

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:

**«Інтелектуальна система автоматичного
тегування текстів для платформи виконання
завдань та тестів»**

Виконав студент групи ПЗм-24-1
НОВІЧКОВ Єгор Максимович

Керівник роботи:
ЧИЖМОТРЯ Олексій Володимирович

Рецензент:
ФАНТ Микола Олександрович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

зав. кафедри інженерії
програмного забезпечення

Тетяна ВАКАЛЮК

«18» вересня 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **НОВІЧКОВ Єгор Максимович**

Керівник роботи: **ЧИЖМОТРЯ Олексій Володимирович**

Тема роботи: **«Інтелектуальна система автоматичного тегування текстів для платформи виконання завдань та тестів»**, затверджена Наказом закладу вищої освіти від **«18» вересня 2025 р. №491/с**

Вихідні дані для роботи: **використання методів обробки природної мови та глибокого навчання для проєктування та розробки системи автоматичного тегування тексту; її інтеграція до платформи виконання завдань та тестів.**

Керівники / консультанти з магістерської кваліфікаційної роботи із зазначенням розділів,

що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Чижмотря О.В.	19.09.2025р.	19.09.2025р.
2	Чижмотря О.В.	19.09.2025р.	19.09.2025р.
3	Чижмотря О.В.	19.09.2025р.	19.09.2025р.

РЕФЕРАТ

Випускна кваліфікаційна робота ступеня «магістр» присвячена розробці системи для автоматизації тегування тексту та її інтеграція у платформу для виконання завдань та тестів. Пояснювальна записка до випускної роботи містить текстову частину, викладену на 68 сторінках друкованого тексту, має 8 сторінок додатків. Список використаних джерел містить 40 найменувань і займає 5 сторінок. В роботі наведено 27 рисунків та 1 таблицю. Загальний обсяг роботи – 76 сторінок.

Метою роботи є розробка системи автоматичного тегування тексту методами глибокого навчання та обробки природної мови, її інтеграція у систему виконання завдань та тестів для спрощення аналізу успішності учня.

У роботі визначено основні завдання на розробку системи, проведено аналіз вимог та функціональності системи, також було проведено аналіз аналогів. Було обрано методи для реалізації системи, здійснено проектування її архітектури. Описано програмну реалізацію системи та її інтеграцію у платформу для виконання завдань та тестів.

КЛЮЧОВІ СЛОВА: РЕКУРЕНТНА НЕЙРОННА МЕРЕЖА, RNN, ГЛИБОКЕ НАВЧАННЯ, NLP, ОБРОБКА ПРИРОДНОЇ МОВИ, LSTM, ДОВГА КОРОТКОЧАСНА ПАМ'ЯТЬ, TF-IDF.

ABSTRACT

The master's degree thesis is devoted to the development of a system for automating text tagging and its integration into a platform for completing tasks and tests. The explanatory note to the thesis contains a text part, set out on 68 pages of printed text, and has 8 pages of appendices. The list of references contains 40 items and occupies 5 pages. The thesis contains 27 figures and 1 table. The total volume of the thesis is 76 pages.

The purpose of the work is to develop a system for automatic text tagging using deep learning and natural language processing methods, and to integrate it into a system for completing tasks and tests to simplify the analysis of student performance.

The work defines the main tasks for developing the system, analyzes the requirements and functionality of the system, and analyzes analogues. Methods for implementing the system were selected, and its architecture was designed. The software implementation of the system and its integration into the platform for performing tasks and tests are described.

KEYWORDS: RECURRENT NEURAL NETWORK, RNN, DEEP LEARNING, NLP, NATURAL LANGUAGE PROCESSING, LSTM, LONG SHORT-TERM MEMORY, TF-IDF.

ЗМІСТ

ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ДОСЛІДЖЕННЯ.....	9
1.1 Аналіз проблеми автоматичного тегування тексту.....	9
1.2 Аналіз наявних методів автоматичного тегування тексту.....	10
1.3 Постановка задачі автоматичного тегування тексту.....	14
1.4 Аналіз засобів тегування тексту.....	16
Висновки до розділу 1.....	22
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ АВТОМАТИЧНОГО ТЕГУВАННЯ ТЕКСТУ.....	24
2.1 Вибір програмних засобів для побудови моделі тегування тексту.....	24
2.2 Підготовка даних для навчання моделі.....	25
2.3 Побудова моделі тегування тексту.....	27
Висновки до розділу 2.....	39
РОЗДІЛ 3. ОПИС ПРОГРАМНОГО ПРОДУКТУ.....	40
3.1 Інтеграція системи автоматичного тегування тексту до платформи виконання завдань та тестів.....	40
3.2 Структура інтерфейсу веб-додатку та порядок роботи.....	48
3.3 Аналіз результатів роботи системи автоматичного тегування тексту. .	58
Висновки до розділу 3.....	61
ВИСНОВКИ.....	62
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	64
ДОДАТКИ.....	69

