useState } from 'react';
import { TtsChunkInput } from '../utils/tts';
import { baseTTSAPIPath } from '../enums/global.config';

interface WordTiming {
  text: string;
  start: number;
  end: number;
  start_ms: number;
  end_ms: number;
}

interface TtsStreamChunk {
  index: number;
  text: string;
  audio: string;
  duration_ms: number;
  sample_rate: number;
  chunk_start: number;
  chunk_end: number;
  words: WordTiming[];
}

interface TtsStreamError {
  error: string;
  code?: string;
  details?: string;
}

interface QueueItem {
  url: string;
  durationMs: number;
```

```
  chunkStart: number;
  chunkEnd: number;
  words: WordTiming[];
}

interface UseTtsPlayerOptions {
  onWordRange?: (range: { start: number; end: number } | null) => void;
}

export function useTtsPlayer(options: UseTtsPlayerOptions = {}) {
  const { onWordRange } = options;
  const [isStreaming, setIsStreaming] = useState(false);
  const [isPlaying, setIsPlaying] = useState(false);
  const [error, setError] = useState<string | null>(null);
  const abortRef = useRef<AbortController | null>(null);
  const audioRef = useRef<HTMLAudioElement | null>(null);
  const queueRef = useRef<QueueItem[]>([]);
  const streamEndedRef = useRef(false);
  const playingRef = useRef(false);
  const receivedRef = useRef(false);
  const streamFailedRef = useRef(false);

  const clearQueue = useCallback(() => {
    queueRef.current.forEach((item) => URL.revokeObjectURL(item.url));
    queueRef.current = [];
  }, []);

  const stop = useCallback(() => {
    abortRef.current?.abort();
    abortRef.current = null;
    streamEndedRef.current = true;
    const audio = audioRef.current;
    if (audio) {
      audio.pause();
      audio.currentTime = 0;
    }
    clearQueue();
    setIsPlaying(false);
    playingRef.current = false;
    setIsStreaming(false);
    setError(null);
    onWordRange?.(null);
  }, [clearQueue, onWordRange]);

  const playNext = useCallback(() => {
    const next = queueRef.current.shift();
    if (!next) {
      setIsPlaying(false);
      playingRef.current = false;
      if (streamEndedRef.current) {
        setIsStreaming(false);
        onWordRange?.(null);
      }
      return;
    }

    const audio = audioRef.current ?? new Audio();
    audioRef.current = audio;
    audio.src = next.url;
```

```
    audio.onended = () => {
      onWordRange?.(null);
      playNext();
    };

    audio.ontimeupdate = () => {
      const currentMs = audio.currentTime * 1000;
      const word = next.words.find((w) => currentMs >= w.start_ms && currentMs <
w.end_ms);
      if (word) {
        onWordRange?.({
          start: next.chunkStart + word.start,
          end: next.chunkStart + word.end,
        });
      }
    };

    audio.play().then(() => {
      setIsPlaying(true);
      playingRef.current = true;
    }).catch((err) => {
      setError(err instanceof Error ? err.message : 'audio_playback_error');
      setIsPlaying(false);
      playingRef.current = false;
    });
  }, [onWordRange]);

  const enqueueChunk = useCallback((chunk: TtsStreamChunk) => {
    const binary = atob(chunk.audio);
    const bytes = new Uint8Array(binary.length);
    for (let i = 0; i < binary.length; i += 1) {
      bytes[i] = binary.charCodeAt(i);
    }
    const blob = new Blob([bytes], { type: 'audio/wav' });
    const url = URL.createObjectURL(blob);
    queueRef.current.push({
      url,
      durationMs: chunk.duration_ms,
      chunkStart: chunk.chunk_start,
      chunkEnd: chunk.chunk_end,
      words: chunk.words,
    });

    if (!playingRef.current) {
      playNext();
    }
  }, [playNext]);

  const parseStreamLine = useCallback((line: string): TtsStreamChunk => {
    const parsed = JSON.parse(line) as TtsStreamChunk | TtsStreamError;
    if ('error' in parsed) {
      throw new Error(parsed.code ? `${parsed.code}: ${parsed.error}` :
parsed.error);
    }
    return parsed;
  }, []);

  const start = useCallback(async (chunks: TtsChunkInput[]) => {
    if (!chunks.length) {
      setError('no_text');
```

```
      return;
    }
    stop();
    setError(null);
    setIsStreaming(true);
    streamEndedRef.current = false;
    receivedRef.current = false;
    streamFailedRef.current = false;
    const controller = new AbortController();
    abortRef.current = controller;

    try {
      const response = await fetch(`${baseTTSAPIPath}/tts/stream`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ chunks }),
        signal: controller.signal,
      });

      if (!response.ok || !response.body) {
        throw new Error(`tts_http_${response.status}`);
      }

      const reader = response.body.getReader();
      const decoder = new TextDecoder();
      let buffer = '';

      while (true) {
        const { value, done } = await reader.read();
        if (done) break;
        buffer += decoder.decode(value, { stream: true });
        const lines = buffer.split('\n');
        buffer = lines.pop() || '';
        for (const line of lines) {
          const trimmed = line.trim();
          if (!trimmed) continue;
          const parsed = parseStreamLine(trimmed);
          receivedRef.current = true;
          enqueueChunk(parsed);
        }
      }
      const finalLine = buffer.trim();
      if (finalLine) {
        const parsed = parseStreamLine(finalLine);
        receivedRef.current = true;
        enqueueChunk(parsed);
      }
    } catch (err) {
      if ((err as Error).name !== 'AbortError') {
        streamFailedRef.current = true;
        setError(err instanceof Error ? err.message : 'tts_stream_error');
      }
    } finally {
      streamEndedRef.current = true;
      if (!streamFailedRef.current && !receivedRef.current &&
!queueRef.current.length && !playingRef.current) {
        setIsStreaming(false);
        setError('tts_empty');
        onWordRange?.(null);
        return;
```

```
      }
      if (!queueRef.current.length && !playingRef.current) {
        setIsStreaming(false);
        onWordRange?.(null);
      }
    }
  }, [enqueueChunk, onWordRange, parseStreamLine, stop]);

  useEffect(() => () => stop(), [stop]);

  return {
    isStreaming,
    isPlaying,
    error,
    start,
    stop,
  };
}
```

react-frontend-app/src/components/ErrorBoundary/ErrorBoundary.tsx

```
import React, { Component, ReactNode, ErrorInfo } from 'react';
import styles from './ErrorBoundary.module.css';
import i18n from '../../i18n';

interface ErrorBoundaryState {
  hasError: boolean;
  error: Error | null;
  errorInfo: ErrorInfo | null;
  showDetails: boolean;
}

interface ErrorBoundaryProps {
  children: ReactNode;
}

class ErrorBoundary extends Component<ErrorBoundaryProps, ErrorBoundaryState> {
  constructor(props: ErrorBoundaryProps) {
    super(props);
    this.state = {
      hasError: false,
      error: null,
      errorInfo: null,
      showDetails: false,
    };
  }

  static getDerivedStateFromError(error: Error): Partial<ErrorBoundaryState> {
    return { hasError: true };
  }

  componentDidCatch(error: Error, errorInfo: ErrorInfo): void {
    this.setState({
      error: error,
      errorInfo: errorInfo,
    });

    // Логування помилки для моніторингу
    console.error('ErrorBoundary caught an error:', error, errorInfo);
  }
```

```
  private handleRetry = (): void => {
    window.location.reload();
  };

  private toggleDetails = (): void => {
    this.setState({ showDetails: !this.state.showDetails });
  };

  render(): ReactNode {
    if (this.state.hasError) {
      return (
        <div className={styles.errorBoundary}>
          <div className={styles.errorContainer}>
            <h2 className={styles.errorTitle}>
              {i18n.t('Something went wrong')}
            </h2>
            <p className={styles.errorMessage}>
              {i18n.t('An error occurred while rendering this component. Try
reloading the page or contact support.')}
            </p>
            <div className={styles.errorActions}>
              <button
                className={styles.retryButton}
                onClick={this.handleRetry}
                type="button"
              >
                {i18n.t('Reload page')}
              </button>
              <button
                className={styles.detailsButton}
                onClick={this.toggleDetails}
                type="button"
              >
                {this.state.showDetails ? i18n.t('Hide details') : i18n.t('Show
details')}
              </button>
            </div>
            {this.state.showDetails && (
              <details className={styles.errorDetails}>
                <summary>{i18n.t('Technical details')}</summary>
                <pre className={styles.errorStack}>
                  {this.state.error && this.state.error.toString()}
                  {this.state.errorInfo?.componentStack}
                </pre>
              </details>
            )}
          </div>
        </div>
      );
    }

    return this.props.children;
  }
}

export default ErrorBoundary;
```

react-frontend-app/src/components/ErrorNotification/ErrorNotification.tsx

```
import React from 'react';
import { ErrorSeverity } from '../../services/error-handler';
import { useErrorNotification } from '../../hooks/useErrorNotification';
import styles from './ErrorNotification.module.css';

interface ErrorNotificationItemProps {
  id: string;
  message: string;
  severity: ErrorSeverity;
  timestamp: number;
  onDismiss: (id: string) => void;
}

const ErrorNotificationItem: React.FC<ErrorNotificationItemProps> = ({
  id,
  message,
  severity,
  timestamp,
  onDismiss,
}) => {
  const getSeverityClass = (severity: ErrorSeverity): string => {
    switch (severity) {
      case ErrorSeverity.LOW:
        return styles.severityLow;
      case ErrorSeverity.MEDIUM:
        return styles.severityMedium;
      case ErrorSeverity.HIGH:
        return styles.severityHigh;
      case ErrorSeverity.CRITICAL:
        return styles.severityCritical;
      default:
        return styles.severityMedium;
    }
  };

  const getSeverityIcon = (severity: ErrorSeverity): string => {
    switch (severity) {
      case ErrorSeverity.LOW:
        return 'ℹ️';
      case ErrorSeverity.MEDIUM:
        return '⚠️';
      case ErrorSeverity.HIGH:
        return '❗';
      case ErrorSeverity.CRITICAL:
        return '🚨';
      default:
        return '⚠️';
    }
  };

  const formatTimestamp = (timestamp: number): string => {
    const date = new Date(timestamp);
    return date.toLocaleTimeString();
  };

  return (
    <div className={`${styles.notification} ${getSeverityClass(severity)}`}
role="alert">
      <div className={styles.notificationContent}>
        <div className={styles.notificationHeader}>
```

```
          <span className={styles.severityIcon}>
            {getSeverityIcon(severity)}
          </span>
          <span className={styles.severityText}>
            {severity.toLowerCase()}
          </span>
          <span className={styles.timestamp}>
            {formatTimestamp(timestamp)}
          </span>
          <button
            className={styles.dismissButton}
            onClick={() => onDismiss(id)}
            aria-label="Dismiss notification"
            type="button"
          >
            ×
          </button>
        </div>
        <div className={styles.notificationMessage}>
          {message}
        </div>
      </div>
    </div>
  );
};

export const ErrorNotificationContainer: React.FC = () => {
  const { notifications, dismissNotification, dismissAllNotifications } =
useErrorNotification();

  if (notifications.length === 0) {
    return null;
  }

  return (
    <div className={styles.notificationContainer} aria-live="assertive">
      <div className={styles.notificationHeader}>
        {notifications.length > 1 && (
          <button
            className={styles.dismissAllButton}
            onClick={dismissAllNotifications}
            type="button"
          >
            Dismiss all ({notifications.length})
          </button>
        )}
      </div>
      <div className={styles.notificationList}>
        {notifications.map((notification) => (
          <ErrorNotificationItem
            key={notification.id}
            id={notification.id}
            message={notification.message}
            severity={notification.severity}
            timestamp={notification.timestamp}
            onDismiss={dismissNotification}
          />
        ))}
      </div>
    </div>
```

```
  );
};
```

react-frontend-app/src/components/SimplificationWidget.tsx

```
import React from 'react';
import styles from './SimplificationWidget.module.css';
import { useTranslation } from 'react-i18next';
import { getLocalizedEmotionLabel } from '../utils/emotion-label';

interface SimplificationWidgetProps {
  onSimplifyOnce: () => void;
  onSimplifyAuto: (steps: number) => void;
  onStop: () => void;
  onNeedHelp?: () => void;
  onPanic?: () => void;
  isLoading?: boolean;
  progress?: { current: number; total: number } | null;
  stableEmotion?: string;
  consensusActive?: boolean;
  showSimplifyButton?: boolean;
}

const SimplificationWidget: React.FC<SimplificationWidgetProps> = ({
  onSimplifyOnce,
  onSimplifyAuto,
  onStop,
  onNeedHelp,
  onPanic,
  isLoading = false,
  progress = null,
  stableEmotion,
  consensusActive = false,
  showSimplifyButton = true,
}) => {
  const { t } = useTranslation();

  const handleSimplifyOnce = (): void => {
    onSimplifyOnce();
  };

  const handleSimplifyAuto = (steps: number): void => {
    onSimplifyAuto(steps);
  };

  const handleStop = (): void => {
    onStop();
  };

  const handleNeedHelp = (): void => {
    onNeedHelp?.();
  };

  const handlePanic = (): void => {
    onPanic?.();
  };

  return (
    <div className={styles.widget} role="region" aria-labelledby="widget-title">
      <div className={styles.header}>
```

```
        <h3 className={styles.title} id="widget-title">
          {t('Iterative simplification')}
        </h3>
      </div>

      <div className={styles.content}>
        <p className={styles.description} id="simplify-description">
          {t('Iterative simplification description')}
        </p>

        <div className={styles.statusRow} aria-live="polite">
          <span className={styles.statusLabel}>{t('Stable Emotion')}:</span>
          <span className={styles.statusValue}>
            {getLocalizedEmotionLabel(stableEmotion, t)}
          </span>
        </div>

        <div className={styles.assistButtons}>
          <button
            type="button"
            className={styles.assistButton}
            onClick={handleNeedHelp}
            disabled={!consensusActive}
            aria-disabled={!consensusActive}
          >
            {t('Need help')}
          </button>
          <button
            type="button"
            className={`${styles.assistButton} ${styles.panicButton}`}
            onClick={handlePanic}
            disabled={!consensusActive}
            aria-disabled={!consensusActive}
          >
            {t('Panic button')}
          </button>
        </div>

        {showSimplifyButton && (
          <div className={styles.simplifyControls}>
            <button
              className={`${styles.simplifyButton} ${isLoading ? styles.loading
: ''}`}
              onClick={handleSimplifyOnce}
              disabled={isLoading}
              aria-label={t('Trigger iterative simplification')}
              aria-describedby="simplify-description"
              type="button"
            >
              {isLoading ? (
                <>
                  <span className={styles.spinner} aria-hidden="true"></span>
                  <span aria-live="polite">
                    {progress ? `Крок ${progress.current}/${progress.total}` :
t('Simplifying')}
                  </span>
                </>
              ) : (
                <>1 крок</>
              )}
```

```
            </button>
            <div className={styles.simplifyRow}>
              <button
                className={styles.secondaryButton}
                onClick={() => handleSimplifyAuto(5)}
                disabled={isLoading}
                type="button"
              >
                5 кроків
              </button>
              <button
                className={styles.stopButton}
                onClick={handleStop}
                disabled={!isLoading}
                type="button"
              >
                Стоп
              </button>
            </div>
          </div>
        )}
      </div>
    </div>
  );
};

export default SimplificationWidget;
```

react-frontend-app/src/components/button/index.tsx

```
import { ButtonProps } from '../../types';
import styles from './index.module.css';

export function Button({ children, onClick, disabled = false, type = 'button',
variant = 'primary', className }: ButtonProps) {
  const buttonClass = `${styles.btn} ${styles[`btn-${variant}`]} ${className ||
''}`;

  return (
    <button
      className={buttonClass}
      onClick={onClick}
      disabled={disabled}
      type={type}
    >
      {children}
    </button>
  );
}
```

react-frontend-app/src/components/editor/Editor.tsx

```
import React, { useMemo, useState, useRef, useEffect } from 'react';
import styles from './Editor.module.css';
import btnStyles from '../button/index.module.css';
import { FEEDBACK_HIGHLIGHT_COLORS } from '../../constants/editor';
import { useSettings } from '../../context/AppProviders';
import { DetectedWords, WordType } from '../../types/api';
import { paginateTextByFit, getPageStatistics, PageInfo } from
'../../utils/textPagination';
import { PageInfo as PageInfoComponent } from '../page-info';
```

```
type DetectionKey =
  | 'profanity'
  | 'abbr'
  | 'slang'
  | 'terms'
  | 'obsolete'
  | 'complex'
  | 'rare';

const HIGHLIGHT_PRIORITY: DetectionKey[] = [
  'profanity',
  'abbr',
  'slang',
  'terms',
  'obsolete',
  'complex',
  'rare',
];

interface WordPart {
  text: string;
  isWord: boolean;
  start: number; // local offset within current page string
  end: number;   // local offset within current page string
}

interface EditorComponentProps {
  initialText?: string;
  detected?: DetectedWords | null;
  wordTypes?: Array<WordType | null> | null;
  activeRange?: { start: number; end: number } | null;
  t?: (key: string) => string; // Translation function
  onPageContextChange?: (ctx: { pageIndex: number; totalPages: number; scope: {
start: number; end: number } }) => void;
  onCursorContextChange?: (ctx: { cursorIndex: number | null }) => void;
}

export function Editor({ initialText = '', detected = null, wordTypes,
activeRange, t, onPageContextChange, onCursorContextChange }:
EditorComponentProps) {
  const { wordMode, highlightMode, highContrast } = useSettings();
  const [page, setPage] = useState<number>(0);
  const [currentWord, setCurrentWord] = useState<number>(0); // індекс поточного
слова

  // Use initialText directly instead of state to allow updates
  const text = initialText;
  const prefersHighContrast =
    typeof window !== 'undefined' &&
    typeof window.matchMedia === 'function' &&
    window.matchMedia('(prefers-contrast: high)').matches;
  const highContrastEnabled =
    Boolean(highContrast) || prefersHighContrast || (typeof document !==
'undefined' && document.documentElement.getAttribute('data-contrast') ===
'high');
  const textBlockRef = useRef<HTMLDivElement>(null);
  const measureRef = useRef<HTMLDivElement | null>(null);
  const rafRef = useRef<number | null>(null);
  const lastCursorIndexRef = useRef<number | null>(null);
```

```
  const [pageInfos, setPageInfos] = useState<PageInfo[]>(() => [{ content: '',
startIndex: 0, endIndex: 0, wordCount: 0, paragraphCount: 0 }]);

  const pages = pageInfos.map(pageInfo => pageInfo.content);
  const pageStats = getPageStatistics(pageInfos);

  // Reset page and current word when text changes
  useEffect(() => {
    setPage(0);
    setCurrentWord(0);
  }, [text]);

  // Height-based pagination: ensure the text block never scrolls.
  useEffect(() => {
    if (!textBlockRef.current) return;

    if (!measureRef.current) {
      const el = document.createElement('div');
      el.setAttribute('aria-hidden', 'true');
      el.style.position = 'fixed';
      el.style.left = '-99999px';
      el.style.top = '0';
      el.style.visibility = 'hidden';
      el.style.pointerEvents = 'none';
      el.style.whiteSpace = 'pre-wrap';
      el.style.wordBreak = 'break-word';
      el.style.overflow = 'hidden';
      document.body.appendChild(el);
      measureRef.current = el;
    }

    const computePages = (): void => {
      const textBlock = textBlockRef.current;
      const measureEl = measureRef.current;
      if (!textBlock || !measureEl) return;

      const { clientWidth, clientHeight } = textBlock;
      if (clientWidth <= 0 || clientHeight <= 0) return;

      const computed = window.getComputedStyle(textBlock);
      measureEl.style.width = `${clientWidth}px`;
      measureEl.style.height = `${clientHeight}px`;
      measureEl.style.fontFamily = computed.fontFamily;
      measureEl.style.fontSize = computed.fontSize;
      measureEl.style.fontWeight = computed.fontWeight;
      measureEl.style.lineHeight = computed.lineHeight;
      measureEl.style.letterSpacing = computed.letterSpacing;
      measureEl.style.padding = computed.padding;
      measureEl.style.boxSizing = computed.boxSizing;

      const fits = (startIndex: number, endIndex: number): boolean => {
        // Avoid empty strings that "fit" but don't advance.
        if (endIndex <= startIndex) return false;
        measureEl.textContent = text.slice(startIndex, endIndex);
        return measureEl.scrollHeight <= measureEl.clientHeight;
      };

      const nextPageInfos = paginateTextByFit(text, fits, { preferParagraphs:
true });
```

```
    setPageInfos(nextPageInfos);

    // Keep current page in bounds after re-pagination
    setPage((p) => Math.min(p, Math.max(0, nextPageInfos.length - 1)));
  };

  computePages();

  const schedule = (): void => {
    if (rafRef.current !== null) {
      cancelAnimationFrame(rafRef.current);
    }
    rafRef.current = requestAnimationFrame(() => {
      rafRef.current = null;
      computePages();
    });
  };

  let ro: ResizeObserver | null = null;
  if (typeof ResizeObserver !== 'undefined') {
    ro = new ResizeObserver(schedule);
    ro.observe(textBlockRef.current);
  } else {
    window.addEventListener('resize', schedule);
  }

  return () => {
    if (ro) {
      ro.disconnect();
    } else {
      window.removeEventListener('resize', schedule);
    }
  };
}, [text]);

useEffect(() => {
  return () => {
    if (rafRef.current !== null) {
      cancelAnimationFrame(rafRef.current);
    }
    if (measureRef.current) {
      measureRef.current.remove();
      measureRef.current = null;
    }
  };
}, []);

// Expose current page scope to parent (for backend scope-based algorithms).
useEffect(() => {
  if (!onPageContextChange) return;
  const current = pageInfos[page];
  if (!current) return;
  onPageContextChange({
    pageIndex: page,
    totalPages: pageInfos.length,
    scope: { start: current.startIndex, end: current.endIndex },
  });
}, [onPageContextChange, page, pageInfos]);