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

LSTM – (англ. long short-term memory) – довга короткочасна пам'ять

TF-IDF – (англ. term frequency–inverse document frequency) – частота терміна – обернена частота документа

NLP – (англ. natural language processing) – обробка природної мови

LLM – (англ. large language model) – велика мовна модель

RNN – (англ. recurrent neural network) – рекурентна нейронна мережа

ВСТУП

Темою даного дипломного проєкту є розробка інтелектуальної системи автоматичного тегування текстів для платформи виконання завдань та тестів.

Актуальність теми. З процесом цифровізації освітнього процесу виникла проблема класифікації контенту. Ручна класифікація є не лише часозатратною, а й неефективною через високу ймовірність людських помилок. Відсутність структуризації матеріалів ускладнює навігацію системою, та гальмує розвиток адаптивних освітніх технологій, що вимагає впровадження інтелектуальних автоматизованих рішень.

Використання методів глибокого навчання та обробки природної мови дає багато можливостей для вирішення цієї проблеми. На відміну від класичних алгоритмів, що базуються на ключових словах, сучасні нейромережеві моделі, здатні проводити семантичний аналіз завдань, розуміючи контекст. Це дозволяє системі присвоювати теги навіть тим питанням, де відсутня пряма вказівка на тему, що забезпечує гнучкість платформи.

Дана система може бути використана для побудови модулю аналітики успішності учнів. Завдяки класифікації завдань, ми можемо перейти від загального оцінювання за предметом, до формування рейтингу за темами, що дозволяє виявити прогалини у знаннях на рівні окремих тем. На основі цих даних, вчителі можуть побачити загальну картину засвоєння матеріалу учнями без аналізу кожної відповіді окремо та відповідно корегувати план навчання, а учні можуть цілеспрямовано працювати над слабкими місцями.

Отже, розробка інтелектуальної системи автоматичного тегування є актуальним завданням, оскільки вона поєднує в собі передові досягнення у сфері штучного інтелекту з нагальними потребами менеджменту освітнього процесу.

Метою дослідження є розробка інтелектуальної системи автоматичного тегування тексту, проєктування, розробка її алгоритмів та її інтеграція до платформи виконання завдань та тестів.

Встановлена мета обумовлює наступні завдання:

- проведення аналізу задачі;
- проведення аналізу аналогічних систем;
- проведення аналізу доступних методів для побудови системи;
- проєктування та побудова системи;
- інтеграція системи.

Об’єктом дослідження є процес автоматизованої класифікації неструктурованого тексту у системі виконання завдань та тестів.

Предметом дослідження є методи обробки природної мови та моделі глибокого навчання для класифікації завдань за кількома мітками з метою формування аналітики успішності учнів.

Наукова новизна роботи полягає в удосконаленні методу автоматичної класифікації навчального контенту шляхом гібридного поєднання рекурентної нейронної мережі Bi-LSTM для врахування контексту та алгоритму TF-IDF для виділення ключових слів, що на відміну від існуючих підходів, враховує специфічну лексику предметної області та контекст запитань, що дозволяє підвищити точність класифікації за кількома мітками в умовах обмеженого обсягу тексту завдання.

Практичне значення роботи полягає у можливості автоматичної класифікації завдань для переходу від лінійної системи (оцінка за предмет) оцінювання до матричної (декілька оцінок за предмет по темам), що дозволить аналізувати успішність учня на рівні підтем та корегувати процес навчання базуючись на цих даних.

Апробація. За темою кваліфікаційної роботи було опубліковано 3 тези конференцій [1, 2, 3].

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ДОСЛІДЖЕННЯ

1.1 Аналіз проблеми автоматичного тегування тексту

Автоматичне тегування текстів (або класифікація з декількома мітками) – це ідентифікація семантичних категорій, тем або понять, які визначають зміст конкретного тексту [4]. Як правило, навчальні дані у вигляді завдань можуть бути пов'язані з однією або декількома темами, наприклад, лінійна алгебра, синтаксис мови, історичні дати, закони термодинаміки тощо, що робить можливість автоматично вибирати, до якої категорії віднести завдання, дуже бажаною, оскільки точна ідентифікація цих тем є критичною з точки зору побудови адаптивного навчального процесу [5].