// Розбиваємо текст на частини: слова та не-слова
```

```
  const value = pages[page] || '';
  const currentPageInfo = pageInfos[page];
  const pageGlobalStart = currentPageInfo?.startIndex ?? 0;
  const pageGlobalEnd = currentPageInfo?.endIndex ?? pageGlobalStart +
value.length;

  const detectedRangesByType: Partial<Record<DetectionKey, Array<{ start:
number; end: number }>>> = {};
  if (detected && currentPageInfo) {
    for (const type of HIGHLIGHT_PRIORITY) {
      const entries = detected[type] ?? [];
      const ranges: Array<{ start: number; end: number }> = [];
      for (const entry of entries) {
        if (!entry || typeof entry !== 'object') continue;
        const start = (entry as any).start;
        const end = (entry as any).end;
        if (!Number.isInteger(start) || !Number.isInteger(end)) continue;
        if (start < 0 || end <= start) continue;
        // Keep only ranges that intersect the current page slice.
        if (end <= pageGlobalStart || start >= pageGlobalEnd) continue;
        ranges.push({ start, end });
      }
      if (ranges.length > 0) {
        detectedRangesByType[type] = ranges;
      }
    }
  }

  const getHighlightTypeForRange = (globalStart: number, globalEnd: number):
DetectionKey | null => {
    for (const type of HIGHLIGHT_PRIORITY) {
      const ranges = detectedRangesByType[type];
      if (!ranges) continue;
      for (const r of ranges) {
        // Overlap check
        if (r.end > globalStart && r.start < globalEnd) {
          return type;
        }
      }
    }
    return null;
  };

  const { wordParts, wordIndices, wordsOnPage } = useMemo(() => {
    const parts: WordPart[] = [];
    const wordRegex = /[\p{L}\p{N}''\-]+/gu;
    let lastIndex = 0;
    let match: RegExpExecArray | null;

    while ((match = wordRegex.exec(value)) !== null) {
      if (match.index > lastIndex) {
        parts.push({ text: value.slice(lastIndex, match.index), isWord: false,
start: lastIndex, end: match.index });
      }
      parts.push({ text: match[0], isWord: true, start: match.index, end:
match.index + match[0].length });
      lastIndex = match.index + match[0].length;
    }
    if (lastIndex < value.length) {
      parts.push({ text: value.slice(lastIndex), isWord: false, start:
```

```
lastIndex, end: value.length });
    }

    const indices = parts.map((p, i) => (p.isWord ? i : null)).filter((i): i is
number => i !== null);
    return { wordParts: parts, wordIndices: indices, wordsOnPage: indices.length
};
  }, [value]);

  useEffect(() => {
    if (!onCursorContextChange) return;
    if (!wordMode) {
      lastCursorIndexRef.current = null;
      onCursorContextChange({ cursorIndex: null });
      return;
    }
    if (!currentPageInfo) {
      lastCursorIndexRef.current = null;
      onCursorContextChange({ cursorIndex: null });
      return;
    }
    const partIndex = wordIndices[currentWord];
    if (typeof partIndex !== 'number') {
      lastCursorIndexRef.current = null;
      onCursorContextChange({ cursorIndex: null });
      return;
    }
    const part = wordParts[partIndex];
    if (!part || !part.isWord) {
      lastCursorIndexRef.current = null;
      onCursorContextChange({ cursorIndex: null });
      return;
    }
    const nextCursorIndex = pageGlobalStart + part.start;
    if (lastCursorIndexRef.current === nextCursorIndex) return;
    lastCursorIndexRef.current = nextCursorIndex;
    onCursorContextChange({ cursorIndex: nextCursorIndex });
  }, [currentWord, currentPageInfo, onCursorContextChange, pageGlobalStart,
wordIndices, wordMode, wordParts]);

  // Переміщення по словах з переходом між сторінками
  const moveWord = (direction: 'left' | 'right'): void => {
    setCurrentWord((prev) => {
      if (direction === 'left') {
        if (prev > 0) return prev - 1;
        // Якщо на першому слові і не на першій сторінці — переходимо на
попередню сторінку
        if (page > 0) {
          setPage((p) => p - 1);
          // Після зміни сторінки useEffect скине currentWord на 0, тому треба
повернути останній індекс слова на попередній сторінці
          // Але це зробимо у useEffect нижче
        }
        return 0;
      }
      if (direction === 'right') {
        if (prev < wordsOnPage - 1) return prev + 1;
        // Якщо на останньому слові і не на останній сторінці — переходимо на
наступну сторінку
        if (page < pages.length - 1) {
```

```
          setPage((p) => p + 1);
          // currentWord скинеться на 0 автоматично
        }
        return prev;
      }
      return prev;
    });
  };

  // Скидаємо currentWord на перше слово при зміні сторінки, значення або
активації режиму "по словам"
  useEffect(() => {
    if (wordMode) {
      setCurrentWord(0);
    }
  }, [wordMode, page, value]);

  // Додаємо обробник клавіш для навігації
  useEffect(() => {
    const handleKeyDown = (e: KeyboardEvent): void => {
      // Navigation by words in word mode
      if (wordMode) {
        if (e.key === '1') {
          moveWord('left');
          e.preventDefault();
        } else if (e.key === '2') {
          moveWord('right');
          e.preventDefault();
        }
      }

      // Page navigation (works always)
      if (e.key === 'ArrowLeft' && e.ctrlKey) {
        setPage((p) => Math.max(0, p - 1));
        e.preventDefault();
      } else if (e.key === 'ArrowRight' && e.ctrlKey) {
        setPage((p) => Math.min(pages.length - 1, p + 1));
        e.preventDefault();
      } else if (e.key === 'Home' && e.ctrlKey) {
        setPage(0);
        e.preventDefault();
      } else if (e.key === 'End' && e.ctrlKey) {
        setPage(pages.length - 1);
        e.preventDefault();
      }
    };

    window.addEventListener('keydown', handleKeyDown);
    return () => window.removeEventListener('keydown', handleKeyDown);
  }, [wordMode, wordsOnPage, pages.length, page]);

  // Рендеримо текст з підсвіткою лише якщо є типи та включена підсвітка
  const renderText = wordParts.map((part, i) => {
    if (part.isWord) {
      const wordIdx = wordIndices.indexOf(i);
      const style: React.CSSProperties = {};

      // Підсвітка типу слова
      if (highlightMode && currentPageInfo) {
        const globalStart = pageGlobalStart + part.start;
```

```
        const globalEnd = pageGlobalStart + part.end;
        const detectedType = getHighlightTypeForRange(globalStart, globalEnd);
        if (detectedType) {
          style.background = FEEDBACK_HIGHLIGHT_COLORS[detectedType] ||
undefined;
          style.textDecorationLine = 'underline';
          style.textDecorationThickness = '2px';
          style.textDecorationStyle = 'solid';
          style.textUnderlineOffset = '2px';
          if (highContrastEnabled) {
            style.boxShadow = 'inset 0 0 0 2px var(--color-on-surface)';
          }
        }
        style.borderRadius = '2px';
        style.transition = 'background 0.15s';
      }

      if (activeRange && currentPageInfo) {
        const globalStart = pageGlobalStart + part.start;
        const globalEnd = pageGlobalStart + part.end;
        if (activeRange.end > globalStart && activeRange.start < globalEnd) {
          style.background = 'rgba(255, 215, 0, 0.28)';
          style.boxShadow = 'inset 0 0 0 1px var(--color-primary)';
        }
      }

      // Підсвічування поточного слова в режимі по словам
      if (wordMode && wordIdx === currentWord) {
        style.outline = '2px solid var(--color-primary)';
        style.background = 'rgba(120, 80, 255, 0.18)';
        if (highContrastEnabled) {
          style.background = 'rgba(255, 255, 255, 0.2)';
          style.boxShadow = 'inset 0 0 0 2px var(--color-on-surface)';
        }
      }

      return (
        <span key={i} className={styles.word} style={style}>
          {part.text}
        </span>
      );
    } else {
      return <React.Fragment key={i}>{part.text}</React.Fragment>;
    }
  });

  return (
    <div className={styles.editorWrapper}>
      {/* Page Info Component */}
      {t && pageInfos.length > 1 && (
        <PageInfoComponent
          pageInfos={pageInfos}
          currentPage={page}
          t={t}
          showDetails={pages.length > 3} // Show details for longer documents
        />
      )}

      <div
        className={styles.textarea}
```

```
        style={{ whiteSpace: 'pre-wrap', fontFamily: 'inherit' }}
        ref={textBlockRef}
        tabIndex={0}
        role="document"
        aria-label={t ? t('Document text') : 'Document text'}
        aria-describedby="editor-shortcuts-hint"
      >
        {renderText}
      </div>
      <div className={styles.srOnly} id="editor-shortcuts-hint">
        {t ? t('Editor shortcuts hint') : 'Keyboard shortcuts: Ctrl + left or
right arrow for previous or next page. In word mode, keys 1 and 2 move to the
previous or next word.'}
      </div>
      <div className={styles.navPanel}>
        <div className={styles.navSection}>
          <div className={styles.navTitle}>
            {t ? t('Pages') : 'Сторінки'}
            {pages.length > 5 && (
              <small className={styles.navHint}>
                (Ctrl + ←/→)
              </small>
            )}
          </div>
          <div className={styles.controls}>
            {pages.length > 2 && (
              <button
                className={`${btnStyles.btn} ${styles.squareBtn}
${styles.firstLastBtn}`}
                onClick={() => setPage(0)}
                disabled={page === 0}
                aria-label={t ? t('First page') : 'Перша сторінка'}
                type="button"
                title={t ? t('First page') : 'Перша сторінка'}
              >
                ⇤
              </button>
            )}
            <button
              className={`${btnStyles.btn} ${styles.squareBtn}`}
              onClick={() => setPage((p) => Math.max(0, p - 1))}
              disabled={page === 0}
              aria-label={t ? t('Previous page') : 'Попередня сторінка'}
              type="button"
            >
              ←
            </button>
            <span
              className={styles.pageCounter}
              role="status"
              aria-live="polite"
              aria-label={`${t ? t('Page') : 'Page'} ${page + 1} ${t ? t('of') :
'of'} ${pages.length}`}
            >
              {page + 1} / {pages.length}
            </span>
            <button
              className={`${btnStyles.btn} ${styles.squareBtn}`}
              onClick={() => setPage((p) => Math.min(pages.length - 1, p + 1))}
              disabled={page === pages.length - 1}
```

```
              aria-label={t ? t('Next page') : 'Наступна сторінка'}
              type="button"
            >
              →
            </button>
            {pages.length > 2 && (
              <button
                className={`${btnStyles.btn} ${styles.squareBtn}
${styles.firstLastBtn}`}
                onClick={() => setPage(pages.length - 1)}
                disabled={page === pages.length - 1}
                aria-label={t ? t('Last page') : 'Остання сторінка'}
                type="button"
                title={t ? t('Last page') : 'Остання сторінка'}
              >
                ⇥
              </button>
            )}
          </div>
        </div>
        {wordMode && (
          <div className={styles.navSection}>
            <div className={styles.navTitle}>{t ? t('Words') : 'Words'}</div>
            <div className={styles.wordNav}>
              <button
                className={`${btnStyles.btn} ${styles.squareBtn}`}
                onClick={() => moveWord('left')}
                aria-label={t ? t('Previous word') : 'Previous word'}
                type="button"
              >
                ←
              </button>
              <span
                className={styles.wordCounter}
                role="status"
                aria-live="polite"
                aria-label={`${t ? t('Word') : 'Word'} ${wordsOnPage === 0 ? 0 :
currentWord + 1} ${t ? t('of') : 'of'} ${wordsOnPage}`}
              >
                {wordsOnPage === 0 ? 0 : currentWord + 1} / {wordsOnPage}
              </span>
              <button
                className={`${btnStyles.btn} ${styles.squareBtn}`}
                onClick={() => moveWord('right')}
                aria-label={t ? t('Next word') : 'Next word'}
                type="button"
              >
                →
              </button>
            </div>
          </div>
        )}
      </div>
    </div>
  );
}
```

react-frontend-app/src/components/editor/index.tsx

```
import { Editor as EditorComponent } from './Editor';
import styles from './index.module.css';
import { EditorProps } from '../../types';

export function Editor({ initialText, detected, wordTypes, activeRange,
onPageContextChange, onCursorContextChange }: EditorProps) {
  return (
    <div className={styles.editor}>
      <EditorComponent
        initialText={initialText}
        detected={detected}
        wordTypes={wordTypes}
        activeRange={activeRange}
        onPageContextChange={onPageContextChange}
        onCursorContextChange={onCursorContextChange}
      />
    </div>
  );
}
```

react-frontend-app/src/components/expert-validation/ExpertValidationModal.tsx

```
import { useCallback, useEffect, useMemo, useRef, useState, type KeyboardEvent }
from 'react';
import { useTranslation } from 'react-i18next';
import styles from './ExpertValidationModal.module.css';
import { getPendingReplacements, verifyReplacement, declineReplacement } from
'../../utils/api';

interface PendingReplacement {
  word: string;
  group: string;
  replacements: Array<{ replacement: string }>;
  metadata?: Record<string, any>;
}

interface ExpertValidationModalProps {
  isOpen: boolean;
  onClose: () => void;
}

const GROUP_LABELS: Record<string, string> = {
  slang: 'Сленг',
  rare: 'Рідкісні слова',
  abbr: 'Абревіатури',
  terms: 'Терміни',
  complex_group: 'Складні',
  obsolete_group: 'Застарілі',
  profanity_group: 'Нецензурна лексика',
};

const GROUP_ORDER = [
  'slang',
  'rare',
  'abbr',
  'terms',
  'complex_group',
  'obsolete_group',
  'profanity_group',
];
```

```
export function ExpertValidationModal({ isOpen, onClose }:
ExpertValidationModalProps): JSX.Element | null {
  const { t } = useTranslation();
  const [entries, setEntries] = useState<PendingReplacement[]>([]);
  const [loading, setLoading] = useState<boolean>(false);
  const [error, setError] = useState<string | null>(null);
  const [edits, setEdits] = useState<Record<string, string>>({});
  const [entryErrors, setEntryErrors] = useState<Record<string, string>>({});
  const [loadingKey, setLoadingKey] = useState<string | null>(null);
  const modalRef = useRef<HTMLDivElement | null>(null);
  const closeButtonRef = useRef<HTMLButtonElement | null>(null);
  const lastActiveElementRef = useRef<HTMLElement | null>(null);

  const fetchPending = useCallback(async () => {
    setLoading(true);
    setError(null);
    try {
      const result = await getPendingReplacements();
      const pending = result.pending || [];
      setEntries(pending);
      const nextEdits: Record<string, string> = {};
      for (const entry of pending) {
        const key = `${entry.group}::${entry.word}`;
        const replacements = entry.replacements?.map((r) =>
r.replacement).filter(Boolean) || [];
        nextEdits[key] = replacements.join('\n');
      }
      setEdits(nextEdits);
      setEntryErrors({});
    } catch (err) {
      setError(err instanceof Error ? err.message : t('Load error'));
    } finally {
      setLoading(false);
    }
  }, [t]);

  useEffect(() => {
    if (isOpen) {
      void fetchPending();
    }
  }, [isOpen, fetchPending]);

  useEffect(() => {
    if (!isOpen) return;
    if (typeof document !== 'undefined') {
      const active = document.activeElement;
      lastActiveElementRef.current = active instanceof HTMLElement ? active :
null;
    }
    if (closeButtonRef.current) {
      closeButtonRef.current.focus();
    }
    return () => {
      if (lastActiveElementRef.current) {
        lastActiveElementRef.current.focus();
      }
      lastActiveElementRef.current = null;
    };
  }, [isOpen]);
```

```
  const handleKeyDown = (event: KeyboardEvent<HTMLDivElement>) => {
    if (event.key === 'Escape') {
      event.preventDefault();
      onClose();
      return;
    }
    if (event.key !== 'Tab') return;
    const container = modalRef.current;
    if (!container) return;
    const focusables = Array.from(
      container.querySelectorAll<HTMLElement>(
        'button,[href],input,select,textarea,[tabindex]:not([tabindex="-1"])'
      )
    ).filter((el) => !el.hasAttribute('disabled') && !el.getAttribute('aria-
hidden'));
    if (focusables.length === 0) {
      event.preventDefault();
      return;
    }
    const first = focusables[0];
    const last = focusables[focusables.length - 1];
    if (event.shiftKey) {
      if (document.activeElement === first || document.activeElement ===
container) {
        event.preventDefault();
        last.focus();
      }
    } else if (document.activeElement === last) {
      event.preventDefault();
      first.focus();
    }
  };

  const groupedEntries = useMemo(() => {
    const grouped: Record<string, PendingReplacement[]> = {};
    for (const entry of entries) {
      if (!grouped[entry.group]) grouped[entry.group] = [];
      grouped[entry.group].push(entry);
    }
    return grouped;
  }, [entries]);

  const handleEditChange = (key: string, value: string) => {
    setEdits((prev) => ({ ...prev, [key]: value }));
  };

  const handleApprove = async (entry: PendingReplacement) => {
    const key = `${entry.group}::${entry.word}`;
    if (loadingKey) return;
    const raw = edits[key] || '';
    const replacements = raw
      .split('\n')
      .map((line) => line.trim())
      .filter(Boolean);
    if (!replacements.length) {
      setEntryErrors((prev) => ({ ...prev, [key]: t('Add at least one
replacement') }));
      return;
    }
```

```
    setEntryErrors((prev) => ({ ...prev, [key]: '' }));
    setLoadingKey(key);
    try {
      await verifyReplacement(entry.word, entry.group, replacements);
      setEntries((prev) => prev.filter((item) => !(item.word === entry.word &&
item.group === entry.group)));
    } catch (err) {
      setEntryErrors((prev) => ({ ...prev, [key]: t('Save error') }));
    } finally {
      setLoadingKey(null);
    }
  };

  const handleDecline = async (entry: PendingReplacement) => {
    const key = `${entry.group}::${entry.word}`;
    if (loadingKey) return;
    setEntryErrors((prev) => ({ ...prev, [key]: '' }));
    setLoadingKey(key);
    try {
      await declineReplacement(entry.word, entry.group);
      setEntries((prev) => prev.filter((item) => !(item.word === entry.word &&
item.group === entry.group)));
    } catch (err) {
      setEntryErrors((prev) => ({ ...prev, [key]: t('Save error') }));
    } finally {
      setLoadingKey(null);
    }
  };

  if (!isOpen) return null;

  return (
    <div className={styles.overlay} role="dialog" aria-modal="true" aria-
labelledby="expert-validation-title" onKeyDown={handleKeyDown}>
      <div className={styles.modal} ref={modalRef}>
        <div className={styles.header}>
          <div className={styles.title} id="expert-validation-title">
            {t('Expert validation')}
          </div>
          <button
            type="button"
            className={styles.closeButton}
            onClick={onClose}
            aria-label={t('Close')}
            ref={closeButtonRef}
          >
            ×
          </button>
        </div>
        <div className={styles.content}>
          {loading && <div className={styles.status}>{t('Loading...')}</div>}
          {error && <div className={`${styles.status}
${styles.error}`}>{error}</div>}
          {!loading && !error && entries.length === 0 && (
            <div className={styles.status}>{t('No words to review')}</div>
          )}
          {!loading && !error && entries.length > 0 && (
            <>
              {GROUP_ORDER.filter((group) =>
groupedEntries[group]?.length).map((group) => (
```

```
                <details key={group} className={styles.group} open>
                  <summary className={styles.groupSummary}>
                    <span>{t(GROUP_LABELS[group] || group)}</span>
                    <span
className={styles.groupCount}>{groupedEntries[group].length}</span>
                  </summary>
                  {groupedEntries[group].map((entry) => {
                    const key = `${entry.group}::${entry.word}`;
                    return (
                      <div key={key} className={styles.entry}>
                        <div className={styles.wordRow}>{entry.word}</div>
                        <textarea
                          className={styles.textarea}
                          value={edits[key] || ''}
                          onChange={(e) => handleEditChange(key,
e.target.value)}
                          placeholder={t('Replacement variants, one per line')}
                        />
                        {entryErrors[key] && (
                          <div
className={styles.entryError}>{entryErrors[key]}</div>
                        )}
                        <div className={styles.actions}>
                          <button
                            type="button"
                            className={styles.actionButton}
                            onClick={() => handleApprove(entry)}
                            disabled={loadingKey === key}
                          >
                            {t('Approve')}
                          </button>
                          <button
                            type="button"
                            className={`${styles.actionButton}
${styles.actionSecondary}`}
                            onClick={() => handleDecline(entry)}
                            disabled={loadingKey === key}
                          >
                            {t('Decline')}
                          </button>
                        </div>
                      </div>
                    );
                  })}
                </details>
              ))}
            </>
          )}
        </div>
      </div>
    </div>
  );
}
```

react-frontend-app/src/components/file-info/FileInfo.tsx

```
import React from 'react';
import { useCurrentFile } from '../../context/AppProviders';
import styles from './FileInfo.module.css';
```

```
interface FileInfoProps {
  t: (key: string) => string;
}

export const FileInfo: React.FC<FileInfoProps> = ({ t }) => {
  const currentFile = useCurrentFile();

  // Don't show file info for default example content
  if (!currentFile.originalFileName && !currentFile.uploadedAt) {
    return null;
  }

  const formatFileSize = (sizeInBytes?: number): string => {
    if (!sizeInBytes) return 'Unknown';

    if (sizeInBytes < 1024) {
      return `${sizeInBytes} B`;
    } else if (sizeInBytes < 1024 * 1024) {
      return `${Math.round(sizeInBytes / 1024)} KB`;
    } else {
      return `${Math.round(sizeInBytes / (1024 * 1024) * 100) / 100} MB`;
    }
  };

  const formatUploadTime = (timestamp?: number): string => {
    if (!timestamp) return '';

    const date = new Date(timestamp);
    return date.toLocaleString();
  };

  const getFileExtension = (fileName?: string): string => {
    if (!fileName) return '';

    const parts = fileName.split('.');
    return parts.length > 1 ? parts[parts.length - 1].toUpperCase() : '';
  };

  return (
    <div className={styles.fileInfo}>
      <div className={styles.fileInfoHeader}>
        <span className={styles.fileIcon}></span>
        <div className={styles.fileDetails}>
          <div className={styles.fileName}>
            {currentFile.originalFileName || currentFile.name}
          </div>
          <div className={styles.fileMetadata}>
            <span className={styles.fileExtension}>
              {getFileExtension(currentFile.originalFileName ||
currentFile.name)}
            </span>
            {currentFile.fileSize && (
              <>
                <span className={styles.separator}>•</span>
                <span className={styles.fileSize}>
                  {formatFileSize(currentFile.fileSize)}
                </span>
              </>
            )}
          </div>
```

```
        </div>
      </div>

      {currentFile.uploadedAt && (
        <div className={styles.uploadInfo}>
          <span className={styles.uploadTime}>
            {t('Uploaded')}: {formatUploadTime(currentFile.uploadedAt)}
          </span>
        </div>
      )}

      <div className={styles.contentInfo}>
        <span className={styles.contentLength}>
          {t('Content length')}: {currentFile.content.length} {t('characters')}
        </span>
      </div>
    </div>
  );
};
```

react-frontend-app/src/components/footer/index.tsx

```
import styles from './index.module.css';
import { FooterProps } from '../../types';

export function Footer({ t }: FooterProps) {
  return (
    <footer className={styles.footer}>
      <div className={styles.description}>
        <p className={styles.author}>
          {t('Created by')} <strong>Roman Savitskyi</strong>
        </p>
        <p className={styles.project}>
          {t('Accessibility cognity reader')} &copy; {new Date().getFullYear()}
        </p>
      </div>
      <div className={styles.links}>
        <a
          href='https://github.com/roman-savitskyi/diser/tree/develop/react-
frontend-app'
          target='_blank'
          rel='noopener noreferrer'
          className={styles.link}
        >
          GitHub
        </a>
        <a
          href='https://www.linkedin.com/in/roman-savitskyi-98b49158/'
          target='_blank'
          rel='noopener noreferrer'
          className={styles.link}
        >
          LinkedIn
        </a>
        <a
          href='mailto:Roman.savitskyi@gmail.com'
          rel='noopener noreferrer'
          className={styles.link}
        >
          Email
        </a>
```

```
      </div>
    </footer>
  );
}
```

react-frontend-app/src/components/header/index.tsx

```
import useFetchEmotion from '../../hooks/useFetchEmotion';
import styles from './index.module.css';
import { useTranslation } from 'react-i18next';
import { HeaderProps } from '../../types';
import { ChangeEvent } from 'react';
import { getLocalizedEmotionLabel } from '../../utils/emotion-label';
import { useDebugSettings } from '../../context/AppProviders';
import { useAppLanguage } from '../../context/UISettingsContext';

export function Header({ t }: HeaderProps) {
  const emotionStatus = useFetchEmotion();
  const { i18n } = useTranslation();
  const { options: debugOptions } = useDebugSettings();
  const { current: language, set: setLanguage } = useAppLanguage();

  const handleLanguageChange = (e: ChangeEvent<HTMLSelectElement>): void => {
    const nextLanguage = e.target.value;
    setLanguage(nextLanguage);
    void i18n.changeLanguage(nextLanguage);
  };

  return (
    <header className={styles.header}>
      <div className={styles.container}>
        <h1 className={styles.title}>{t('Accessibility Cognity Reader')}</h1>
        {debugOptions.widgets && (
          <div className={styles.emotionWrapper}>
            <div className={styles.emotionBlock}>
              <span className={styles.emotionLabel}>{t('Stable
Emotion')}:</span>
              <span className={styles.emotionValue}>
                {getLocalizedEmotionLabel(emotionStatus.value, t)}
              </span>
            </div>
          </div>
        )}
        <div className={styles.languageSwitcher}>
          <label htmlFor="lang-switcher"
className={styles.langLabel}>{t('Language')}:</label>
          <select
            id="lang-switcher"
            onChange={handleLanguageChange}
            value={language}
            className={styles.langSelect}
          >
            <option value="ua">{t('Ukrainian')}</option>
            <option value="en">{t('English')}</option>
          </select>
        </div>
      </div>
    </header>
  );
```

```
}
```

react-frontend-app/src/components/index.ts

```
export { Button } from './button/index';
export { Editor } from './editor/index';
export { Footer } from './footer/index';
export { Header } from './header/index';
export { LeftAside } from './left-aside/index';
export { RightAside } from './right-aside/index';
export { default as ErrorBoundary } from './ErrorBoundary';
export { ErrorNotificationContainer } from './ErrorNotification';
export { FileInfo } from './file-info';
export { PageInfo } from './page-info';
export { default as SimplificationWidget } from './SimplificationWidget';
```

react-frontend-app/src/components/left-aside/FeedbackPanel/index.tsx

```
import { useTranslation } from 'react-i18next';
import styles from './index.module.css';
import { FEEDBACK_HIGHLIGHT_COLORS } from '../../../constants/editor';
import { useSettings } from '../../../context/AppProviders';
import { DetectionType } from '../../../types/api';

interface FeedbackType {
  key: DetectionType;
  label: string;
}

interface FeedbackPanelProps {
  score?: Record<string, number>;
}

const FEEDBACK_TYPES: FeedbackType[] = [
  { key: 'profanity', label: 'Profanity or Vulg' },
  { key: 'abbr', label: 'Abbreviation' },
  { key: 'slang', label: 'Slang' },
  { key: 'terms', label: 'Terms' },
  { key: 'obsolete', label: 'Obsolete' },
  { key: 'complex', label: 'Complex' },
  { key: 'rare', label: 'Rare' },
  { key: 'flesch', label: 'FleschKincaid' },
];

export function FeedbackPanel({ score }: FeedbackPanelProps): JSX.Element {
  const { t } = useTranslation();
  const { highlightMode } = useSettings();

  return (
    <div className={styles.feedbackPanel}>
      <div className={styles.title}>{t('Feedback panel')}</div>
      {FEEDBACK_TYPES.map(type => (
        <div className={styles.item} key={type.key}>
          {highlightMode && (
            <span
              className={styles.colorBox}
              style={{
                background: FEEDBACK_HIGHLIGHT_COLORS[type.key],
                display: 'inline-block',
                width: 12,
                height: 12,
                borderRadius: 2,
```

```
                marginRight: 8
              }}
            />
          )}
          {t(type.label)}: <span>{score?.[type.key] || 0}</span>
        </div>
      ))}
    </div>
  );
}
```

react-frontend-app/src/components/left-aside/SettingsPanel/index.tsx

```
import { ChangeEvent, useState } from 'react';
import { useSettings, useDebugSettings } from '../../../context/AppProviders';
import { useTranslation } from 'react-i18next';
import styles from './index.module.css';

export function SettingsPanel(): JSX.Element {
  const { t } = useTranslation();
  const {
    wordMode,
    setWordMode,
    highlightMode,
    setHighlightMode,
    ttsEnabled,
    setTtsEnabled,
    highContrast,
    setHighContrast,
    simplifyScopeMode,
    setSimplifyScopeMode,
    analysisScopeMode,
    setAnalysisScopeMode,
  } = useSettings();
  const { options: debugOptions, setOption: setDebugOption } =
useDebugSettings();
  const [debugOpen, setDebugOpen] = useState<boolean>(false);

  const handleWordModeChange = (e: ChangeEvent<HTMLInputElement>): void => {
    setWordMode(e.target.checked);
  };

  const handleHighlightModeChange = (e: ChangeEvent<HTMLInputElement>): void =>
{
    setHighlightMode(e.target.checked);
  };

  const toggleWordMode = (): void => {
    setWordMode(!wordMode);
  };

  const toggleHighlightMode = (): void => {
    setHighlightMode(!highlightMode);
  };

  const handleTtsEnabledChange = (e: ChangeEvent<HTMLInputElement>): void => {
    setTtsEnabled?.(e.target.checked);
  };

  const toggleTtsEnabled = (): void => {
```

```
    setTtsEnabled?.(!ttsEnabled);
  };

  const handleHighContrastChange = (e: ChangeEvent<HTMLInputElement>): void => {
    setHighContrast?.(e.target.checked);
  };

  const toggleHighContrast = (): void => {
    setHighContrast?.(!highContrast);
  };

  const handleSimplifyScopeRadio = (mode: 'document' | 'page' | 'cursor') => ():
void => {
    setSimplifyScopeMode?.(mode);
  };

  const handleAnalysisScopeRadio = (mode: 'document' | 'page') => (): void => {
    setAnalysisScopeMode?.(mode);
  };

  const handleDebugToggle = (key: keyof typeof debugOptions) => (e:
ChangeEvent<HTMLInputElement>): void => {
    const checked = e.target.checked;
    setDebugOption(key, checked);
  };

  const renderDebugCheckbox = (
    key: keyof typeof debugOptions,
    label: string,
    inputId: string,
  ) => (
    <div className={styles.settingRow} key={key}>
      <input
        id={inputId}
        type="checkbox"
        className={styles.checkbox}
        checked={debugOptions[key]}
        onChange={handleDebugToggle(key)}
      />
      <label className={styles.label} htmlFor={inputId}>
        {label}
      </label>
    </div>
  );

  return (
    <div className={styles.settingsPanel}>
      <div className={styles.title}>{t('Settings')}</div>
      <div className={styles.settingGroup} role="group" aria-label={t('Analysis
scope')}>
        <div className={styles.groupTitle}>{t('Analysis scope')}</div>
        <label className={styles.radioRow}>
          <input
            type="radio"
            name="analysisScope"
            checked={analysisScopeMode === 'page'}
            onChange={handleAnalysisScopeRadio('page')}
          />
          <span className={styles.label}>{t('Current page')}</span>
        </label>
```

```
        <label className={styles.radioRow}>
          <input
            type="radio"
            name="analysisScope"
            checked={analysisScopeMode === 'document'}
            onChange={handleAnalysisScopeRadio('document')}
          />
          <span className={styles.label}>{t('Whole document')}</span>
        </label>
      </div>

      <div className={styles.settingGroup} role="group" aria-
label={t('Simplification scope')}>
        <div className={styles.groupTitle}>{t('Simplification scope')}</div>
        <label className={styles.radioRow}>
          <input
            type="radio"
            name="simplifyScope"
            checked={simplifyScopeMode === 'page'}
            onChange={handleSimplifyScopeRadio('page')}
          />
          <span className={styles.label}>{t('Current page')}</span>
        </label>
        <label className={styles.radioRow}>
          <input
            type="radio"
            name="simplifyScope"
            checked={simplifyScopeMode === 'document'}
            onChange={handleSimplifyScopeRadio('document')}
          />
          <span className={styles.label}>{t('Whole document')}</span>
        </label>
        <label className={styles.radioRow}>
          <input
            type="radio"
            name="simplifyScope"
            checked={simplifyScopeMode === 'cursor'}
            onChange={handleSimplifyScopeRadio('cursor')}
          />
          <span className={styles.label}>{t('From cursor')}</span>
        </label>
      </div>
      <div className={styles.settingRow}>
        <input
          id="wordModeCheckbox"
          type="checkbox"
          className={styles.checkbox}
          checked={wordMode}
          onChange={handleWordModeChange}
        />
        <label
          className={styles.label}
          htmlFor="wordModeCheckbox"
          onClick={toggleWordMode}
          style={{ cursor: 'pointer' }}
        >
          {t('Word mode')}
        </label>
      </div>
      <div className={styles.settingRow}>
```

```
          <input
            id="highlightModeCheckbox"
            type="checkbox"
            className={styles.checkbox}
            checked={highlightMode}
            onChange={handleHighlightModeChange}
          />
          <label
            className={styles.label}
            htmlFor="highlightModeCheckbox"
            onClick={toggleHighlightMode}
            style={{ cursor: 'pointer' }}
          >
            {t('Highlighting')}
          </label>
        </div>
        <div className={styles.settingRow}>
          <input
            id="ttsEnabledCheckbox"
            type="checkbox"
            className={styles.checkbox}
            checked={Boolean(ttsEnabled)}
            onChange={handleTtsEnabledChange}
          />
          <label
            className={styles.label}
            htmlFor="ttsEnabledCheckbox"
            onClick={toggleTtsEnabled}
            style={{ cursor: 'pointer' }}
          >
            {t('Text to speech (TTS)')}
          </label>
        </div>
        <div className={styles.settingRow}>
          <input
            id="highContrastCheckbox"
            type="checkbox"
            className={styles.checkbox}
            checked={Boolean(highContrast)}
            onChange={handleHighContrastChange}
          />
          <label
            className={styles.label}
            htmlFor="highContrastCheckbox"
            onClick={toggleHighContrast}
            style={{ cursor: 'pointer' }}
          >
            {t('High contrast')}
          </label>
        </div>
        <div className={styles.debugSection}>
          <button
            type="button"
            className={`${styles.debugToggle} ${debugOpen ? styles.debugToggleOpen
: ''}`}
            onClick={() => setDebugOpen((prev) => !prev)}
            aria-expanded={debugOpen}
          >
            <span>{t('Debug')}</span>
            <span className={styles.chevron} aria-hidden="true">
```

```
            ▾
          </span>
        </button>
        {debugOpen && (
          <div className={styles.debugContent}>
            {renderDebugCheckbox('widgets', t('Widgets'), 'debugWidgets')}
            {renderDebugCheckbox('camera', t('Stop camera'), 'debugCamera')}
            {renderDebugCheckbox('llm', t('Stop LLM'), 'debugLLM')}
            {renderDebugCheckbox('expertValidation', t('Expert validation'),
'debugExpertValidation')}
          </div>
        )}
      </div>
    </div>
  );
}
```

react-frontend-app/src/components/left-aside/SurveyForm/index.tsx

```
import { FormEvent, type KeyboardEvent } from 'react';
import styles from './index.module.css';
import { useSurveyForm } from '../../../hooks/useSurveyForm';
import { Button } from '../../button';

interface SurveyFormProps {
  t: (key: string) => string;
  fileName: string;
  page: number;
  simplificationLevel: number;
}

export function SurveyForm({ t, fileName, page, simplificationLevel }:
SurveyFormProps): JSX.Element {
  const {
    rating,
    reasons,
    loading,
    submitted,
    error,
    REASONS,
    handleStarClick,
    handleReasonChange,
    handleSubmit
  } = useSurveyForm({ t, fileName, page, simplificationLevel });

  const onSubmit = (e: FormEvent<HTMLFormElement>): void => {
    handleSubmit(e);
  };

  const handleStarKeyDown = (e: KeyboardEvent<HTMLDivElement>): void => {
    if (loading) return;
    if (e.key === 'ArrowRight' || e.key === 'ArrowUp') {
      e.preventDefault();
      const next = Math.min(10, (rating || 0) + 1);
      handleStarClick(next);
    }
    if (e.key === 'ArrowLeft' || e.key === 'ArrowDown') {
      e.preventDefault();
      const next = Math.max(1, (rating || 1) - 1);
      handleStarClick(next);
```

```
    }
    if (e.key === 'Home') {
      e.preventDefault();
      handleStarClick(1);
    }
    if (e.key === 'End') {
      e.preventDefault();
      handleStarClick(10);
    }
  };

  return (
    <form className={styles.surveyForm} onSubmit={onSubmit}>
      <div className={styles.label}>{t('Rate your experience')}:</div>
      <div
        className={styles.stars}
        role="radiogroup"
        aria-label={t('Rate your experience')}
        tabIndex={0}
        onKeyDown={handleStarKeyDown}
      >
        {[...Array(10)].map((_, i) => (
          <button
            key={i}
            type='button'
            className={i < rating ? styles.starActive : styles.star}
            role="radio"
            aria-checked={i + 1 === rating}
            aria-label={`${t('Rate')} ${i + 1}`}
            onClick={() => handleStarClick(i + 1)}
          >
            {'★'}
          </button>
        ))}
      </div>
      {rating > 0 && rating < 7 && (
        <div className={styles.reasons}>
          <div>{t('Please select reasons')}:</div>
          {REASONS.map((reason) => (
            <label key={reason} className={styles.checkboxLabel}>
              <input
                type='checkbox'
                checked={reasons.includes(reason)}
                onChange={() => handleReasonChange(reason)}
              />
              {t(reason)}
            </label>
          ))}
        </div>
      )}
      <Button
        type='submit'
        disabled={loading || rating === 0}
        className={styles.submitButton}
      >
        {t('Save')}
      </Button>
      {error && <div className={styles.errorMsg}>{error}</div>}
      {submitted && <div className={styles.successMsg}>{t('Survey
saved!')}</div>}
```

```
    </form>
  );
}
```

react-frontend-app/src/components/left-aside/index.tsx

```
import { useRef, useState } from 'react';
import styles from './index.module.css';
import { SurveyForm } from './SurveyForm';
import { SettingsPanel } from './SettingsPanel';
import SimplificationWidget from '../SimplificationWidget';
import { LeftAsideProps } from '../../types';
import { iterativeSimplifyCursor, iterativeSimplifyDocument,
iterativeSimplifyPage } from '../../services/iterative-simplify-backend';
import { api } from '../../services/api-client';
import useFetchEmotion from '../../hooks/useFetchEmotion';
import { useDebugSettings, useUISettings } from '../../context/AppProviders';

export function LeftAside({ t, fileName, page, simplificationLevel, dispatch,
currentFile }: LeftAsideProps) {
  const { options: debugOptions } = useDebugSettings();
  const { settings } = useUISettings();
  const stopCamera = debugOptions.camera;
  const showDebugWidgets = debugOptions.widgets;
  const showSimplifyButton = showDebugWidgets && stopCamera;
  const emotionStatus = useFetchEmotion();

  const [isSimplifying, setIsSimplifying] = useState(false);
  const [simplifyProgress, setSimplifyProgress] = useState<{ current: number;
total: number } | null>(null);
  const stopRequestedRef = useRef(false);
  const isAbortError = (err: unknown): boolean => {
    const anyErr = err as any;
    return Boolean(anyErr && (anyErr.name === 'AbortError' || anyErr.code ===
'ABORT_ERR'));
  };

  const getScopeAndMaxChanges = (text: string): { scope?: { start: number; end:
number }; max_changes: number } => {
    const mode = settings.simplifyScopeMode;
    const textLen = text.length;

    let scope: { start: number; end: number } | undefined;
    let max_changes = 5;

    if (mode === 'document') {
      scope = { start: 0, end: textLen };
      max_changes = 20;
      return { scope, max_changes };
    }

    if (mode === 'cursor') {
      if (typeof currentFile.cursorIndex === 'number' && currentFile.cursorIndex
>= 0 && currentFile.cursorIndex < textLen) {
        scope = { start: currentFile.cursorIndex, end: textLen };
        max_changes = 10;
        return { scope, max_changes };
      }
      if (currentFile.pageScopeEnd > currentFile.pageScopeStart) {
        scope = { start: currentFile.pageScopeStart, end:
```

```
currentFile.pageScopeEnd };
      }
      return { scope, max_changes };
    }

    if (currentFile.pageScopeEnd > currentFile.pageScopeStart) {
      scope = { start: currentFile.pageScopeStart, end: currentFile.pageScopeEnd
};
    }
    return { scope, max_changes };
  };

  const simplify = async (text: string, steps: number) => {
    const mode = settings.simplifyScopeMode;
    const { scope, max_changes } = getScopeAndMaxChanges(text);

    const config = { abortKey: 'iterative-simplify' };
    const common = {
      text,
      max_changes,
      steps,
      allow_llm: false,
      include_analysis: true,
      include_cai: true,
      syntax: { enabled: true, max_iterations: 1, min_improvement: 0.01 },
    };

    const res =
      mode === 'document'
        ? await iterativeSimplifyDocument(common, config)
        : mode === 'cursor' && typeof currentFile.cursorIndex === 'number' &&
currentFile.cursorIndex >= 0 && currentFile.cursorIndex <= text.length
          ? await iterativeSimplifyCursor({ ...common, cursor_index:
currentFile.cursorIndex }, config)
          : scope
            ? await iterativeSimplifyPage({ ...common, scope }, config)
            : await iterativeSimplifyDocument(common, config);

    if (stopRequestedRef.current) {
      return res;
    }

    dispatch({
      content: res.text,
      lastSimplify: {
        algorithm_id: res.algorithm_id,
        algorithm_version: res.algorithm_version,
        scope: res.scope ?? null,
        limits_used: res.limits_used,
        timings_ms: res.timings_ms,
        steps_requested: res.steps_requested,
        steps_performed: res.steps_performed,
        applied_count: res.applied_count,
        total_applied_count: res.total_applied_count,
        applied_changes: res.applied_changes,
        iterations: res.iterations,
        mode: (res.mode ?? mode) as any,
        createdAt: Date.now(),
      },
      lastSimplifyStatus: 'ok',
```

```
  });

  return res;
};

const handleSimplifyOnce = async (): Promise<void> => {
  if (isSimplifying) return;
  const currentText = currentFile.content;
  if (!currentText.trim()) return;

  stopRequestedRef.current = false;
  setIsSimplifying(true);
  setSimplifyProgress({ current: 1, total: 1 });
  dispatch({ lastSimplifyStatus: 'running' });

  try {
    await simplify(currentText, 1);
  } catch (error) {
    if (isAbortError(error) || stopRequestedRef.current) {
      dispatch({ lastSimplifyStatus: 'canceled' });
    } else {
      dispatch({ lastSimplifyStatus: 'error' });
      console.error('Помилка спрощення тексту:', error);
    }
  } finally {
    setIsSimplifying(false);
    setSimplifyProgress(null);
  }
};

const handleSimplifyAuto = async (steps: number): Promise<void> => {
  if (isSimplifying) return;
  const initialText = currentFile.content;
  if (!initialText.trim()) return;

  stopRequestedRef.current = false;
  setIsSimplifying(true);
  setSimplifyProgress({ current: 0, total: steps });
  dispatch({ lastSimplifyStatus: 'running' });

  try {
    if (stopRequestedRef.current) return;
    setSimplifyProgress({ current: steps, total: steps });
    await simplify(initialText, steps);
  } catch (error) {
    if (isAbortError(error) || stopRequestedRef.current) {
      dispatch({ lastSimplifyStatus: 'canceled' });
    } else {
      dispatch({ lastSimplifyStatus: 'error' });
      console.error('Помилка спрощення тексту:', error);
    }
  } finally {
    setIsSimplifying(false);
    setSimplifyProgress(null);
  }
};

const handleStop = (): void => {
  stopRequestedRef.current = true;
  api.abort('iterative-simplify');
```

```
    dispatch({ lastSimplifyStatus: 'canceled' });
    setIsSimplifying(false);
    setSimplifyProgress(null);
  };

  const handleNeedHelp = (): void => {
    window.dispatchEvent(new CustomEvent('emotion-help-requested', {
      detail: { emotion: emotionStatus.value, state: emotionStatus.slowState },
    }));
    console.info('Emotion help requested', emotionStatus);
  };

  const handlePanic = (): void => {
    window.dispatchEvent(new CustomEvent('emotion-panic', {
      detail: { emotion: emotionStatus.value, state: emotionStatus.slowState },
    }));
    console.warn('Emotion panic button triggered', emotionStatus);
  };

  return (
    <aside className={styles.leftSidebar}>
      <SimplificationWidget
        onSimplifyOnce={handleSimplifyOnce}
        onSimplifyAuto={handleSimplifyAuto}
        onStop={handleStop}
        onNeedHelp={handleNeedHelp}
        onPanic={handlePanic}
        isLoading={isSimplifying}
        progress={simplifyProgress}
        stableEmotion={emotionStatus.value}
        consensusActive={emotionStatus.consensusActive}
        showSimplifyButton={showSimplifyButton}
      />