У простих випадках, наявність у тексті специфічного ключового слова може надати нам потрібну інформацію для класифікації. Наприклад, слово «інтеграл» чітко вказує на математичний аналіз. Однак, аналізуючи завдання зі складною структурою або міждисциплінарним змістом, неможливо ідентифікувати тему безпосередньо через пошук окремих слів, в такому випадку потрібно аналізувати контекст та комбінацію різних термінів, що суттєво ускладнює задачу, коли потрібно масштабувати систему від простих тестів до тисяч різнопланових завдань.

Найпоширеніший спосіб класифікації інформації – це ручний перегляд кожного завдання викладачем та вибір відповідного тегу зі списку, або використання примітивних фільтрів за ключовими словами. Якщо слово знайдено – тег присвоюється. Проте такий підхід призводить до великої кількості помилок класифікації, коли присвоюється тема, до якої завдання насправді не відноситься, або тема не присвоюється взагалі. Такі помилки є неприпустимими, оскільки вони унеможливлюють використання цих даних, для, наприклад, розрахунку рейтингу успішності учня по темах, призводячи до втрати об'єктивності оцінювання.

Ці проблеми викликані або людською помилкою, або недосконалістю простих алгоритмів. Саме тому для даної задачі потрібно використовувати автоматизовані методи обробки природної мови (NLP) та глибокого навчання [6]. Таким чином можна уникнути людської помилки та зосередити увагу викладачів на аналізі вже сформованих рейтингів успішності та роботі з проблемними темами учнів.

Уявімо універсальну платформу тестування, яка з великою вірогідністю буде мати контент з різних дисциплін, що має зовсім різну лексику, структуру речень, наявність формул чи графіків. Таким чином постає питання контролю усього масиву завдань, ведення звітності по успішності, та ускладнюється процес виявлення прогалин у знаннях учнів, якщо враховувати пропорційний ріст бази тестів. У результаті зі збільшенням кількості матеріалів все складнішим стає процес підтримки актуальності рейтингової системи. Зазвичай у такому разі залучається більше модераторів, які вручну тегують контент. Або ж розробляються окремі скрипти для кожного предмету. Тому треба зауважити, що універсальність використання нейромережевих моделей грає дуже важливу роль у подальших проектуваннях програмного продукту, так як різні дисципліни вимагають гнучкого підходу до аналізу тексту, здатного працювати як з гуманітарними текстами, так і з технічними формулюваннями.

1.2 Аналіз наявних методів автоматичного тегування тексту

Для визначення оптимальної стратегії тегування освітнього контенту було проаналізовано три категорії існуючих рішень. Цей аналіз зосереджується на хмарних NLP-сервісах, SaaS-платформах та великих мовних моделях (LLM).

Хмарні NLP-сервіси, прикладом яких є Google Cloud Natural Language API, пропонують професійну інфраструктуру та попередньо навчені моделі,

що одразу готові до застосування. Основною перевагою такого підходу є висока доступність та масштабованість без необхідності управління фізичним обладнанням. Рішення Google вирізняється високою якістю синтаксичного аналізу та розпізнавання об'єктів, підтримуючи кілька мов, зокрема українську [7]. Однак операційна модель має значні недоліки для платформи, орієнтованої на учнів. Структура цін базується на споживанні, що означає, що витрати лінійно зростають із використанням. Для платформи, що обробляє сотні завдань щодня, витрати можуть швидко стати дуже високими. Окрім того, такі API функціонують як «чорні скриньки», вони не є гнучкими і ми не можемо змінити архітектуру системи або налаштувати модель під конкретні нюанси української освітньої програми. Інтерфейс веб-демо системи зображено на рис. 1.1.