      <SurveyForm t={t} fileName={fileName} page={page}
simplificationLevel={simplificationLevel} />
      <SettingsPanel />
    </aside>
  );
}
```

react-frontend-app/src/components/page-info/PageInfo.tsx

```
import React from 'react';
import { PageInfo as PageInfoType, getPageStatistics } from
'../../utils/textPagination';
import styles from './PageInfo.module.css';

interface PageInfoProps {
  pageInfos: PageInfoType[];
  currentPage: number;
  t: (key: string) => string;
  showDetails?: boolean;
}

export const PageInfo: React.FC<PageInfoProps> = ({
  pageInfos,
  currentPage,
  t,
  showDetails = false
```

```
}) => {
  const stats = getPageStatistics(pageInfos);
  const currentPageInfo = pageInfos[currentPage];

  if (!currentPageInfo) return null;

  return (
    <div className={styles.pageInfo}>
      <div className={styles.basicInfo}>
        <div className={styles.pageNumber}>
          <span className={styles.label}>{t('Page')}</span>
          <span className={styles.value}>
            {currentPage + 1} / {stats.totalPages}
          </span>
        </div>

        <div className={styles.progress}>
          <div
            className={styles.progressBar}
            style={{
              width: `${((currentPage + 1) / stats.totalPages) * 100}%`
            }}
          />
        </div>

        <div className={styles.percentage}>
          {Math.round(((currentPage + 1) / stats.totalPages) * 100)}%
        </div>
      </div>

      {showDetails && (
        <div className={styles.detailsInfo}>
          <div className={styles.currentPageStats}>
            <div className={styles.stat}>
              <span className={styles.statLabel}>{t('Words on page')}:</span>
              <span
className={styles.statValue}>{currentPageInfo.wordCount}</span>
            </div>
            <div className={styles.stat}>
              <span className={styles.statLabel}>{t('Characters')}:</span>
              <span
className={styles.statValue}>{currentPageInfo.content.length}</span>
            </div>
            {currentPageInfo.paragraphCount > 1 && (
              <div className={styles.stat}>
                <span className={styles.statLabel}>{t('Paragraphs')}:</span>
                <span
className={styles.statValue}>{currentPageInfo.paragraphCount}</span>
              </div>
            )}
          </div>

          <div className={styles.totalStats}>
            <div className={styles.stat}>
              <span className={styles.statLabel}>{t('Total words')}:</span>
              <span className={styles.statValue}>{stats.totalWords}</span>
            </div>
            <div className={styles.stat}>
              <span className={styles.statLabel}>{t('Reading time')}:</span>
              <span className={styles.statValue}>
```

```
              {stats.estimatedReadingTime} {t('min')}
            </span>
          </div>
          <div className={styles.stat}>
            <span className={styles.statLabel}>{t('Avg words/page')}:</span>
            <span
className={styles.statValue}>{stats.averageWordsPerPage}</span>
          </div>
        </div>
      </div>
    )}
  </div>
 );
};
```

react-frontend-app/src/components/right-aside/AccessibilityPanel/index.tsx

```
import React from 'react';
import { useTranslation } from 'react-i18next';
import styles from './index.module.css';

interface AccessibilityData {
  cai: number;
  category: string;
  components: {
    lexical: {
      lexical_score: number;
    };
    syntactic: {
      syntactic_score: number;
    };
    readability: {
      readability_score: number;
    };
  };
}

interface AccessibilityPanelProps {
  data?: AccessibilityData;
}

export function AccessibilityPanel({ data }: AccessibilityPanelProps):
JSX.Element {
  const { t } = useTranslation();
  const formatValue = (value: number) => {
    return value ? value.toFixed(3) : '0.000';
  };

  const getCategoryColor = (cai: number) => {
    if (cai >= 0.8) return '#4CAF50'; // Відмінна - зелений
    if (cai >= 0.6) return '#8BC34A'; // Хороша - світло-зелений
    if (cai >= 0.4) return '#FF9800'; // Помірна - помаранчевий
    if (cai >= 0.2) return '#FF5722'; // Низька - червоний
    return '#9E9E9E'; // Дуже складний - сірий
  };

  const getCategoryLabel = (category?: string) => {
    if (!category) return null;
    const normalized = category.trim().toLowerCase();
    const categoryMap: Record<string, string> = {
      excellent: t('Excellent'),
```

```
    good: t('Good'),
    moderate: t('Moderate'),
    low: t('Low'),
    very_complex: t('Very complex'),
    'very complex': t('Very complex'),
  };
  return categoryMap[normalized] || category;
};

return (
  <div className={styles.accessibilityPanel}>
    <div className={styles.title}>{t('Accessibility coefficient')}</div>

    <div className={styles.item}>
      <div className={styles.itemLeft}>
        <span className={styles.itemLabel}>CAI:</span>
      </div>
      <span
        className={`${styles.itemValue} ${styles.caiValue}`}
        style={{ color: getCategoryColor(data?.cai || 0) }}
      >
        {formatValue(data?.cai || 0)}
      </span>
    </div>

    <div className={styles.item}>
      <div className={styles.itemLeft}>
        <span className={styles.itemLabel}>{t('Lexical')}:</span>
      </div>
      <span className={styles.itemValue}>
        {formatValue(data?.components?.lexical?.lexical_score || 0)}
      </span>
    </div>

    <div className={styles.item}>
      <div className={styles.itemLeft}>
        <span className={styles.itemLabel}>{t('Syntactic')}:</span>
      </div>
      <span className={styles.itemValue}>
        {formatValue(data?.components?.syntactic?.syntactic_score || 0)}
      </span>
    </div>

    <div className={styles.item}>
      <div className={styles.itemLeft}>
        <span className={styles.itemLabel}>{t('Readability')}:</span>
      </div>
      <span className={styles.itemValue}>
        {formatValue(data?.components?.readability?.readability_score || 0)}
      </span>
    </div>

    {data?.category && (
      <div className={styles.categoryIndicator}>
        <span
          className={styles.categoryLabel}
          style={{ color: getCategoryColor(data?.cai || 0) }}
        >
          {getCategoryLabel(data.category)}
        </span>
```

```
        </div>
      )}
    </div>
  );
}
```

react-frontend-app/src/components/right-aside/FeedbackPanel/index.tsx

```
import { useTranslation } from 'react-i18next';
import styles from './index.module.css';
import { FEEDBACK_HIGHLIGHT_COLORS } from '../../../constants/editor';
import { useSettings, useTextAnalysis } from '../../../context/AppProviders';
import { DetectionType } from '../../../types/api';
import { useCurrentFile } from '../../../context/AppProviders';

interface FeedbackType {
  key: DetectionType;
  label: string;
  showHighlight?: boolean;
}

interface FeedbackPanelProps {
  score?: Record<string, number>;
}

const LEXICAL_TYPES: FeedbackType[] = [
  { key: 'profanity', label: 'Profanity or Vulg', showHighlight: true },
  { key: 'abbr', label: 'Abbreviation', showHighlight: true },
  { key: 'slang', label: 'Slang', showHighlight: true },
  { key: 'terms', label: 'Terms', showHighlight: true },
  { key: 'obsolete', label: 'Obsolete', showHighlight: true },
  { key: 'complex', label: 'Complex', showHighlight: true },
  { key: 'rare', label: 'Rare', showHighlight: true },
];

const SYNTACTIC_TYPES: FeedbackType[] = [
  { key: 'sentence_structure', label: 'Sentence structure' },
  { key: 'clause_simplicity', label: 'Clause simplicity' },
  { key: 'word_order', label: 'Word order (SVO)' },
];

const READABILITY_TYPES: FeedbackType[] = [
  { key: 'flesch', label: 'FleschKincaid' },
  { key: 'smog', label: 'SMOG' },
  { key: 'dlevel', label: 'D-Level' },
];

const renderFeedbackItem = (type: FeedbackType, score?: Record<string, number>,
highlightMode?: boolean, t?: any) => (
  <div className={styles.item} key={type.key}>
    <div className={styles.itemLeft}>
      {highlightMode && type.showHighlight ? (
        <span
          className={styles.colorBox}
          style={{
            background: FEEDBACK_HIGHLIGHT_COLORS[type.key],
            display: 'inline-block',
            width: 12,
            height: 12,
            borderRadius: 2,
```

```
            marginRight: 8
          }}
        />
      ) : (
        <span className={styles.spacer} />
      )}
      <span className={styles.itemLabel}>{t(type.label)}:</span>
    </div>
    <span className={styles.itemValue}>{score?.[type.key] || 0}</span>
  </div>
);

const renderSection = (
  title: string,
  types: FeedbackType[],
  score?: Record<string, number>,
  highlightMode?: boolean,
  t?: any,
) => (
  <div className={styles.section}>
    <div className={styles.sectionTitle}>{t(title)}</div>
    {types.length > 0 ? (
      types.map(type => renderFeedbackItem(type, score, highlightMode, t))
    ) : (
      <div className={styles.emptyState}>{t('No data')}</div>
    )}
  </div>
);

export function FeedbackPanel({ score }: FeedbackPanelProps): JSX.Element {
  const { t } = useTranslation();
  const { highlightMode } = useSettings();
  const { accessibilityData } = useTextAnalysis();
  const currentFile = useCurrentFile();
  const last = currentFile.lastSimplify;
  const lastStatus = currentFile.lastSimplifyStatus;

  const fmtMs = (v: number | undefined): string => (typeof v === 'number' &&
Number.isFinite(v) ? `${Math.round(v)}ms` : '—');

  const handleExportLastRun = (): void => {
    const payload = {
      exportedAt: new Date().toISOString(),
      file: {
        name: currentFile.name,
        page: currentFile.page,
        pageIndex: currentFile.pageIndex,
        pageScopeStart: currentFile.pageScopeStart,
        pageScopeEnd: currentFile.pageScopeEnd,
        cursorIndex: currentFile.cursorIndex,
        contentLength: currentFile.content?.length ?? 0,
      },
      lastSimplifyStatus: lastStatus,
      lastSimplify: last,
      score: score ?? null,
      accessibilityData: accessibilityData ?? null,
    };

    const safeName = (currentFile.name || 'export').replace(/[^\w.\-]+/g, '_');
    const ts = new Date().toISOString().replace(/[:.]/g, '-');
```

```
    const filename = `${safeName}.simplify-run.${ts}.json`;
    const blob = new Blob([JSON.stringify(payload, null, 2)], { type:
'application/json' });
    const url = URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.href = url;
    a.download = filename;
    document.body.appendChild(a);
    a.click();
    a.remove();
    URL.revokeObjectURL(url);
  };

  return (
    <div className={styles.feedbackPanel}>
      <div className={styles.title}>
        {t('Feedback panel')}
        <span className={styles.subtitle}>
          {lastStatus === 'running'
            ? ' (simplifying…)'
            : lastStatus === 'canceled'
              ? ' (canceled)'
              : lastStatus === 'error'
                ? ' (error)'
                : ' (ready)'}
        </span>
      </div>
      {last && (
        <div className={styles.section}>
          <div className={styles.sectionTitle}>{t('Last iteration')}</div>
          <div className={styles.lastIterMeta}>
            {t('Algorithm')}: {last.algorithm_id ?? '—'}
v{last.algorithm_version ?? '—'} · {t('Mode')}: {last.mode} · {t('Steps')}:
{last.steps_performed ?? '—'}/{last.steps_requested ?? '—'} · {t('Changes last
step')}: {last.applied_count} · {t('Changes total')}: {last.total_applied_count
?? last.applied_count}
          </div>
          <div className={styles.lastIterMeta}>
            {t('Limits')}: max_changes={last.limits_used?.max_changes ?? '—'},
steps={last.limits_used?.steps ?? '—'} · {t('Time')}: total
{fmtMs(last.timings_ms?.total)}, analysis {fmtMs(last.timings_ms?.analysis)}
          </div>
          <button type="button" className={styles.exportButton}
onClick={handleExportLastRun}>
            {t('Export JSON')}
          </button>
          {last.applied_changes.length > 0 && (
            <details className={styles.lastIterDetails}>
              <summary>{t('Show replacements')}</summary>
              <ul className={styles.lastIterList}>
                {last.applied_changes.slice(0, 10).map((c, idx) => (
                  <li key={`${c.type}:${c.start}:${c.end}:${idx}`}
className={styles.lastIterItem}>
                    <span className={styles.lastIterType}>{c.type}</span>{' '}
                    <span className={styles.lastIterFrom}>"{c.from}"</span> →{'
'}
                    <span className={styles.lastIterTo}>"{c.to}"</span>
                  </li>
                ))}
              </ul>
```

```
              {last.applied_changes.length > 10 && (
                <div className={styles.lastIterMore}>… {t('more')}
{last.applied_changes.length - 10}</div>
              )}
            </details>
          )}
        </div>
      )}
      {renderSection('Lexical parameters', LEXICAL_TYPES, score, highlightMode,
t)}
      {renderSection('Syntactic parameters', SYNTACTIC_TYPES, score,
highlightMode, t)}
      {renderSection('Readability parameters', READABILITY_TYPES, score,
highlightMode, t)}
    </div>
  );
}
```

react-frontend-app/src/components/right-aside/index.tsx

```
import { KeyboardEvent, useCallback, useEffect, useMemo, useState } from
'react';
import { ALLOWED_EXTENSIONS, MAX_FILE_SIZE } from '../../constants/file';
import styles from './index.module.css';
import { useDragAndDrop } from './useDragAndDrop';
import { useFileUpload } from './useFileUpload';
import { FeedbackPanel } from './FeedbackPanel';
import { AccessibilityPanel } from './AccessibilityPanel';
import { FileInfo } from '../file-info';
import { useCurrentFile, useDebugSettings, useAppActions, useSettings } from
'../../context/AppProviders';
import { RightAsideProps } from '../../types';
import btnStyles from '../button/index.module.css';
import { useTtsPlayer } from '../../hooks/useTtsPlayer';
import { getWordAtCursor, splitTextIntoSentenceChunks } from '../../utils/tts';
import { getTransformText } from '../../services/text-transformation';

export function RightAside({ t, score, accessibilityData }: RightAsideProps) {
  const { loading, error, upload } = useFileUpload(t);
  const currentFile = useCurrentFile();
  const { setCurrentFile } = useAppActions();
  const { wordMode, ttsEnabled } = useSettings();
  const { options: debugOptions } = useDebugSettings();
  const [inputError, setInputError] = useState<string | null>(null);
  const textForTts = getTransformText(currentFile.content || '');

  const handleUpload = (file: File): void => {
    upload(file);
  };

  const [ttsLiveText, setTtsLiveText] = useState<string>('');

  const handleWordRange = useCallback((range: { start: number; end: number } |
null) => {
    setCurrentFile({ ttsActiveRange: range });
    if (range && range.start >= 0 && range.end > range.start) {
      setTtsLiveText(textForTts.slice(range.start, range.end));
    } else {
      setTtsLiveText('');
    }
```

```
  }, [setCurrentFile, textForTts]);

  const { isStreaming, isPlaying, error: ttsError, start, stop } =
useTtsPlayer({
    onWordRange: handleWordRange,
  });

  const { dragActive, inputRef, handleChange, handleDrag, handleDrop,
handleZoneClick } = useDragAndDrop(handleUpload);


  const handleKeyDown = (e: KeyboardEvent<HTMLDivElement>): void => {
    if (e.key === 'Enter' || e.key === ' ') {
      handleZoneClick();
    }
  };

  const handleTtsStart = useCallback(() => {
    setInputError(null);
    const text = textForTts || '';
    if (!text.trim()) {
      setInputError('No text to read');
      return;
    }

    if (wordMode) {
      const wordChunk = getWordAtCursor(text, currentFile.cursorIndex);
      if (wordChunk) {
        start([wordChunk]);
        return;
      }
    }

    const scopeStart =
      currentFile.pageScopeEnd > currentFile.pageScopeStart
        ? currentFile.pageScopeStart
        : 0;
    const scopeEnd =
      currentFile.pageScopeEnd > currentFile.pageScopeStart
        ? currentFile.pageScopeEnd
        : text.length;
    const pageText = text.slice(scopeStart, scopeEnd);
    const chunks = splitTextIntoSentenceChunks(pageText, scopeStart);
    if (!chunks.length) {
      setInputError('No text to read');
      return;
    }
    start(chunks);
  }, [currentFile, start, wordMode]);

  const handleTtsStop = useCallback(() => {
    stop();
    setCurrentFile({ ttsActiveRange: null });
    setTtsLiveText('');
  }, [setCurrentFile, stop]);

  useEffect(() => {
    if (!ttsEnabled) {
      stop();
      setCurrentFile({ ttsActiveRange: null });
```

```
      setTtsLiveText('');
    }
  }, [ttsEnabled, setCurrentFile, stop]);

  const ttsErrorMessage = useMemo(() => {
    if (inputError) {
      return t(inputError);
    }
    if (!ttsError) return null;
    if (ttsError.startsWith('tts_http_')) return t('TTS server error');
    if (ttsError === 'tts_stream_error') return t('tts_stream_error');
    if (ttsError === 'tts_empty') return t('tts_empty');
    if (ttsError === 'no_text') return t('No text to read');
    return ttsError;
  }, [inputError, ttsError, t]);

  return (
    <aside className={styles.rightSidebar}>
      <div
        className={`${styles.dropZone} ${dragActive ? styles.dropZoneActive :
''}`}
        onClick={handleZoneClick}
        onDragEnter={handleDrag}
        onDragOver={handleDrag}
        onDragLeave={handleDrag}
        onDrop={handleDrop}
        tabIndex={0}
        role='button'
        aria-label={t('Text check')}
        aria-describedby="dropzone-instructions"
        onKeyDown={handleKeyDown}
      >
        <input
          type='file'
          accept={ALLOWED_EXTENSIONS.map((ext) => '.' +
ext.toLowerCase()).join(',')}
          ref={inputRef}
          onChange={handleChange}
          disabled={loading}
          className={styles.fileInput}
        />
        <h2>{t('Text check')}</h2>
        <p id="dropzone-instructions">{t('Drag and drop your file here or click
to select a file.')}</p>
        <div className={styles.dropZoneInfo}>
          <div>
            {t('Max file size')}: {MAX_FILE_SIZE / (1024 * 1024)}MB
          </div>
          <div>
            {t('Supported formats')}: {ALLOWED_EXTENSIONS.join(', ')}
          </div>
          <div className={styles.formatNote}>
            <small>
              * TXT files are parsed locally, PDF/DOC/DOCX require backend
processing
            </small>
          </div>
        </div>
        {loading && <div className={styles.statusMsg} aria-
live="polite">{t('Uploading...')}</div>}
```

```
        {error && (
          <div className={`${styles.statusMsg} ${styles.error}`} role="alert">
            {t('Not saved')}: {error}
          </div>
        )}
        {!loading && !error && currentFile.originalFileName && (
          <div className={`${styles.statusMsg} ${styles.success}`} aria-
live="polite">
            <span role="img" aria-label={t('Success')}>✅</span> {t('File
uploaded and parsed successfully')}
          </div>
        )}
      </div>
      {ttsEnabled && (
        <div className={styles.ttsPanel}>
          <div className={styles.ttsHeader}>
            <h3>{t('Read aloud')}</h3>
            {(isStreaming || isPlaying) && (
              <span className={styles.listeningBadge}>{t('Playing')}</span>
            )}
          </div>
          <div className={styles.ttsMode}>
            {t('Mode')}: {wordMode ? t('Cursor word') : t('Current page')}
          </div>
          <div className={styles.ttsControls}>
            {!isStreaming && !isPlaying ? (
              <button
                className={`${btnStyles.btn} ${styles.ttsButton}`}
                onClick={handleTtsStart}
                type="button"
                aria-pressed={false}
              >
                <span role="img" aria-label={t('Read aloud')}>🔊</span>
{t('Start reading')}
              </button>
            ) : (
              <button
                className={`${btnStyles.btn} ${styles.ttsButton}`}
                onClick={handleTtsStop}
                type="button"
                aria-pressed
              >
                <span role="img" aria-label={t('Stop')}>⏹</span> {t('Stop')}
              </button>
            )}
          </div>
          {ttsLiveText && (
            <div className={styles.statusMsg} aria-live="polite">
              {t('Now reading')}: {ttsLiveText}
            </div>
          )}
          {ttsErrorMessage && (
            <div className={`${styles.statusMsg} ${styles.error}`} role="alert">
              {ttsErrorMessage}
            </div>
          )}
        </div>
      )}
      <FileInfo t={t} />
      {debugOptions.widgets && (
```

```
        <>
          <AccessibilityPanel data={accessibilityData} />
          <FeedbackPanel score={score} />
        </>
      )}
    </aside>
  );
}
```

react-frontend-app/src/components/right-aside/useDragAndDrop.tsx

```
import { useRef, useState, type ChangeEvent, type DragEvent } from 'react';

export function useDragAndDrop(onFile?: (file: File) => void) {
  const [dragActive, setDragActive] = useState(false);
  const inputRef = useRef<HTMLInputElement>(null);

  const handleChange = (e: ChangeEvent<HTMLInputElement>) => {
    const files = e.target.files;
    const file = files && files[0];
    if (file && onFile) onFile(file);
  };

  const handleDrag = (e: DragEvent<HTMLElement>) => {
    e.preventDefault();
    e.stopPropagation();
    if (e.type === 'dragenter' || e.type === 'dragover') setDragActive(true);
    else if (e.type === 'dragleave') setDragActive(false);
  };

  const handleDrop = (e: DragEvent<HTMLElement>) => {
    e.preventDefault();
    e.stopPropagation();
    setDragActive(false);
    if (e.dataTransfer.files && e.dataTransfer.files[0] && onFile) {
      onFile(e.dataTransfer.files[0]);
    }
  };

  const handleZoneClick = () => {
    inputRef.current?.click?.();
  };

  return {
    dragActive,
    inputRef,
    handleChange,
    handleDrag,
    handleDrop,
    handleZoneClick,
  };
}
```

react-frontend-app/src/components/right-aside/useFileUpload.tsx

```
import { ALLOWED_EXTENSIONS, ALLOWED_TYPES, MAX_FILE_SIZE } from
'../../constants/file';
import { uploadFileApi } from '../../utils/api';
import { useFileUpload as useFileUploadState, useFileUploadActions } from
'../../context/AppProviders';
import { FileUploadResponse } from '../../types/api';
```

```
import { debugLog, debugError, debugWarn, detectApiStructure } from
'../../utils/debug';

export function useFileUpload(t: (key: string) => string) {
  const { loading, error } = useFileUploadState();
  const { setFileUploadLoading, setFileUploadSuccess, setFileUploadError } =
useFileUploadActions();

  const validate = (file: File): string => {
    if (!file) return t('No file selected');
    if (!ALLOWED_TYPES.includes(file.type)) return t('Allowed formats') + `:
${ALLOWED_EXTENSIONS.join(', ')}`;
    if (file.size > MAX_FILE_SIZE) return t('Max file size') + `:
${MAX_FILE_SIZE / (1024 * 1024)}MB`;
    return '';
  };

  // Helper function to read text files locally as fallback
  const readTextFile = (file: File): Promise<string> => {
    return new Promise((resolve, reject) => {
      const reader = new FileReader();
      reader.onload = (e) => {
        const content = (e.target?.result as string | null) ?? '';
        resolve(content || '');
      };
      reader.onerror = () => reject(new Error('Failed to read file'));
      reader.readAsText(file, 'UTF-8');
    });
  };

  const upload = async (file: File): Promise<void> => {
    const validationError = validate(file);
    if (validationError) {
      setFileUploadError(validationError);
      return;
    }

    setFileUploadLoading(true);
    setFileUploadError(null);

    try {
      // For TXT files, try local reading as fallback if backend parsing fails
      if (file.type === 'text/plain') {
        try {
          const result: FileUploadResponse = await uploadFileApi(file, t);

          // Enhanced debugging
          debugLog('Upload API Response', result);
          debugLog('API Structure Analysis', detectApiStructure(result));

          // Check different possible response structures
          if (result.success && result.data?.content) {
            setFileUploadSuccess(
              result.data.content,
              result.data.originalName || file.name,
              result.data.size || file.size,
              result.data.type || file.type
            );
            return;
          } else if (result.text) {
```

```
            const text = result.text ?? result.data ??result.content;
            setFileUploadSuccess(text, file.name, file.size, file.type);
            return;
          }
        } catch (apiError) {
          debugWarn('Backend parsing failed for TXT file, trying local reading',
apiError);
        }

        // Fallback: read text file locally
        try {
          const content = await readTextFile(file);
          setFileUploadSuccess(content, file.name, file.size, file.type);
          return;
        } catch (readError) {
          const msg = readError instanceof Error ? readError.message :
String(readError);
          throw new Error(`${t('Failed to parse file content')}: ${msg}`);
        }
      }

      // For other file types, require backend processing
      const result: FileUploadResponse = await uploadFileApi(file, t);

      // Enhanced debugging
      debugLog('Upload API Response (Non-TXT)', result);
      debugLog('API Structure Analysis (Non-TXT)', detectApiStructure(result));
      // Fallback: якщо є поле text, але не спрацював жоден if
      if (result.text && typeof result.text === 'string') {
        setFileUploadSuccess(result.text, file.name, file.size, file.type);
        return;
      }
      // Check different possible response structures
      if (result.success && result.data?.content) {
        // Expected structure with parsed content
        setFileUploadSuccess(
          result.data.content,
          result.data.originalName || file.name,
          result.data.size || file.size,
          result.data.type || file.type
        );
      } else if (result.text) {
        // Backend returns text in 'text' field (actual backend response)
        setFileUploadSuccess(result.text, file.name, file.size, file.type);
      } else if (result.data && typeof result.data === 'string') {
        // If data is directly a string (alternate backend response)
        setFileUploadSuccess(
          result.data,
          file.name,
          file.size,
          file.type
        );
      } else if (result.content) {
        // If content is at root level
        setFileUploadSuccess(
          result.content,
          file.name,
          file.size,
          file.type
        );
```

```
      } else {
        // Log the structure for debugging and provide helpful error
        debugError('Unexpected API response structure', result);
        const errorMsg = result.message ||
          `${t('Failed to parse file content')}. Response:
${detectApiStructure(result)}`;
        throw new Error(errorMsg);
      }
    } catch (error: any) {
      debugError('File upload error', error);
      const errorMessage = error instanceof Error ? error.message : t('API
error');
      setFileUploadError(errorMessage);
    }
  };

  return {
    loading,
    error,
    upload,
    // Helper function for validation without upload
    validate: (file: File): boolean => validate(file) === '',
  };
}
```

react-frontend-app/src/types/api.ts

```
// API Response Types
export interface DetectionResult {
  words: Array<string | DetailedWord>;
  found_count: number;
}

export interface BatchAnalyzeScope {
  start: number;
  end: number;
}

export interface BatchAnalyzeResponse {
  scope: BatchAnalyzeScope;

  profanity?: DetectionResult;
  abbr?: DetectionResult;
  slang?: DetectionResult;
  terms?: DetectionResult;
  obsolete?: DetectionResult;
  complex?: DetectionResult;
  rare?: DetectionResult;
  flesch?: DetectionResult;

  cai?: AccessibilityData;
}

// Детальна інформація про слово з морфологією
export interface DetailedWord {
  text: string;
  lemma?: string;
  word?: string; // для зворотної сумісності
  start?: number;
  end?: number;
```

```
  pos?: string;
  category?: string;
  upos?: string;
  morph_features?: {
    [key: string]: string;
  };
  score?: number;
  entity_group?: string;
  has_term_features?: boolean;
  type?: string;
  confident?: boolean;
  original_prediction?: string;
}

export interface DetectionResponse {
  profanity?: DetectionResult;
  abbr?: DetectionResult;
  slang?: DetectionResult;
  terms?: DetectionResult;
  obsolete?: DetectionResult;
  complex?: DetectionResult;
  rare?: DetectionResult;
  flesch?: DetectionResult;
  smog?: DetectionResult;
  dlevel?: DetectionResult;
}

// File Upload Types
export interface FileUploadResponse {
  success?: boolean;
  message?: string;
  text?: string; // Actual backend response format
  data?: {
    filename?: string;
    originalName?: string;
    size?: number;
    type?: string;
    url?: string;
    content?: string; // Alternative content field
    pages?: number; // Number of pages in the document
  };
  content?: string; // Alternative content location
}

// Text Analysis Types
export interface WordType {
  type: string;
  word?: string;
  lemma?: string;
  text?: string;
}

export interface TextAnalysisResponse {
  words: WordType[];
  score: Record<string, number>;
}

export type SimplifyRunStatus = 'idle' | 'running' | 'canceled' | 'error' |
'ok';
```

```
export interface AppliedChange {
  type: string;
  start: number;
  end: number;
  from: string;
  to: string;
}

export interface LastSimplifyResult {
  algorithm_id?: string;
  algorithm_version?: number;
  experiment_config?: Record<string, any>;
  scope: { start: number; end: number } | null;
  limits_used?: { max_changes: number; steps: number };
  timings_ms?: { total: number; analysis: number; iterations: number[] };
  steps_requested?: number;
  steps_performed?: number;
  applied_count: number;
  total_applied_count?: number;
  applied_changes: AppliedChange[];
  mode: 'document' | 'page' | 'cursor' | 'custom';
  iterations?: Array<{ applied_count: number; applied_changes: AppliedChange[]
}>;
  createdAt: number;
}

// Settings Types
export interface SettingsContextValue {
  wordMode: boolean;
  setWordMode: (val: boolean) => void;
  highlightMode: boolean;
  setHighlightMode: (val: boolean) => void;
  ttsEnabled?: boolean;
  setTtsEnabled?: (val: boolean) => void;
  simplifyScopeMode?: 'document' | 'page' | 'cursor';
  setSimplifyScopeMode?: (mode: 'document' | 'page' | 'cursor') => void;
  analysisScopeMode?: 'document' | 'page';
  setAnalysisScopeMode?: (mode: 'document' | 'page') => void;
}

// Component Props Types
export interface LeftAsideProps {
  t: (key: string) => string;
  fileName: string;
  page: number;
  simplificationLevel: number;
  dispatch: (file: Partial<{
    name: string;
    page: number;
    pageIndex: number;
    pageScopeStart: number;
    pageScopeEnd: number;
    cursorIndex: number | null;
    ttsActiveRange: { start: number; end: number } | null;
    lastSimplify: LastSimplifyResult | null;
    lastSimplifyStatus: SimplifyRunStatus;
    simplificationLevel: number;
    content: string;
    originalFileName?: string;
    fileSize?: number;
```

```
    fileType?: string;
    uploadedAt?: number;
  }>) => void;
  currentFile: {
    name: string;
    page: number;
    pageIndex: number;
    pageScopeStart: number;
    pageScopeEnd: number;
    cursorIndex: number | null;
    ttsActiveRange: { start: number; end: number } | null;
    lastSimplify: LastSimplifyResult | null;
    lastSimplifyStatus: SimplifyRunStatus;
    simplificationLevel: number;
    content: string;
    originalFileName?: string;
    fileSize?: number;
    fileType?: string;
    uploadedAt?: number;
  };
}

export interface AccessibilityData {
  cai: number;
  category: string;
  components: {
    lexical: {
      lexical_score: number;
    };
    syntactic: {
      syntactic_score: number;
      factors?: {
        sentence_structure_score?: number;
        clause_score?: number;
        svo_ratio?: number;
      };
    };
    readability: {
      readability_score: number;
    };
  };
}

export interface RightAsideProps {
  t: (key: string) => string;
  score: Record<string, number>;
  accessibilityData?: AccessibilityData;
}

export interface HeaderProps {
  t: (key: string) => string;
}

export interface FooterProps {
  t: (key: string) => string;
}

// Emotion Detection Types
export interface EmotionResponse {
  emotion: string | null;
```

```
  state?: string | null;
  emotionFast?: string | null;
  emotionSlow?: string | null;
  stateFast?: string | null;
  stateSlow?: string | null;
  negativeConsensus?: boolean;
  scores?: Record<string, Record<string, number>>;
  stateScores?: Record<string, Record<string, number>>;
  confidence?: number;
}

// Survey Form Types
export interface SurveyFormData {
  date: string;
  rating: number;
  reasons: string[];
  fileName?: string;
  page?: number;
  simplificationLevel?: number;
}

// API Error Types
export class ApiError extends Error {
  status?: number;
  code?: string;

  constructor({ message, status, code }: { message: string; status?: number;
code?: string }) {
    super(message);
    this.name = 'ApiError';
    this.status = status;
    this.code = code;
  }
}

// Utility Types
export type DetectionType =
  | 'profanity'
  | 'abbr'
  | 'slang'
  | 'terms'
  | 'obsolete'
  | 'complex'
  | 'rare'
  | 'flesch'
  | 'smog'
  | 'dlevel'
  | 'sentence_structure'
  | 'clause_simplicity'
  | 'word_order';

export interface DetectedWords {
  [key: string]: Array<string | DetailedWord>;
}
```

react-frontend-app/src/types/css-modules.d.ts

```
declare module "*.module.css" {
  const classes: { [key: string]: string };
  export default classes;
```

```
}

declare module "*.module.scss" {
  const classes: { [key: string]: string };
  export default classes;
}

declare module "*.module.sass" {
  const classes: { [key: string]: string };
  export default classes;
}

declare module "*.css" {
  const classes: { [key: string]: string };
  export default classes;
}
```

react-frontend-app/src/types/editor.ts

```
import { DetectedWords, WordType } from './api';

export interface EditorPageContext {
  pageIndex: number;
  totalPages: number;
  scope: { start: number; end: number };
}

export interface EditorCursorContext {
  cursorIndex: number | null;
}

export interface EditorProps {
  initialText: string;
  detected?: DetectedWords | null;
  wordTypes?: Array<WordType | null> | null;
  activeRange?: { start: number; end: number } | null;
  onPageContextChange?: (ctx: EditorPageContext) => void;
  onCursorContextChange?: (ctx: EditorCursorContext) => void;
}
```

react-frontend-app/src/types/index.ts

```
import type { ReactNode } from 'react';

// Re-export all types from api.ts
export * from './api';
export * from './editor';

// Additional utility types
export interface BaseProps {
  className?: string;
  children?: ReactNode;
}

export interface ButtonProps extends BaseProps {
  onClick?: () => void;
  disabled?: boolean;
  type?: 'button' | 'submit' | 'reset';
  variant?: 'primary' | 'secondary' | 'danger';
}

export interface ModalProps extends BaseProps {
```

```
  isOpen: boolean;
  onClose: () => void;
  title?: string;
}

export interface LoadingState {
  isLoading: boolean;
  error?: string | null;
}

export interface PaginationProps {
  currentPage: number;
  totalPages: number;
  onPageChange: (page: number) => void;
}

// Form Types
export interface FormField {
  name: string;
  label: string;
  type: string;
  required?: boolean;
  placeholder?: string;
}

export interface FormErrors {
  [key: string]: string | undefined;
}

// File Types
export interface FileInfo {
  name: string;
  size: number;
  type: string;
  lastModified: number;
}

// Theme Types
export interface ThemeConfig {
  primaryColor: string;
  secondaryColor: string;
  backgroundColor: string;
  textColor: string;
}
```

react-frontend-app/src/utils/api.ts

```
import { api } from '../services/api-client';
import { ApiError, FileUploadResponse, SurveyFormData } from '../types/api';

// Generic API response interface
interface ApiResponse<T = unknown> {
  success: boolean;
  data?: T;
  message?: string;
  error?: string;
}

// Legacy API function - kept for backward compatibility
export const apiLegacy = async <T = unknown>(name: string, data: Record<string,
```

```
unknown>): Promise<ApiResponse<T>> => {
  return api.post<ApiResponse<T>>(name, data);
};

export const uploadFileApi = async (file: File, t: (key: string) => string):
Promise<FileUploadResponse> => {
  try {
    return await api.uploadFile<FileUploadResponse>('upload', file);
  } catch (error) {
    if (error instanceof ApiError) {
      throw new ApiError({
        message: t('Network error'),
        status: error.status,
      });
    }
    console.error('Error while uploading file:', error);
    throw error;
  }
};

interface SurveyApiResponse {
  success: boolean;
  message?: string;
  surveyId?: string;
}

export const sendSurveyApi = async (data: SurveyFormData, t: (key: string) =>
string): Promise<SurveyApiResponse> => {
  try {
    return await api.post<SurveyApiResponse>('/survey', data);
  } catch (error) {
    if (error instanceof ApiError) {
      throw new ApiError({
        message: t('Network error'),
        status: error.status,
      });
    }
    console.error('Error while sending survey:', error);
    throw error;
  }
};

export interface PendingReplacementEntry {
  word: string;
  group: string;
  replacements: Array<{ replacement: string }>;
  metadata?: Record<string, any>;
}

export interface PendingReplacementsResponse {
  total: number;
  pending: PendingReplacementEntry[];
}

export const getPendingReplacements = async ():
Promise<PendingReplacementsResponse> => {
  return api.get<PendingReplacementsResponse>('moderation/pending', { skipCache:
true });
};
```

```
export const verifyReplacement = async (
  word: string,
  group: string,
  replacements?: string[]
): Promise<{ success: boolean }> => {
  return api.post<{ success: boolean }>('moderation/verify', { word, group,
replacements });
};

export const declineReplacement = async (
  word: string,
  group: string
): Promise<{ success: boolean }> => {
  return api.post<{ success: boolean }>('moderation/decline', { word, group });
};
```

react-frontend-app/src/utils/debug.ts

```
// Debug utilities for development

export const isDevelopment = process.env.NODE_ENV === 'development';

export const debugLog = (message: string, data?: any): void => {
  if (isDevelopment) {
    // console.log(`[DEBUG] ${message}`, data);
  }
};

export const debugError = (message: string, error?: any): void => {
  if (isDevelopment) {
    // console.error(`[DEBUG ERROR] ${message}`, error);
  }
};

export const debugWarn = (message: string, data?: any): void => {
  if (isDevelopment) {
    // console.warn(`[DEBUG WARN] ${message}`, data);
  }
};

// Function to safely stringify objects for logging
export const safeStringify = (obj: any): string => {
  try {
    return JSON.stringify(obj, null, 2);
  } catch (error) {
    return `[Unable to stringify: ${error.message}]`;
  }
};

// Function to detect backend API structure
export const detectApiStructure = (response: any): string => {
  const structure = {
    hasSuccess: 'success' in response,
    hasData: 'data' in response,
    hasContent: 'content' in response,
    hasMessage: 'message' in response,
    dataType: response.data ? typeof response.data : 'undefined',
    rootKeys: Object.keys(response || {}),
  };
```

```
  return safeStringify(structure);
};
```

react-frontend-app/src/utils/emotion-label.ts

```
import { TFunction } from 'i18next';

const EMOTION_KEY_MAP: Record<string, string> = {
  angry: 'emotion.angry',
  disgust: 'emotion.disgust',
  fear: 'emotion.fear',
  happy: 'emotion.happy',
  sad: 'emotion.sad',
  surprise: 'emotion.surprise',
  neutral: 'emotion.neutral',
};

export const getLocalizedEmotionLabel = (
  emotion: string | null | undefined,
  t: TFunction,
): string => {
  if (!emotion) {
    return t('Not detected');
  }

  const key = EMOTION_KEY_MAP[emotion.toLowerCase()];
  return key ? t(key) : emotion;
};
```

react-frontend-app/src/utils/textPagination.ts

```
// Text pagination utilities for smart page splitting

export interface PaginationOptions {
  charLimit: number;
  preferParagraphs: boolean;
  minParagraphLength: number;
  maxParagraphLength: number;
}

export interface PageInfo {
  content: string;
  startIndex: number;
  endIndex: number;
  wordCount: number;
  paragraphCount: number;
}

export interface FitPaginationOptions {
  preferParagraphs: boolean;
}

const DEFAULT_OPTIONS: PaginationOptions = {
  charLimit: 500, // Increased from 500 for better readability
  preferParagraphs: true,
  minParagraphLength: 50,
  maxParagraphLength: 2000,
};

const DEFAULT_FIT_OPTIONS: FitPaginationOptions = {
  preferParagraphs: true,
};
```

```
/**
 * Smart text splitting that prioritizes paragraph boundaries
 */
export function splitTextSmartly(text: string, options:
Partial<PaginationOptions> = {}): PageInfo[] {
  const opts = { ...DEFAULT_OPTIONS, ...options };
  const pages: PageInfo[] = [];

  if (!text.trim()) {
    return [createPageInfo('', 0, 0)];
  }

  // First, try to split by paragraphs
  if (opts.preferParagraphs) {
    const paragraphPages = splitByParagraphs(text, opts);
    if (paragraphPages.length > 0) {
      return paragraphPages;
    }
  }

  // Fallback to sentence splitting
  return splitBySentences(text, opts);
}

function lowerBound(arr: number[], value: number): number {
  let lo = 0;
  let hi = arr.length;
  while (lo < hi) {
    const mid = Math.floor((lo + hi) / 2);
    if (arr[mid] <= value) {
      lo = mid + 1;
    } else {
      hi = mid;
    }
  }
  return lo;
}

function collectBreakpoints(text: string): {
  paragraph: number[];
  sentence: number[];
  word: number[];
} {
  const paragraph: number[] = [];
  const sentence: number[] = [];
  const word: number[] = [];

  // Paragraph boundary: blank line(s)
  const paragraphRegex = /\n\s*\n+/g;
  let match: RegExpExecArray | null;
  while ((match = paragraphRegex.exec(text)) !== null) {
    paragraph.push(match.index + match[0].length);
  }

  // Sentence boundary: punctuation + whitespace
  const sentenceRegex = /[.!?…]+\s+/g;
  while ((match = sentenceRegex.exec(text)) !== null) {
    sentence.push(match.index + match[0].length);
  }
```

```
  // Word-ish boundary: any whitespace
  const wordRegex = /\s+/g;
  while ((match = wordRegex.exec(text)) !== null) {
    word.push(match.index + match[0].length);
  }

  const end = text.length;
  if (paragraph[paragraph.length - 1] !== end) paragraph.push(end);
  if (sentence[sentence.length - 1] !== end) sentence.push(end);
  if (word[word.length - 1] !== end) word.push(end);

  return { paragraph, sentence, word };
}

function findMaxFittingBreakpoint(
  startIndex: number,
  breakpoints: number[],
  fits: (start: number, end: number) => boolean
): number | null {
  const left = lowerBound(breakpoints, startIndex);
  if (left >= breakpoints.length) return null;

  const firstEnd = breakpoints[left];
  if (firstEnd <= startIndex) return null;
  if (!fits(startIndex, firstEnd)) return null;

  let lo = left;
  let hi = breakpoints.length - 1;
  let best = firstEnd;

  while (lo <= hi) {
    const mid = Math.floor((lo + hi) / 2);
    const end = breakpoints[mid];
    if (end <= startIndex) {
      lo = mid + 1;
      continue;
    }
    if (fits(startIndex, end)) {
      best = end;
      lo = mid + 1;
    } else {
      hi = mid - 1;
    }
  }

  return best;
}

function findMaxFittingCharEnd(
  text: string,
  startIndex: number,
  fits: (start: number, end: number) => boolean
): number {
  const minEnd = Math.min(text.length, startIndex + 1);
  if (minEnd <= startIndex) return text.length;

  if (!fits(startIndex, minEnd)) {
    return minEnd;
  }
```

```
  let lo = minEnd;
  let hi = text.length;
  let best = minEnd;

  while (lo <= hi) {
    const mid = Math.floor((lo + hi) / 2);
    if (mid <= startIndex) {
      lo = mid + 1;
      continue;
    }
    if (fits(startIndex, mid)) {
      best = mid;
      lo = mid + 1;
    } else {
      hi = mid - 1;
    }
  }

  return best;
}

/**
 * Paginate text using a caller-provided "fits" predicate (e.g. height-based
pagination).
 * Returns PageInfo ranges with indices that always refer to the original input
string.
 */
export function paginateTextByFit(
  text: string,
  fits: (startIndex: number, endIndex: number) => boolean,
  options: Partial<FitPaginationOptions> = {}
): PageInfo[] {
  const opts = { ...DEFAULT_FIT_OPTIONS, ...options };

  if (!text) {
    return [createPageInfoRaw('', 0, 0)];
  }

  const breakpoints = collectBreakpoints(text);
  const pages: PageInfo[] = [];

  let startIndex = 0;
  while (startIndex < text.length) {
    let endIndex: number | null = null;

    if (opts.preferParagraphs) {
      endIndex = findMaxFittingBreakpoint(startIndex, breakpoints.paragraph,
fits);
    }

    if (endIndex === null) {
      endIndex = findMaxFittingBreakpoint(startIndex, breakpoints.sentence,
fits);
    }

    if (endIndex === null) {
      endIndex = findMaxFittingBreakpoint(startIndex, breakpoints.word, fits);
    }
```