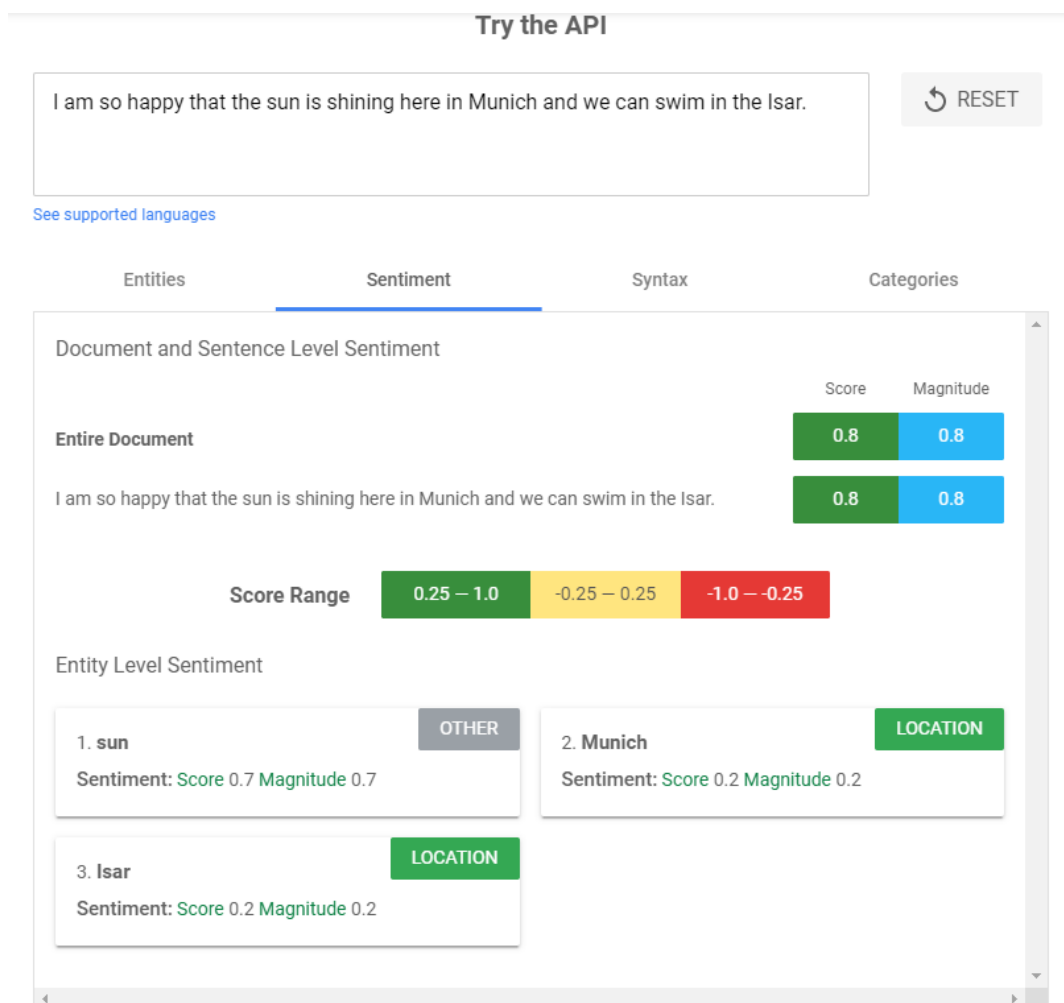


Рис. 1.1. Інтерфейс веб-демо Google Cloud Natural Language API

Платформи SaaS, такі як MonkeyLearn, забезпечують зручне для користувача середовище без кодування, призначене для бізнес-аналітики та підтримки клієнтів. Сильна сторона цього рішення полягає в його простоті використання; нетехнічний персонал може тегувати дані та навчати класифікатори за допомогою графічного інтерфейсу. Проте ця зручність супроводжується компромісом у технічному контролі та глибині інтеграції. Ці платформи часто призначені для загальних випадків, таких як аналіз настроїв у відгуках, а не для конкретної категоризації навчального контенту, необхідної для шкільних завдань. Крім того, залежність від постачальника SaaS створює потенційні проблеми з конфіденційністю даних, оскільки вони передаються на сервери третіх сторін, які можуть знаходитися за межами України, що ускладнює дотримання місцевих правил захисту даних. Інтерфейс системи зображено на рис. 1.2.