```
    if (endIndex === null) {
      endIndex = findMaxFittingCharEnd(text, startIndex, fits);
    }

    if (endIndex <= startIndex) {
      endIndex = Math.min(text.length, startIndex + 1);
    }

    pages.push(createPageInfoRaw(text.slice(startIndex, endIndex), startIndex,
endIndex));
    startIndex = endIndex;
  }

  return pages.length > 0 ? pages : [createPageInfoRaw(text, 0, text.length)];
}

/**
 * Split text by paragraphs, respecting character limits
 */
function splitByParagraphs(text: string, options: PaginationOptions): PageInfo[]
{
  const pages: PageInfo[] = [];

  // Split by double newlines or line breaks that indicate paragraphs
  const paragraphs = text.split(/\n\s*\n|\r\n\s*\r\n/).filter(p =>
p.trim().length > 0);

  let currentPage = '';
  let currentStartIndex = 0;
  let processedLength = 0;

  for (let i = 0; i < paragraphs.length; i++) {
    const paragraph = paragraphs[i].trim();

    // Skip empty paragraphs
    if (!paragraph) continue;

    // If single paragraph is too long, split it further
    if (paragraph.length > options.maxParagraphLength) {
      // Save current page if it has content
      if (currentPage.trim()) {
        pages.push(createPageInfo(currentPage.trim(), currentStartIndex,
processedLength));
        currentPage = '';
        currentStartIndex = processedLength;
      }

      // Split long paragraph by sentences
      const sentencePages = splitLongParagraph(paragraph, options,
processedLength);
      pages.push(...sentencePages);
      processedLength += paragraph.length + 2; // +2 for paragraph separator
      currentStartIndex = processedLength;
      continue;
    }

    // Check if adding this paragraph would exceed limit
    const testPage = currentPage ? currentPage + '\n\n' + paragraph : paragraph;

    if (testPage.length <= options.charLimit) {
```

```
      // Add to current page
      currentPage = testPage;
    } else {
      // Save current page and start new one
      if (currentPage.trim()) {
        pages.push(createPageInfo(currentPage.trim(), currentStartIndex,
processedLength));
      }

      currentPage = paragraph;
      currentStartIndex = processedLength;
    }

    processedLength += paragraph.length + 2; // +2 for paragraph separator
  }

  // Add final page
  if (currentPage.trim()) {
    pages.push(createPageInfo(currentPage.trim(), currentStartIndex,
processedLength));
  }

  return pages.length > 0 ? pages : [createPageInfo(text, 0, text.length)];
}

/**
 * Split long paragraph by sentences
 */
function splitLongParagraph(paragraph: string, options: PaginationOptions,
startOffset: number): PageInfo[] {
  const pages: PageInfo[] = [];
  const sentences = paragraph.match(/[^.!?…]+[.!?…]?/g) || [paragraph];

  let currentPage = '';
  let currentStartIndex = startOffset;
  let processedLength = startOffset;

  for (const sentence of sentences) {
    const trimmedSentence = sentence.trim();
    if (!trimmedSentence) continue;

    const testPage = currentPage ? currentPage + ' ' + trimmedSentence :
trimmedSentence;

    if (testPage.length <= options.charLimit) {
      currentPage = testPage;
    } else {
      if (currentPage.trim()) {
        pages.push(createPageInfo(currentPage.trim(), currentStartIndex,
processedLength));
      }
      currentPage = trimmedSentence;
      currentStartIndex = processedLength;
    }

    processedLength += sentence.length;
  }

  if (currentPage.trim()) {
    pages.push(createPageInfo(currentPage.trim(), currentStartIndex,
```

```
processedLength));
  }

  return pages;
}

/**
 * Fallback: split by sentences (legacy behavior)
 */
function splitBySentences(text: string, options: PaginationOptions): PageInfo[]
{
  const pages: PageInfo[] = [];
  const sentences = text.match(/[^.!?…]+[.!?…]?/g) || [text];

  let currentPage = '';
  let currentStartIndex = 0;
  let processedLength = 0;

  for (const sentence of sentences) {
    const trimmedSentence = sentence.trim();
    if (!trimmedSentence) continue;

    if (trimmedSentence.length > options.charLimit) {
      // Save current page
      if (currentPage.trim()) {
        pages.push(createPageInfo(currentPage.trim(), currentStartIndex,
processedLength));
        currentPage = '';
        currentStartIndex = processedLength;
      }

      // Split long sentence by words
      const wordPages = splitLongSentence(trimmedSentence, options,
processedLength);
      pages.push(...wordPages);
      processedLength += sentence.length;
      currentStartIndex = processedLength;
      continue;
    }

    const testPage = currentPage ? currentPage + ' ' + trimmedSentence :
trimmedSentence;

    if (testPage.length <= options.charLimit) {
      currentPage = testPage;
    } else {
      if (currentPage.trim()) {
        pages.push(createPageInfo(currentPage.trim(), currentStartIndex,
processedLength));
      }
      currentPage = trimmedSentence;
      currentStartIndex = processedLength;
    }

    processedLength += sentence.length;
  }

  if (currentPage.trim()) {
    pages.push(createPageInfo(currentPage.trim(), currentStartIndex,
processedLength));
```

```
  }

  return pages.length > 0 ? pages : [createPageInfo(text, 0, text.length)];
}

/**
 * Split very long sentence by words
 */
function splitLongSentence(sentence: string, options: PaginationOptions,
startOffset: number): PageInfo[] {
  const pages: PageInfo[] = [];
  const words = sentence.split(/\s+/);

  let currentPage = '';
  let currentStartIndex = startOffset;
  let processedLength = startOffset;

  for (const word of words) {
    const testPage = currentPage ? currentPage + ' ' + word : word;

    if (testPage.length <= options.charLimit) {
      currentPage = testPage;
    } else {
      if (currentPage.trim()) {
        pages.push(createPageInfo(currentPage.trim(), currentStartIndex,
processedLength));
      }
      currentPage = word;
      currentStartIndex = processedLength;
    }

    processedLength += word.length + 1; // +1 for space
  }

  if (currentPage.trim()) {
    pages.push(createPageInfo(currentPage.trim(), currentStartIndex,
processedLength));
  }

  return pages;
}

/**
 * Create page info object
 */
function createPageInfo(content: string, startIndex: number, endIndex: number):
PageInfo {
  const wordCount = content.split(/\s+/).filter(word => word.length > 0).length;
  const paragraphCount = content.split(/\n\s*\n/).length;

  return {
    content: content.trim(),
    startIndex,
    endIndex,
    wordCount,
    paragraphCount,
  };
}

function createPageInfoRaw(content: string, startIndex: number, endIndex:
```

```
number): PageInfo {
  const wordCount = content.split(/\s+/).filter(word => word.length > 0).length;
  const paragraphCount = content.split(/\n\s*\n/).length;

  return {
    content,
    startIndex,
    endIndex,
    wordCount,
    paragraphCount,
  };
}

/**
 * Calculate reading time estimate
 */
export function estimateReadingTime(text: string, wordsPerMinute: number = 200):
number {
  const wordCount = text.split(/\s+/).filter(word => word.length > 0).length;
  return Math.ceil(wordCount / wordsPerMinute);
}

/**
 * Get page statistics
 */
export function getPageStatistics(pages: PageInfo[]): {
  totalPages: number;
  totalWords: number;
  totalCharacters: number;
  averageWordsPerPage: number;
  estimatedReadingTime: number;
} {
  const totalWords = pages.reduce((sum, page) => sum + page.wordCount, 0);
  const totalCharacters = pages.reduce((sum, page) => sum + page.content.length,
0);

  return {
    totalPages: pages.length,
    totalWords,
    totalCharacters,
    averageWordsPerPage: pages.length > 0 ? Math.round(totalWords /
pages.length) : 0,
    estimatedReadingTime: estimateReadingTime(pages.map(p => p.content).join('
')),
  };
}
```

react-frontend-app/src/utils/tts.ts

```
export interface TtsChunkInput {
  text: string;
  start: number;
  end: number;
}

const WORD_REGEX = /[\p{L}\p{N}''\-]+/gu;

export function getWordAtCursor(text: string, cursorIndex: number | null):
TtsChunkInput | null {
  if (cursorIndex === null || cursorIndex < 0 || cursorIndex >= text.length)
```

```
return null;
  WORD_REGEX.lastIndex = 0;
  let match: RegExpExecArray | null;
  let lastBefore: TtsChunkInput | null = null;
  while ((match = WORD_REGEX.exec(text)) !== null) {
    const start = match.index;
    const end = start + match[0].length;
    if (cursorIndex >= start && cursorIndex < end) {
      return {
        text: match[0],
        start,
        end,
      };
    }
    if (start > cursorIndex) {
      return {
        text: match[0],
        start,
        end,
      };
    }
    lastBefore = {
      text: match[0],
      start,
      end,
    };
  }
  return lastBefore;
}

export function splitTextIntoSentenceChunks(text: string, globalStart: number):
TtsChunkInput[] {
  const chunks: TtsChunkInput[] = [];
  const sentenceRegex = /[.!?…]+[\s\n]+/g;
  let lastIndex = 0;
  let match: RegExpExecArray | null;

  while ((match = sentenceRegex.exec(text)) !== null) {
    const end = match.index + match[0].length;
    const raw = text.slice(lastIndex, end);
    const leading = raw.match(/^\s+/)?.[0].length ?? 0;
    const trailing = raw.match(/\s+$/)?.[0].length ?? 0;
    const trimmed = raw.trim();
    if (trimmed) {
      chunks.push({
        text: trimmed,
        start: globalStart + lastIndex + leading,
        end: globalStart + end - trailing,
      });
    }
    lastIndex = end;
  }

  if (lastIndex < text.length) {
    const raw = text.slice(lastIndex);
    const leading = raw.match(/^\s+/)?.[0].length ?? 0;
    const trailing = raw.match(/\s+$/)?.[0].length ?? 0;
    const trimmed = raw.trim();
    if (trimmed) {
      chunks.push({
```

```
        text: trimmed,
        start: globalStart + lastIndex + leading,
        end: globalStart + text.length - trailing,
      });
    }
  }

  return chunks;
}
```

react-frontend-app/src/constants/editor.ts

```
export const PAGE_CHAR_LIMIT = 1500; // Increased for better readability

export const FEEDBACK_HIGHLIGHT_COLORS = {
  profanity: 'rgba(255, 99, 132, 0.18)', // червоний
  abbr: 'rgba(255, 206, 86, 0.18)',     // жовтий
  slang: 'rgba(54, 162, 235, 0.18)',    // синій
  terms: 'rgba(0, 200, 120, 0.18)',     // зелений (замість бірюзового)
  obsolete: 'rgba(255, 105, 180, 0.18)', // рожевий (pink)
  complex: 'rgba(138, 43, 226, 0.18)',  // темно-фіолетовий (blue-violet)
  rare: 'rgba(255, 159, 64, 0.18)',     // помаранчевий
  flesch: 'rgba(150, 75, 0, 0.18)',     // коричневий (brown)
  smog: 'rgba(128, 0, 128, 0.18)',      // фіолетовий (purple)
};
```

react-frontend-app/src/constants/file.ts

```
export const MAX_FILE_SIZE = 50 * 1024 * 1024; // 50MB
export const ALLOWED_EXTENSIONS = ['PDF', 'DOC', 'DOCX', 'TXT'];

export const ALLOWED_TYPES = [
  'application/pdf',
  'application/msword',
  'text/plain',
  'application/vnd.openxmlformats-officedocument.wordprocessingml.document',
];
```

react-frontend-app/src/enums/global.config.ts

```
export const defaultDelay = 5000;
export const basePath = (import.meta as any).env?.VITE_EMOTION_API_URL ||
"http://127.0.0.1:5005";
export const timeDetectionStableEmotionMS = 1000;
export const baseTextAPIPath = (import.meta as any).env?.VITE_TEXT_API_URL ||
"http://127.0.0.1:5002";
export const baseTTSAPIPath = (import.meta as any).env?.VITE_TTS_API_URL ||
"http://127.0.0.1:5006";
export const negativeConsensusWindowMS = 2500;
export const consensusHoldDurationMS = 5000;
```

emotion-api/emotion.py

```
import cv2
from deepface import DeepFace

from flask import Flask, jsonify, Response, request
from flask_cors import CORS
import threading
import time
import os
```

```
import json
from datetime import datetime
from collections import defaultdict, deque
from typing import Dict, Optional

app = Flask(__name__)
CORS(app)

emotionData: Dict[str, Optional[str]] = {'fast': None, 'slow': None}
emotionStateData: Dict[str, Optional[str]] = {'fast': None, 'slow': None}
consensusData: Dict[str, Optional[str]] = {'emotion': None, 'state': None}
consensusNegativeFlag = False
emotion_lock = threading.Lock()

MIN_FACE_AREA = int(os.getenv('MIN_FACE_AREA', 60 * 60))
MODEL_NAME = os.getenv('DEEPFACE_MODEL', 'VGG-Face')

EMOTION_DECAY_FAST = float(os.getenv('EMOTION_DECAY', '0.78'))
EMOTION_THRESHOLD_FAST = float(os.getenv('EMOTION_THRESHOLD', '0.55'))
EMOTION_MIN_FRAMES_FAST = int(os.getenv('EMOTION_MIN_FRAMES', '6'))

EMOTION_DECAY_SLOW = float(os.getenv('EMOTION_DECAY_SLOW', '0.78'))
EMOTION_THRESHOLD_SLOW = float(os.getenv('EMOTION_THRESHOLD_SLOW', '0.5'))
EMOTION_MIN_FRAMES_SLOW = int(os.getenv('EMOTION_MIN_FRAMES_SLOW', '4'))

STATE_DECAY_FAST = float(os.getenv('STATE_DECAY', '0.8'))
STATE_THRESHOLD_FAST = float(os.getenv('STATE_THRESHOLD', '0.5'))
STATE_MIN_FRAMES_FAST = int(os.getenv('STATE_MIN_FRAMES', '4'))

STATE_DECAY_SLOW = float(os.getenv('STATE_DECAY_SLOW', '0.78'))
STATE_THRESHOLD_SLOW = float(os.getenv('STATE_THRESHOLD_SLOW', '0.45'))
STATE_MIN_FRAMES_SLOW = int(os.getenv('STATE_MIN_FRAMES_SLOW', '6'))

HISTORY_LIMIT = int(os.getenv('EMOTION_HISTORY_SIZE', '120'))
EMOTION_MODE = os.getenv('EMOTION_MODE', 'live').lower()
EMOTION_MOCK_FILE = os.getenv('EMOTION_MOCK_FILE', 'emotion_mock.json')
EMOTION_MOCK_INTERVAL = float(os.getenv('EMOTION_MOCK_INTERVAL', '0.5'))

class WeightedLabelAggregator:
    def __init__(
        self,
        decay: float = 0.78,
        min_frames: int = 6,
        activation_threshold: float = 0.55,
        min_score: float = 0.02,
    ):
        self.decay = decay
        self.min_frames = min_frames
        self.activation_threshold = activation_threshold
        self.min_score = min_score
        self._scores: Dict[str, float] = {}
        self._observations = 0
        self._stable_label = None

    def _apply_decay(self) -> None:
        for label in list(self._scores.keys()):
            self._scores[label] *= self.decay
            if self._scores[label] < self.min_score:
                del self._scores[label]
```

```
    def _recompute(self):
        if self._observations < self.min_frames or not self._scores:
            self._stable_label = None
            return self._stable_label

        best_label = max(self._scores, key=self._scores.get)
        if self._scores[best_label] >= self.activation_threshold:
            self._stable_label = best_label
        else:
            self._stable_label = None
        return self._stable_label

    def update(self, contributions: Dict[str, float]):
        if not contributions:
            return self.decay_only()

        self._observations += 1
        self._apply_decay()

        for label, weight in contributions.items():
            clamped = max(0.0, min(weight, 1.0))
            if clamped == 0.0:
                continue
            contribution = clamped * (1 - self.decay)
            self._scores[label] = self._scores.get(label, 0.0) + contribution

        return self._recompute()

    def decay_only(self):
        if not self._scores:
            self._stable_label = None
            return self._stable_label

        self._apply_decay()
        return self._recompute()

    def snapshot_scores(self) -> Dict[str, float]:
        return dict(self._scores)

    @property
    def stable_label(self):
        return self._stable_label

emotion_aggregators = {
    'fast': WeightedLabelAggregator(
        decay=EMOTION_DECAY_FAST,
        min_frames=EMOTION_MIN_FRAMES_FAST,
        activation_threshold=EMOTION_THRESHOLD_FAST,
    ),
    'slow': WeightedLabelAggregator(
        decay=EMOTION_DECAY_SLOW,
        min_frames=EMOTION_MIN_FRAMES_SLOW,
        activation_threshold=EMOTION_THRESHOLD_SLOW,
    ),
}

state_aggregators = {
    'fast': WeightedLabelAggregator(
        decay=STATE_DECAY_FAST,
        min_frames=STATE_MIN_FRAMES_FAST,
```

```
        activation_threshold=STATE_THRESHOLD_FAST,
    ),
    'slow': WeightedLabelAggregator(
        decay=STATE_DECAY_SLOW,
        min_frames=STATE_MIN_FRAMES_SLOW,
        activation_threshold=STATE_THRESHOLD_SLOW,
    ),
}

print(
    "[EmotionAPI] "
    f"model={MODEL_NAME}, "
    f"fast_emotion_decay={EMOTION_DECAY_FAST},
fast_emotion_threshold={EMOTION_THRESHOLD_FAST}, "
    f"slow_emotion_decay={EMOTION_DECAY_SLOW},
slow_emotion_threshold={EMOTION_THRESHOLD_SLOW}, "
    f"history_limit={HISTORY_LIMIT}"
)

EMOTION_STATE_MAP: Dict[str, Dict[str, float]] = {
    'neutral': {'calm': 1.0},
    'happy': {'calm': 1.0},
    'sad': {'confused': 0.7, 'overloaded': 0.3},
    'fear': {'overloaded': 0.7, 'confused': 0.3},
    'angry': {'overloaded': 1.0},
    'disgust': {'overloaded': 0.9, 'confused': 0.1},
    'surprise': {'confused': 0.8, 'calm': 0.2},
}

DEFAULT_STATE_CONTRIBUTION = {'calm': 0.4}

emotion_history = deque(maxlen=HISTORY_LIMIT)
MODEL_NAME_SUPPORTED = True
NEGATIVE_STATES = {'confused', 'overloaded'}
ALLOW_SLOW_NEUTRAL_SET = {'calm', None}

def is_valid_roi(width, height):
    return width * height >= MIN_FACE_AREA

def snapshot_scores_map(aggregators):
    return {
        mode: {label: float(score) for label, score in
aggregator.snapshot_scores().items()}
        for mode, aggregator in aggregators.items()
    }

def analyze_emotion(face_roi):
    global MODEL_NAME_SUPPORTED
    if MODEL_NAME_SUPPORTED:
        try:
            return DeepFace.analyze(
                face_roi,
                actions=['emotion'],
                enforce_detection=False,
                model_name=MODEL_NAME,
            )
        except TypeError:
            MODEL_NAME_SUPPORTED = False
            print("[EmotionAPI] DeepFace.analyze() does not support
'model_name'; falling back to default backend.")
```

```
    return DeepFace.analyze(
        face_roi,
        actions=['emotion'],
        enforce_detection=False,
    )

def record_observation(raw_emotion, confidence, stable_emotions, stable_states,
consensus_state, negative_consensus):
    confidence_value = float(confidence) if confidence is not None else None
    emotion_scores = snapshot_scores_map(emotion_aggregators)
    state_scores = snapshot_scores_map(state_aggregators)

    entry = {
        'timestamp': datetime.utcnow().isoformat() + 'Z',
        'rawEmotion': raw_emotion,
        'confidence': confidence_value,
        'stableEmotionFast': stable_emotions['fast'],
        'stableEmotionSlow': stable_emotions['slow'],
        'stateFast': stable_states['fast'],
        'stateSlow': stable_states['slow'],
        'consensusState': consensus_state,
        'negativeConsensus': negative_consensus,
        'emotionScores': emotion_scores,
        'stateScores': state_scores,
    }
    emotion_history.append(entry)

def compute_consensus_state(stable_states, stable_emotions):
    fast_state = stable_states.get('fast')
    slow_state = stable_states.get('slow')
    fast_negative = fast_state in NEGATIVE_STATES
    slow_negative = slow_state in NEGATIVE_STATES

    negative_consensus = (fast_negative and slow_negative) or (fast_negative and
(slow_state in ALLOW_SLOW_NEUTRAL_SET))
    state_for_api = slow_state or fast_state

    return state_for_api, negative_consensus

def _normalize_mock_entry(entry: Dict) -> Dict:
    emotion = entry.get('emotion')
    state = entry.get('state')
    negative = entry.get('negativeConsensus')
    if negative is None and state:
        negative = state in NEGATIVE_STATES
    return {
        'emotionFast': entry.get('emotionFast', emotion),
        'emotionSlow': entry.get('emotionSlow', emotion),
        'stateFast': entry.get('stateFast', state),
        'stateSlow': entry.get('stateSlow', state),
        'emotion': emotion,
        'state': state,
        'negativeConsensus': bool(negative),
        'confidence': entry.get('confidence'),
    }

def _apply_mock_entry(entry: Dict) -> None:
    normalized = _normalize_mock_entry(entry)
    stable_emotions = {'fast': normalized['emotionFast'], 'slow':
normalized['emotionSlow']}
```

```
    stable_states = {'fast': normalized['stateFast'], 'slow':
normalized['stateSlow']}
    consensus_state = normalized['state']
    negative_consensus = normalized['negativeConsensus']

    with emotion_lock:
        record_observation(
            normalized['emotion'],
            normalized.get('confidence'),
            stable_emotions,
            stable_states,
            consensus_state,
            negative_consensus,
        )
        emotionData.update(stable_emotions)
        emotionStateData.update(stable_states)
        consensusData['emotion'] = normalized['emotion']
        consensusData['state'] = consensus_state
        global consensusNegativeFlag
        consensusNegativeFlag = negative_consensus

def mock_emotion_loop():
    try:
        with open(EMOTION_MOCK_FILE, 'r', encoding='utf-8') as f:
            entries = json.load(f)
    except FileNotFoundError:
        print(f"[EmotionAPI] Mock file not found: {EMOTION_MOCK_FILE}")
        return
    except json.JSONDecodeError as exc:
        print(f"[EmotionAPI] Mock file invalid JSON: {exc}")
        return

    if not isinstance(entries, list) or not entries:
        print("[EmotionAPI] Mock file is empty or invalid format.")
        return

    idx = 0
    while True:
        _apply_mock_entry(entries[idx])
        idx = (idx + 1) % len(entries)
        time.sleep(EMOTION_MOCK_INTERVAL)

def video_emotion_detection():
    global emotionData, emotionStateData, consensusData, consensusNegativeFlag

    # Load face cascade classifier
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

    # Start capturing video
    cap = cv2.VideoCapture(0)

    if not cap.isOpened():
        print("ERROR: Cannot open camera")
        return

    print("Camera opened successfully. Starting emotion detection...")

    frame_count = 0
    last_face_count = -1
```

```
    while True:
        # Capture frame-by-frame
        ret, frame = cap.read()

        if not ret:
            print("ERROR: Failed to capture frame")
            break

        # Convert frame to grayscale
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Convert grayscale frame to RGB format
        rgb_frame = cv2.cvtColor(gray_frame, cv2.COLOR_GRAY2RGB)

        # Detect faces in the frame
        faces = face_cascade.detectMultiScale(gray_frame, scaleFactor=1.1,
minNeighbors=5, minSize=(30, 30))

        frame_count += 1

        # Log face count only when it changes or every 50 frames
        if len(faces) != last_face_count or frame_count % 50 == 0:
            print(f"Detected {len(faces)} face(s) [frame {frame_count}]")
            last_face_count = len(faces)

        frame_emotion_contrib = defaultdict(float)
        frame_state_contrib = defaultdict(float)
        primary_emotion = None
        primary_confidence = None

        if len(faces) == 0:
            with emotion_lock:
                stable_emotions = {mode: aggregator.decay_only() for mode,
aggregator in emotion_aggregators.items()}
                stable_states = {mode: aggregator.decay_only() for mode,
aggregator in state_aggregators.items()}
                state_for_api, negative_consensus =
compute_consensus_state(stable_states, stable_emotions)
                final_emotion = stable_emotions['slow'] or
stable_emotions['fast']
                record_observation(None, None, stable_emotions, stable_states,
state_for_api, negative_consensus)
                emotionData.update(stable_emotions)
                emotionStateData.update(stable_states)
                consensusData['emotion'] = final_emotion
                consensusData['state'] = state_for_api
                global consensusNegativeFlag
                consensusNegativeFlag = negative_consensus
        else:
            for (x, y, w, h) in faces:
                if not is_valid_roi(w, h):
                    continue

                # Extract the face ROI (Region of Interest)
                face_roi = rgb_frame[y:y + h, x:x + w]

                # Perform emotion analysis on the face ROI
                result = analyze_emotion(face_roi)
```

```
                # Determine the dominant emotion and associated confidence
                emotion = result[0]['dominant_emotion']
                emotion_scores = result[0].get('emotion', {})
                confidence = float(emotion_scores.get(emotion, 0))

                normalized_confidence = float(max(0.0, min(confidence / 100.0,
1.0)))

                frame_emotion_contrib[emotion] = min(
                    1.0,
                    frame_emotion_contrib.get(emotion, 0.0) +
normalized_confidence,
                )

                state_weights = EMOTION_STATE_MAP.get(emotion,
DEFAULT_STATE_CONTRIBUTION)
                for state_label, weight in state_weights.items():
                    frame_state_contrib[state_label] = min(
                        1.0,
                        frame_state_contrib.get(state_label, 0.0) +
normalized_confidence * weight,
                    )

                if primary_emotion is None or normalized_confidence >
(primary_confidence or 0):
                    primary_emotion = emotion
                    primary_confidence = normalized_confidence

                print(
                    f"Detected emotion: {emotion} (confidence:
{confidence:.2f})"
                )

                # Draw rectangle around face and label with predicted emotion
                cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)
                cv2.putText(frame, emotion, (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)

            stable_emotions = {mode: aggregator.update(frame_emotion_contrib)
for mode, aggregator in emotion_aggregators.items()}
            stable_states = {mode: aggregator.update(frame_state_contrib) for
mode, aggregator in state_aggregators.items()}
            state_for_api, negative_consensus =
compute_consensus_state(stable_states, stable_emotions)
            final_emotion = stable_emotions['slow'] or stable_emotions['fast']

            print(
                f"Stable emotions: fast={stable_emotions['fast']}
slow={stable_emotions['slow']} "
                f"| states fast={stable_states['fast']}
slow={stable_states['slow']} | consensus_negative={negative_consensus}"
            )

            with emotion_lock:
                record_observation(primary_emotion, primary_confidence,
stable_emotions, stable_states, state_for_api, negative_consensus)
                emotionData.update(stable_emotions)
                emotionStateData.update(stable_states)
                consensusData['emotion'] = final_emotion
                consensusData['state'] = state_for_api
```

```
                consensusNegativeFlag = negative_consensus

        # Note: cv2.imshow() doesn't work in separate thread on macOS
        # Use logging instead for debugging

        # Small delay to prevent high CPU usage
        time.sleep(0.1)

    # Release the capture and close all windows
    cap.release()
    cv2.destroyAllWindows()

if EMOTION_MODE == 'mock':
    thread = threading.Thread(target=mock_emotion_loop, daemon=True)
    thread.start()
elif EMOTION_MODE != 'off':
    thread = threading.Thread(target=video_emotion_detection, daemon=True)
    thread.start()

@app.route('/emotion', methods=['GET'])
def get_emotion():
    with emotion_lock:
        response = {
            'emotion': consensusData['emotion'],
            'state': consensusData['state'],
            'emotionFast': emotionData['fast'],
            'emotionSlow': emotionData['slow'],
            'stateFast': emotionStateData['fast'],
            'stateSlow': emotionStateData['slow'],
            'negativeConsensus': consensusNegativeFlag,
            'scores': snapshot_scores_map(emotion_aggregators),
            'stateScores': snapshot_scores_map(state_aggregators),
        }
    return jsonify(response)

@app.route('/emotion/timeline', methods=['GET'])
def get_emotion_timeline():
    limit = int(request.args.get('limit', 60))
    with emotion_lock:
        data = list(emotion_history)[-limit:]
    return jsonify(data)

if __name__ == '__main__':
    app.run(port=5005)
def get_consensus_criteria():
    base_window = int(os.getenv('NEGATIVE_CONSENSUS_WINDOW_MS', '2500'))
    return base_window
```

tts-api/app.py

```
import base64
import io
import json
import os
import re
import wave
from typing import Dict, Iterable, List, Tuple

import numpy as np
from flask import Flask, Response, jsonify, request
```

```
from flask_cors import CORS

app = Flask(__name__)
CORS(
    app,
    origins=["http://localhost:5173", "http://127.0.0.1:5173"],
    methods=["POST", "OPTIONS"],
    allow_headers=["Content-Type", "Authorization"],
)

TTS_BACKEND = os.getenv("TTS_BACKEND", "silero")
TTS_MODEL_NAME = os.getenv("TTS_MODEL_NAME", "tts_models/uk/mai/vits")
TTS_MODEL_PATH = os.getenv("TTS_MODEL_PATH")
TTS_CONFIG_PATH = os.getenv("TTS_CONFIG_PATH")
TTS_DEVICE = os.getenv("TTS_DEVICE", "cpu")
TTS_SPEAKER = os.getenv("TTS_SPEAKER", "mykyta")
TTS_LANGUAGE = os.getenv("TTS_LANGUAGE", "uk")

_engine = None


class CoquiEngine:
    def __init__(self) -> None:
        try:
            from TTS.api import TTS
        except Exception as exc:
            raise RuntimeError("Coqui TTS is not installed") from exc

        if TTS_MODEL_PATH:
            self.tts = TTS(model_path=TTS_MODEL_PATH,
config_path=TTS_CONFIG_PATH).to(TTS_DEVICE)
        else:
            self.tts = TTS(model_name=TTS_MODEL_NAME).to(TTS_DEVICE)

        self.sample_rate = getattr(self.tts.synthesizer, "output_sample_rate",
22050)

    def synthesize(self, text: str) -> Tuple[np.ndarray, int]:
        kwargs = {"speaker": TTS_SPEAKER}
        if getattr(self.tts, "is_multi_lingual", False):
            kwargs["language"] = TTS_LANGUAGE
        wav = self.tts.tts(text, **kwargs)
        audio = np.asarray(wav, dtype=np.float32)
        return audio, self.sample_rate


class SileroEngine:
    def __init__(self) -> None:
        try:
            import torch
        except Exception as exc:
            raise RuntimeError("PyTorch is required for Silero TTS") from exc

        self.torch = torch
        self.device = torch.device(TTS_DEVICE)
        self.sample_rate = 48000

        if TTS_MODEL_PATH:
            model = torch.jit.load(TTS_MODEL_PATH, map_location=self.device)
            self.model = model
```

```
        else:
            try:
                model, _ = torch.hub.load(
                    repo_or_dir="snakers4/silero-models",
                    model="silero_tts",
                    language="ua",
                    speaker="v4_ua",
                    trust_repo=True,
                )
            except Exception as exc:
                raise RuntimeError(
                    "Silero model not found. Set TTS_MODEL_PATH or download via
torch.hub."
                ) from exc
            self.model = model
            if hasattr(self.model, "to"):
                self.model.to(self.device)
            if hasattr(self.model, "sample_rate"):
                self.sample_rate = int(self.model.sample_rate)

    def synthesize(self, text: str) -> Tuple[np.ndarray, int]:
        kwargs = {
            "text": text,
            "speaker": TTS_SPEAKER,
            "sample_rate": self.sample_rate,
        }
        audio = self.model.apply_tts(**kwargs)
        if isinstance(audio, self.torch.Tensor):
            audio = audio.detach().cpu().numpy()
        audio = np.asarray(audio, dtype=np.float32)
        return audio, self.sample_rate


def get_engine():
    global _engine
    if _engine is not None:
        return _engine

    if TTS_BACKEND == "coqui":
        _engine = CoquiEngine()
        return _engine
    if TTS_BACKEND == "silero":
        _engine = SileroEngine()
        return _engine

    raise RuntimeError(f"Unsupported backend: {TTS_BACKEND}")


def audio_to_wav_bytes(audio: np.ndarray, sample_rate: int) -> bytes:
    audio = np.clip(audio, -1.0, 1.0)
    pcm = (audio * 32767).astype(np.int16)
    buffer = io.BytesIO()
    with wave.open(buffer, "wb") as wf:
        wf.setnchannels(1)
        wf.setsampwidth(2)
        wf.setframerate(sample_rate)
        wf.writeframes(pcm.tobytes())
    return buffer.getvalue()
```

```
WORD_REGEX = re.compile(r"[^\W_]+(?:['’\-][^\W_]+)*", re.UNICODE)


def build_word_timings(text: str, duration_ms: float) -> List[Dict]:
    words = []
    matches = list(WORD_REGEX.finditer(text))
    if not matches:
        return words
    weights = [max(1, len(m.group(0))) for m in matches]
    total_weight = sum(weights)
    cursor_ms = 0.0
    for match, weight in zip(matches, weights):
        word_duration = duration_ms * (weight / total_weight)
        start_ms = cursor_ms
        end_ms = cursor_ms + word_duration
        cursor_ms = end_ms
        words.append({
            "text": match.group(0),
            "start": match.start(),
            "end": match.end(),
            "start_ms": round(start_ms, 2),
            "end_ms": round(end_ms, 2),
        })
    return words


def stream_tts(chunks: List[Dict]) -> Iterable[str]:
    yielded_any = False
    try:
        engine = get_engine()
        app.logger.info("TTS engine initialized backend=%s", TTS_BACKEND)
        for idx, chunk in enumerate(chunks):
            text = (chunk.get("text") or "").strip()
            if not text:
                app.logger.warning("Skipping empty TTS chunk index=%s", idx)
                continue
            app.logger.info("TTS chunk %s text_len=%s", idx, len(text))
            audio, sample_rate = engine.synthesize(text)
            app.logger.info("TTS chunk %s samples=%s sample_rate=%s", idx,
len(audio), sample_rate)
            duration_ms = (len(audio) / sample_rate) * 1000.0 if sample_rate
else 0.0
            wav_bytes = audio_to_wav_bytes(audio, sample_rate)
            payload = {
                "index": idx,
                "text": text,
                "audio": base64.b64encode(wav_bytes).decode("ascii"),
                "duration_ms": round(duration_ms, 2),
                "sample_rate": sample_rate,
                "chunk_start": int(chunk.get("start", 0)),
                "chunk_end": int(chunk.get("end", 0)),
                "words": build_word_timings(text, duration_ms),
            }
            yielded_any = True
            yield json.dumps(payload, ensure_ascii=False) + "\n"
    except Exception as exc:
        app.logger.exception("TTS stream failed: %s", exc)
        payload = {
            "error": "TTS synthesis failed",
            "code": "tts_synthesis_failed",
```

```
            "details": str(exc),
        }
        yield json.dumps(payload, ensure_ascii=False) + "\n"
        return

    if not yielded_any:
        app.logger.warning("TTS stream finished without audio chunks")
        payload = {
            "error": "No audio chunks produced",
            "code": "tts_empty_stream",
        }
        yield json.dumps(payload, ensure_ascii=False) + "\n"


@app.route("/tts/stream", methods=["POST"])
def tts_stream() -> Response:
    data = request.get_json(silent=True) or {}
    chunks = data.get("chunks")
    if not isinstance(chunks, list) or not chunks:
        return jsonify({"error": "No chunks provided"}), 400
    return Response(stream_tts(chunks), mimetype="application/x-ndjson")


@app.route("/tts/synthesize", methods=["GET", "POST"])
def tts_synthesize() -> Response:
    data = request.get_json(silent=True) or {}
    text = request.args.get("text") or data.get("text") or ""
    text = text.strip()
    if not text:
        return jsonify({"error": "Text is required"}), 400
    engine = get_engine()
    audio, sample_rate = engine.synthesize(text)
    wav_bytes = audio_to_wav_bytes(audio, sample_rate)
    return Response(wav_bytes, mimetype="audio/wav", headers={"X-Sample-Rate":
str(sample_rate)})


@app.route("/tts/health", methods=["GET"])
def tts_health() -> Response:
    return jsonify({"status": "ok", "backend": TTS_BACKEND})


if __name__ == "__main__":
    app.run(debug=True, port=5006)
```

emotion-api/calibrate.py

```
#!/usr/bin/env python3
"""
Програмна калібрування параметрів емоційного агрегатора.

Детермінована процедура:
  1. θ, n_min — дизайн-параметри (рівень впевненості, холодний старт)
  2. α — макс. допустимий за обмеженнями (аналітично):
       T_react = ⌈ log(1−θ/c⁻) / log α ⌉  ≤  T_max   (час реакції)
       τ_eff   = −1/ln α                  ≤  τ_max   (вікно пам'яті)
  3. Верифікація метрик C, U, R, L на телеметричному корпусі
  4. Аналіз чутливості (варіація навколо обраних значень)
"""
```

```
import json
import math
import os
import sys
from datetime import datetime
from itertools import product
from typing import Dict, List, Optional, Tuple


# ── Aggregator (standalone from emotion.py) ──────────────────────────

class WeightedLabelAggregator:
    def __init__(self, decay=0.78, min_frames=6,
                 activation_threshold=0.55, min_score=0.02):
        self.decay = decay
        self.min_frames = min_frames
        self.activation_threshold = activation_threshold
        self.min_score = min_score
        self._scores: Dict[str, float] = {}
        self._observations = 0
        self._stable_label: Optional[str] = None

    def _apply_decay(self):
        for label in list(self._scores):
            self._scores[label] *= self.decay
            if self._scores[label] < self.min_score:
                del self._scores[label]

    def _recompute(self):
        if self._observations < self.min_frames or not self._scores:
            self._stable_label = None
            return self._stable_label
        best = max(self._scores, key=self._scores.get)
        self._stable_label = best if self._scores[best] >=
self.activation_threshold else None
        return self._stable_label

    def update(self, contributions: Dict[str, float]):
        if not contributions:
            return self.decay_only()
        self._observations += 1
        self._apply_decay()
        for label, weight in contributions.items():
            clamped = max(0.0, min(weight, 1.0))
            if clamped == 0.0:
                continue
            self._scores[label] = self._scores.get(label, 0.0) + clamped * (1 -
self.decay)
        return self._recompute()

    def decay_only(self):
        if not self._scores:
            self._stable_label = None
            return self._stable_label
        self._apply_decay()
        return self._recompute()


# ── Mapping ──────────────────────────────────────────────────────────
```

```
EMOTION_STATE_MAP = {
    'neutral':  {'calm': 1.0},
    'happy':    {'calm': 1.0},
    'sad':      {'confused': 0.7, 'overloaded': 0.3},
    'fear':     {'overloaded': 0.7, 'confused': 0.3},
    'angry':    {'overloaded': 1.0},
    'disgust':  {'overloaded': 0.9, 'confused': 0.1},
    'surprise': {'confused': 0.8, 'calm': 0.2},
}
DEFAULT_STATE = {'calm': 0.4}


# ── Data loading ──────────────────────────────────────────────

def load_timeline(path):
    with open(path, 'r', encoding='utf-8') as f:
        entries = json.load(f)
    frames, tabs = [], []
    for e in entries:
        frames.append((e.get('rawEmotion'), e.get('confidence')))
        ts = e.get('timestamp', '')
        try:
            tabs.append(datetime.fromisoformat(ts.replace('Z', '+00:00')))
        except Exception:
            tabs.append(None)
    valid = [t for t in tabs if t is not None]
    t0 = valid[0] if valid else None
    rel = [(t - t0).total_seconds() if t and t0 else i * 0.2
           for i, t in enumerate(tabs)]
    return frames, rel


def compute_c_avg(all_data):
    v = [c for fr, _ in all_data for _, c in fr if c and c > 0]
    return sum(v) / len(v) if v else 0.5


def compute_fps(all_data):
    r = [(len(t) - 1) / (t[-1] - t[0])
         for _, t in all_data if len(t) >= 2 and t[-1] > t[0]]
    return sum(r) / len(r) if r else 5.0


# ── Analytical derivation ─────────────────────────────────────

def t_react_frames(alpha, theta, c_avg):
    """T_react = ⌈log(1−θ/c̄)/log(α)⌉ frames."""
    if theta >= c_avg or alpha <= 0 or alpha >= 1:
        return float('inf')
    r = 1.0 - theta / c_avg
    return math.ceil(math.log(r) / math.log(alpha)) if r > 0 else float('inf')


def tau_eff_frames(alpha):
    """τ_eff = −1/ln(α) frames."""
    return -1.0 / math.log(alpha) if 0 < alpha < 1 else float('inf')


def alpha_from_t_react(theta, c_avg, t_max_frames):
    """Max α such that T_react ≤ t_max."""
```

```
    r = 1.0 - theta / c_avg
    return r ** (1.0 / t_max_frames) if r > 0 else 0.0


def alpha_from_tau(tau_max_frames):
    """Max α such that τ_eff ≤ τ_max."""
    return math.exp(-1.0 / tau_max_frames)


def snap_to_grid(val, step=0.02):
    """Round down to nearest grid step."""
    return math.floor(val / step) * step


# ── Simulation & Metrics ──────────────────────────────────────

def simulate(frames, alpha, theta, n_min, mode='emotion'):
    agg = WeightedLabelAggregator(decay=alpha, min_frames=n_min,
                                  activation_threshold=theta)
    labels = []
    for raw, conf in frames:
        if raw and conf and conf > 0:
            if mode == 'emotion':
                c = {raw: conf}
            else:
                ws = EMOTION_STATE_MAP.get(raw, DEFAULT_STATE)
                c = {s: conf * w for s, w in ws.items()}
            agg.update(c)
        else:
            agg.decay_only()
        labels.append(agg._stable_label)
    return labels


def compute_metrics(labels, times):
    n = len(labels)
    if n == 0:
        return {'C': 0, 'U': 1, 'R': 0, 'L': 0}
    dur = max(times[-1] - times[0], n * 0.2)
    C = sum(1 for l in labels if l is not None) / n
    trans = sum(1 for i in range(1, n) if labels[i] != labels[i - 1])
    R = trans / (dur / 60) if dur > 0 else 0
    runs, rs = [], 0
    for i in range(1, n):
        if labels[i] != labels[rs]:
            if labels[rs] is not None:
                runs.append(times[i] - times[rs])
            rs = i
    if labels[rs] is not None:
        runs.append(times[-1] - times[rs])
    L = sum(runs) / len(runs) if runs else 0
    return {'C': C, 'U': 1 - C, 'R': R, 'L': L}


def I_index(m):
    d = m['C'] * m['L']
    return m['R'] / d if d > 0 else float('inf')


def eval_config(all_data, alpha, theta, n_min, mode):
```

```
    fms = [compute_metrics(simulate(fr, alpha, theta, n_min, mode), tm)
           for fr, tm in all_data]
    avg = {k: sum(fm[k] for fm in fms) / len(fms) for k in ('C', 'U', 'R', 'L')}
    avg['I'] = I_index(avg)
    return avg, fms


# ── Sensitivity analysis ──────────────────────────────────────

def sensitivity(all_data, mode, a0, t0, n0, c_avg, fps):
    alphas = sorted(set(round(a0 + d * 0.02, 2)
                        for d in range(-5, 6) if 0.50 <= a0 + d * 0.02 <= 0.96))
    thetas = sorted(set(round(t0 + d * 0.05, 2)
                        for d in range(-2, 3) if 0.30 <= t0 + d * 0.05 <= 0.80))
    ns = sorted(set(n for n in range(max(2, n0 - 3), min(11, n0 + 4))))

    results = []
    for a, th, nm in product(alphas, thetas, ns):
        avg, _ = eval_config(all_data, a, th, nm, mode)
        tr = t_react_frames(a, th, c_avg)
        te = tau_eff_frames(a)
        avg['T_react'] = tr
        avg['tau_eff'] = te
        results.append(({'decay': a, 'threshold': th, 'min_frames': nm}, avg))

    results.sort(key=lambda x: x[1]['I'])
    return results


# ── Main ──────────────────────────────────────────────────────

def main():
    base = os.path.dirname(os.path.abspath(__file__))
    file_defs = [
        ('VGG-Face',          os.path.join(base, 'timeline_vgg.json')),
        ('FaceNet',           os.path.join(base, 'timeline_facenet.json')),
        ('VGG (high thresh)', os.path.join(base,
'timeline_vgg_high_thresh.json')),
    ]

    all_data = [load_timeline(p) for _, p in file_defs]
    c_avg = compute_c_avg(all_data)
    fps = compute_fps(all_data)

    print("=" * 74)
    print("  ПРОГРАМНА КАЛІБРУВАННЯ ПАРАМЕТРІВ ЕМОЦІЙНОГО АГРЕГАТОРА")
    print("=" * 74)
    print()
    print(f"  Телеметричний корпус (3 серії, {sum(len(fr) for fr, _ in
all_data)} кадрів):")
    for (name, _), (fr, tm) in zip(file_defs, all_data):
        print(f"    {name}: {len(fr)} кадрів, {tm[-1]:.1f}c")
    print(f"  Середня впевненість детектора c¯ = {c_avg:.4f}")
    print(f"  Середня частота кадрів fps = {fps:.2f}")
    print()

    #
═══════════════════════════════════════════════════════════════════
    # ДИЗАЙН-ПАРАМЕТРИ
    #
```

```
    T_MAX = 5  # frames — budget for reaction time (~1.0с at 5 fps)

    # τ_max in SECONDS, converted to frames using actual fps
    TAU_EMOTION = 0.8  # seconds — tight memory for emotion channels
    TAU_STATE_F = 1.0  # seconds — wider for reactive state channel
    TAU_STATE_S = 0.8  # seconds — tight for slow consensus state

    channels = {
        'emotion_fast': {
            'theta': 0.55,  # 71% of c⁻ — high precision, fast reactive
            'n_min': 6,     # T_react + 1, stabilization guard
            'mode': 'emotion',
            'tau_sec': TAU_EMOTION,
        },
        'emotion_slow': {
            'theta': 0.50,  # 65% — moderate precision, smooth tracking
            'n_min': 4,     # lower n_min for faster activation
            'mode': 'emotion',
            'tau_sec': TAU_EMOTION,
        },
        'state_fast': {
            'theta': 0.50,  # 65% — states are derived, moderate precision
            'n_min': 4,     # reactive state detection
            'mode': 'state',
            'tau_sec': TAU_STATE_F,
        },
        'state_slow': {
            'theta': 0.45,  # 58% — max coverage for consensus channel
            'n_min': 6,     # conservative for reliable consensus
            'mode': 'state',
            'tau_sec': TAU_STATE_S,
        },
    }

    print(f"  Дизайн-параметри:")
    print(f"    T_max = {T_MAX} кадрів ({T_MAX/fps:.1f}с) — бюджет часу
реакції")
    print()
    print(f"    {'Канал':<16} {'θ':>5} {'θ/c⁻':>5} {'n_min':>6} {'τ_max':>7}
{'Призначення'}")
    print(f"    {'':—<16} {'':—>5} {'':—>5} {'':—>6} {'':—>7} {'':—<30}")
    for ch, cfg in channels.items():
        ratio = cfg['theta'] / c_avg
        tau_fr = cfg['tau_sec'] * fps
        print(f"    {ch:<16} {cfg['theta']:>5.2f} {ratio:>5.1%}
{cfg['n_min']:>6d} "
              f"{cfg['tau_sec']:>5.1f}с  "
              f"{'висока точність' if cfg['theta'] >= 0.55 else 'помірна
точність' if cfg['theta'] >= 0.50 else 'макс. покриття'}")
    print()

    #
═══════════════════════════════════════════════════════════════════════
    # АНАЛІТИЧНЕ ВИЗНАЧЕННЯ α
    #
═══════════════════════════════════════════════════════════════════════

    summary = {}
```

```
    for ch_name, cfg in channels.items():
        theta = cfg['theta']
        n_min = cfg['n_min']
        mode = cfg['mode']
        tau_sec = cfg['tau_sec']
        tau_frames = tau_sec * fps

        # Analytical α bounds
        alpha_T = alpha_from_t_react(theta, c_avg, T_MAX)
        alpha_tau = alpha_from_tau(tau_frames)
        alpha_bound = min(alpha_T, alpha_tau)
        alpha_opt = snap_to_grid(alpha_bound, 0.02)
        binding = "T_react" if alpha_T <= alpha_tau else "τ_eff"

        # Verify
        tr = t_react_frames(alpha_opt, theta, c_avg)
        te = tau_eff_frames(alpha_opt)
        te_sec = te / fps

        # Metrics on telemetry
        avg, per_file = eval_config(all_data, alpha_opt, theta, n_min, mode)
        avg['T_react'] = tr
        avg['tau_eff'] = te

        print(f"{'—' * 74}")
        print(f"  {ch_name}")
        print(f"  θ={theta}, n_min={n_min}, T_max={T_MAX}, τ_max={tau_sec}c
({tau_frames:.1f} кадрів)")
        print()
        print(f"    α(T_react)  = (1−θ/c⁻)^(1/{T_MAX})  = ({1-
theta/c_avg:.4f})^{1/T_MAX:.1f}  "
              f"= {alpha_T:.4f}")
        print(f"    α(τ_eff)    = exp(−1/{tau_frames:.1f})       "
              f"= {alpha_tau:.4f}")
        print(f"    α*          = min → grid(0.02)  = {alpha_opt:.2f}  "
              f"[обмежуюча: {binding}]")
        print()
        print(f"    T_react = {tr} кадрів ({tr/fps:.2f}c)")
        print(f"    τ_eff   = {te:.2f} кадрів ({te_sec:.2f}c)")
        print()
        print(f"    ═══  α = {alpha_opt:.2f},  θ = {theta},  n_min = {n_min}")
        print()

        # Metrics table
        print(f"    Метрики (телеметрія):")
        print(f"      {'Серія':22s}  {'C':>5} {'U':>5} {'R/хв':>6} {'L(c)':>5}
{'I':>7}")
        fnames = [n for n, _ in file_defs]
        for fn, fm in zip(fnames, per_file):
            print(f"      {fn:22s}  {fm['C']:>5.3f} {fm['U']:>5.3f} "
                  f"{fm['R']:>6.1f} {fm['L']:>5.2f} {I_index(fm):>7.2f}")
        print(f"      {'Середнє':22s}  {avg['C']:>5.3f} {avg['U']:>5.3f} "
              f"{avg['R']:>6.1f} {avg['L']:>5.2f} {avg['I']:>7.2f}")
        print()

        # Sensitivity
        sens = sensitivity(all_data, mode, alpha_opt, theta, n_min, c_avg, fps)
        opt_rank = next((i + 1 for i, (p, _) in enumerate(sens)
                         if p['decay'] == alpha_opt and p['threshold'] == theta
```

```
                        and p['min_frames'] == n_min), None)
        print(f"    Чутливість (ранг {opt_rank}/{len(sens)}, ±0.10/±0.10/±3):")
        print(f"      {'α':>5} {'θ':>5} {'n':>3} {'T_r':>4} {'τ':>4}  "
              f"{'C':>5} {'R/хв':>6} {'L':>5} {'I':>7}")
        for p, a in sens[:8]:
            m = " ◀" if (p['decay'] == alpha_opt and p['threshold'] == theta
                        and p['min_frames'] == n_min) else ""
            print(f"      {p['decay']:>5.2f} {p['threshold']:>5.2f} "
                  f"{p['min_frames']:>3d} {a.get('T_react',0):>4.0f} "
                  f"{a.get('tau_eff',0):>4.1f}  "
                  f"{a['C']:>5.3f} {a['R']:>6.1f} {a['L']:>5.2f}
{a['I']:>7.2f}{m}")
        print()

        summary[ch_name] = {
            'parameters': {'decay': alpha_opt, 'threshold': theta, 'min_frames':
n_min},
            'derivation': {
                'alpha_T_react': round(alpha_T, 4),
                'alpha_tau_eff': round(alpha_tau, 4),
                'alpha_bound': round(alpha_bound, 4),
                'binding': binding,
                'T_react': tr,
                'T_react_sec': round(tr / fps, 2),
                'tau_eff': round(te, 2),
                'tau_eff_sec': round(te_sec, 2),
            },
            'design': {
                'theta': theta, 'theta_ratio': round(theta / c_avg, 4),
                'n_min': n_min, 'T_max': T_MAX, 'tau_max_sec': tau_sec,
                'tau_max_frames': round(tau_frames, 2),
            },
            'telemetry': {
                'c_avg': round(c_avg, 4), 'fps': round(fps, 2),
                'files': [n for n, _ in file_defs],
            },
            'metrics_avg': {k: round(v, 4) for k, v in avg.items()},
            'metrics_per_file': [{k: round(v, 4) for k, v in fm.items()}
                                 for fm in per_file],
            'sensitivity_rank': opt_rank,
            'sensitivity_total': len(sens),
        }

    #
═══════════════════════════════════════════════════════════════════════
    # ПІДСУМОК
    #
═══════════════════════════════════════════════════════════════════════

    print("=" * 74)
    print("  ПІДСУМОК КАЛІБРУВАННЯ")
    print("=" * 74)
    print()
    print(f"  {'Канал':<16} {'α':>5} {'θ':>5} {'n':>3}  "
          f"{'T_r':>3} {'τ':>4}  {'C':>5} {'R/хв':>6} {'L':>5} {'I':>6}")
    print("  " + "—" * 62)

    for ch in ['emotion_fast', 'emotion_slow', 'state_fast', 'state_slow']:
        s = summary[ch]
        p = s['parameters']
```

```
        m = s['metrics_avg']
        d = s['derivation']
        print(f"  {ch:<16} {p['decay']:>5.2f} {p['threshold']:>5.2f} "
              f"{p['min_frames']:>3d}  {d['T_react']:>3d} {d['tau_eff']:>4.1f}
"
              f"{m['C']:>5.3f} {m['R']:>6.1f} {m['L']:>5.2f} {m['I']:>6.2f}")
    print()

    # Reference comparison
    REF = {
        'emotion_fast': (0.78, 0.55, 6),
        'emotion_slow': (0.78, 0.50, 4),
        'state_fast':   (0.80, 0.50, 4),
        'state_slow':   (0.78, 0.45, 6),
    }

    print("  Верифікація відповідності:")
    print(f"  {'Канал':<16} {'α_cal':>6} {'α_ref':>6} {'θ_cal':>6} {'θ_ref':>6}
"
          f"{'n_cal':>6} {'n_ref':>6}")
    print("  " + "—" * 52)
    ok = True
    for ch in ['emotion_fast', 'emotion_slow', 'state_fast', 'state_slow']:
        p = summary[ch]['parameters']
        ra, rt, rn = REF[ch]
        da = "✓" if p['decay'] == ra else "✗"
        dt = "✓" if p['threshold'] == rt else "✗"
        dn = "✓" if p['min_frames'] == rn else "✗"
        if da == "✗" or dt == "✗" or dn == "✗":
            ok = False
        print(f"  {ch:<16} {p['decay']:>6.2f} {ra:>6.2f} {da}  "
              f"{p['threshold']:>5.2f} {rt:>5.2f} {dt}  "
              f"{p['min_frames']:>5d} {rn:>5d} {dn}")

    print()
    if ok:
        print("  ✓ Усі калібровані параметри повністю збігаються з реалізацією.")
    else:
        print("  △ Є розбіжності — перевірте дизайн-параметри або обмеження.")
    print()

    out = os.path.join(base, 'calibration_results.json')
    with open(out, 'w', encoding='utf-8') as f:
        json.dump(summary, f, indent=2, ensure_ascii=False)
    print(f"  Збережено → {out}")
    print()

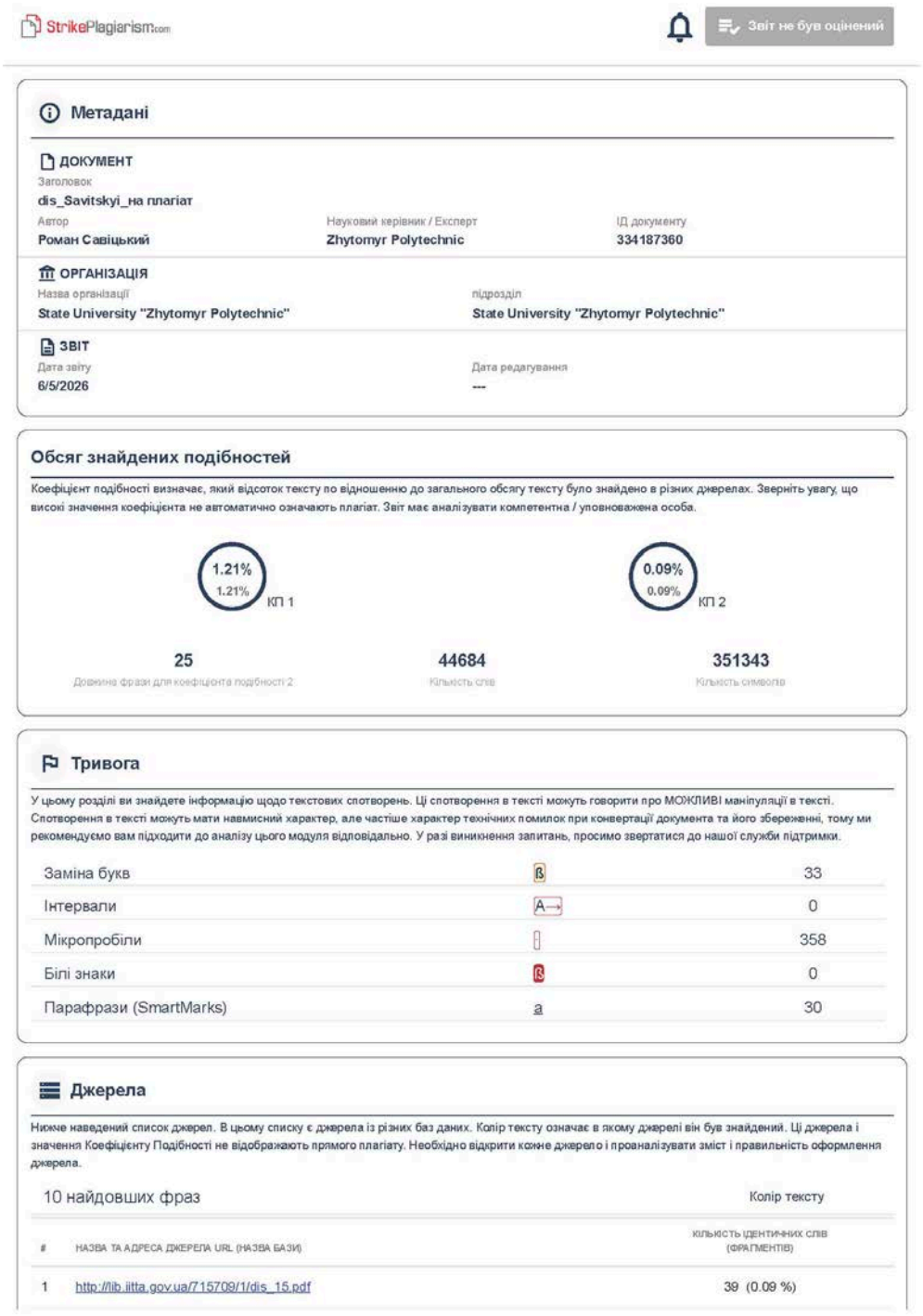

if __name__ == '__main__':
    main()
```

Додаток Е

Звіт подібності дисертації

Савіцького Романа Святославовича

"Методи і програмні засоби аналізу вебдоступності в освітньому процесі для людей з когнітивними та інтелектуальними порушеннями"



StrikePlagiarism.com

Звіт не був оцінений

Метадані

ДОКУМЕНТ

Заголовок: dis_Savitskyi_на плагіат

Автор: Роман Савіцький

Науковий керівник / Експерт: Zhytomyr Polytechnic

ІД документу: 334187360

ОРГАНІЗАЦІЯ

Назва організації: State University "Zhytomyr Polytechnic"

підрозділ: State University "Zhytomyr Polytechnic"

ЗВІТ

Дата звіту: 6/5/2026

Дата редагування: ---

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

1.21% 1.21% КП 1

0.09% 0.09% КП 2

25 Довжина фрази для коефіцієнта подібності 2

44684 Кількість слів

351343 Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	33
Інтервали	A→	0
Мікропробіли		358
Білі знаки		0
Парафрази (SmartMarks)	a	30

Джерела

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз		Колір тексту
#	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	http://lib.iitta.gov.ua/715709/1/dis_15.pdf	39 (0.09 %)

2	https://uacademic.info/ua/document/0525U000045	20 (0.04 %)
3	http://dndi.mvs.gov.ua/files/pdf/dissertation_Fyl_R_S.pdf	19 (0.04 %)
4	http://lib.iitta.gov.ua/715709/1/dis_15.pdf	19 (0.04 %)
5	http://lib.iitta.gov.ua/715709/1/dis_15.pdf	17 (0.04 %)
6	https://library.knu.ua/wp-content/uploads/2024/12/tsp_z_pytan_vyshhoyi_osvity_ta_nauky_vypusk_003.pdf	17 (0.04 %)
7	http://lib.iitta.gov.ua/715709/1/dis_15.pdf	15 (0.03 %)
8	https://uacademic.info/en/document/0425U000119	14 (0.03 %)
9	http://dndi.mvs.gov.ua/files/pdf/dissertation_Fyl_R_S.pdf	14 (0.03 %)
10	http://lib.iitta.gov.ua/715709/1/dis_15.pdf	13 (0.03 %)

Домашня база даних

#	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

Програма обміну базами даних (0.37 %)

#	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	модель.docx [redacted] 5/1/2023 Publishing House "Helvetica" (Видавничий дім "Гельветика")	38 (6) (0.09 %)
2	Дослідження та реалізація системи доступності веб-сайту [redacted] 5/25/2026 Lviv University of Trade and Economics (Lviv University of Trade and Economics)	18 (2) (0.04 %)
3	Диплом_Зінченко_Удосконалення_механізму_управління_розвитком_підприємства [redacted] 1/5/2026 National University "Zaporizhzhia Polytechnic" (Кафедра "Менеджмент та адміністрування")	13 (1) (0.03 %)
4	стаття Чернушкіна [redacted] 12/20/2024 Kryvyi Rih Professional Medical College (Kryvyi Rih Professional Medical College)	12 (1) (0.03 %)
5	Розвиток соціальної інклюзії осіб з обмеженими можливостями [redacted] 2/2/2026 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра соціоекономіки та управління персоналом)	12 (1) (0.03 %)
6	БДР Барасій [redacted] 6/5/2025 Vinnytsya National Technical University (ПЗ)	11 (2) (0.02 %)
7	Методи та програмне забезпечення децентралізованого управління груповим польотом безпілотних літальних апаратів на основі теорії неоднорідного векторного поля [redacted] 3/16/2025 National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute (National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute)	10 (1) (0.02 %)
8	masa communication report.docx [redacted] 11/7/2025 Chevron College (Chevron College)	10 (1) (0.02 %)
9	Dis_Ukrainskyi_antiplagiat [redacted] 8/26/2021 Pryazovskyi State Technical University (PSTU)	9 (1) (0.02 %)

10	ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ОЦІНЮВАННЯ ЯКОСТІ ПРОЄКТУВАННЯ ТА ДИЗАЙНУ ВЕБДОДАТКІВ [redacted] 11/3/2025 National University "Lviv Politechnika" (NULP2)	8 (1) (0.02 %)
11	БС_21_ДР_Коломієць_плаг [redacted] 5/31/2026 National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute (ФБМІ, К-ра біомедичної кібернетики)	8 (1) (0.02 %)
12	6091-1 [redacted] 3/14/2025 Institute for Digitalisation of Education of NAES (Institute for Digitalisation of Education of NAES)	6 (1) (0.01 %)
13	Психологічні особливості феномену ментального здоров'я студентів медичного коледжу [redacted] 5/11/2026 Nizhyn Mykola Gogol State University (Факультет педагогіки, психології, соціальної роботи та мистецтв)	5 (1) (0.01 %)
14	Автореферат Вплив структурних дисбалансів фінансової системи на економічну безпеку держави [redacted] 9/10/2022 Sumy State University (Кафедра економічної кібернетики)	5 (1) (0.01 %)

Інтернет (0.84 %)

#	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
15	http://lib.iitta.gov.ua/715709/1/dis_15.pdf	**187 (15) (0.42 %)**
16	http://dndi.mvs.gov.ua/files/pdf/dissertation_Fyl_R_S.pdf	33 (2) (0.07 %)
17	https://uacademic.info/ua/document/0525U000045	31 (2) (0.07 %)
18	https://library.knu.ua/wp-content/uploads/2024/12/tsp_z_pytan_vyshhoyi_osvity_ta_nauky_vypusk_003.pdf	17 (1) (0.04 %)
19	https://knutd.edu.ua/files/doctorate_postgraduate/Metod-rekom_dis_df_2022.pdf	17 (2) (0.04 %)
20	https://ecoconference.kpi.ua/article/download/301624/298722/709825	15 (2) (0.03 %)
21	https://uacademic.info/en/document/0425U000119	14 (1) (0.03 %)
22	http://ir.polissiauniver.edu.ua/bitstream/123456789/14052/3/Zavorotniuk_AI_KR_T_2023.pdf	12 (1) (0.03 %)
23	http://kk.nau.edu.ua/news/1008	11 (1) (0.02 %)
24	https://mydisser.com/en/catalog/view/386/822/497419.html	11 (1) (0.02 %)
25	http://dspace.opu.ua/jspui/bitstream/123456789/14219/1/13-18-Lugovsky.docx	8 (1) (0.02 %)
26	https://cyberleninka.ru/article/n/razrabotka-adaptivnogo-fuzzy-algoritma-identifikatsii-tehnicheskih-kanalov-utechki-informatsii-1	7 (1) (0.02 %)
27	https://chtivo.org.ua/literature/naukova/9162-socialne-siritstvo-v-konteksti-suchasnih-migracijnih-procesiv-v-ukrayini.html	7 (1) (0.02 %)
28	https://vsmu.edu.ua/downloads/other/vymogy_VNMU_oforml_dyser_PhD_2022.pdf	6 (1) (0.01 %)

Список прийнятих фрагментів

#	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)