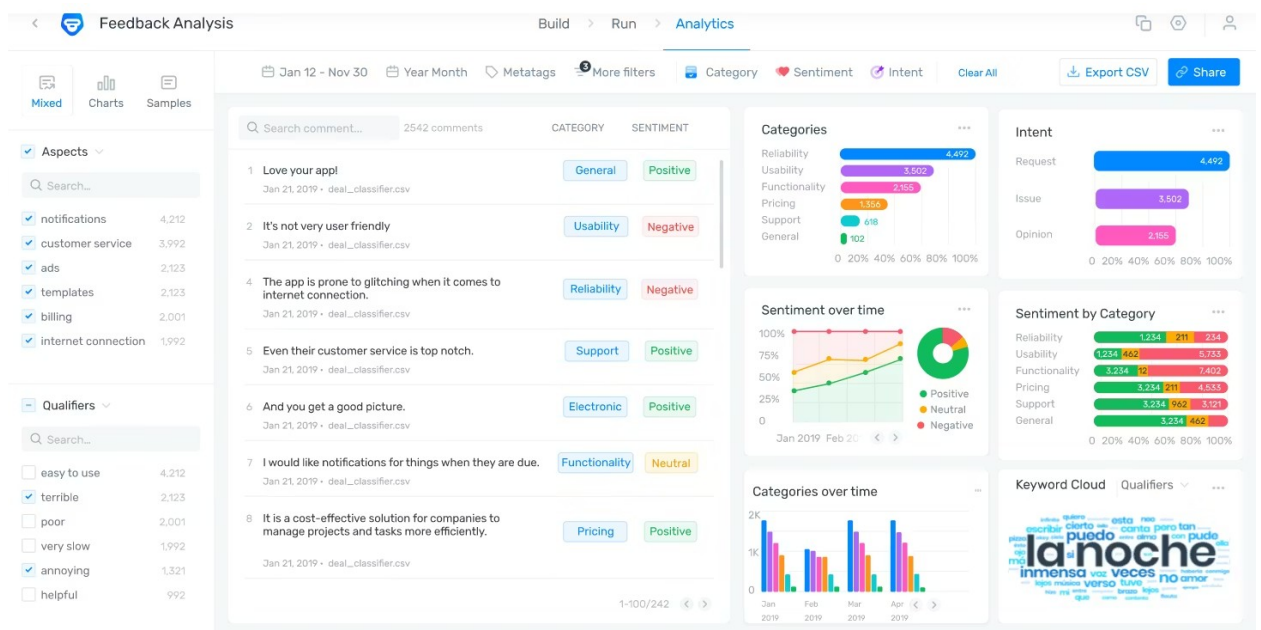


Рис. 1.2. Інтерфейс системи MonkeyLearn

Третя категорія передбачає використання великих мовних моделей, GPT-4 від OpenAI [8]. Семантичні можливості GPT-4 перевершують традиційні методи тегування, що дозволяє йому обробляти складні запити без навчання на специфічних даних, релевантних до задачі. Проте LLM не сильно підходять для для моєї задачі через неефективність використання ресурсів.

Використання моделі з трильйоном параметрів для виконання простого завдання класифікації є надмірним з точки зору обчислювальних потужностей, що призводить до високої затримки та надмірних витрат на API за кожен токен. Більше того, LLM можуть бути недетермінованими, іноді галюцинувати або надавати суперечливі теги для однакових вхідних даних, що знижує рівень надійності, необхідний для автоматизованої системи тегування в освіті [9].

Отже, рішення будувати власну систему автоматичного тегування тексту замість використання готових продуктів обґрунтовується трьома факторами: ефективністю використання ресурсів, конфіденційністю даних та специфікою сфери застосування.

Освітні платформи характеризуються великим обсягом трафіку. Якщо система обробляє 1000 завдань на день, використання API Google Cloud NLP або GPT-4 спричинить значні щомісячні витрати, які будуть зростати разом із кількістю користувачів. На відміну від цього, самостійна розробка системи дозволяє точно балансувати показники продуктивності та обчислювальної складності, що дозволить оптимізувати витрати ресурсів на її підтримку. Після розробки витрати на підтримку є незначними і обмежуються лише електроенергією та обслуговуванням хостинг-сервера, що дозволяє платформі масштабуватися без лінійного зростання витрат.

Використання SaaS або хмарних NLP-рішень вимагає передачі даних на сторонні сервери. Самостійних хостинг сервісу гарантує, що вся обробка даних відбувається на внутрішній інфраструктурі, що усуває ризики витоку даних третім сторонам.

Загальні моделі навчаються на великих, узагальнених наборах даних і можуть мати труднощі з конкретною термінологією, що зустрічається в українській шкільній програмі. Сторонні рішення можуть плутати схожі терміни з різних предметів. Створюючи індивідуальне рішення, модель можна чітко налаштувати та навчити конкретно на українських підручниках і

минулих екзаменаційних завданнях, що забезпечить вищу точність класифікації в освіті, ніж узагальнені інструменти.

1.3 Постановка задачі автоматичного тегування тексту

Проаналізувавши аналогічні сервіси, можна визначити, що система яка буде розроблятись, повинна усунути недоліки сторонніх рішень щодо вартості, гнучкості й контролю та конфіденційності.

Таким чином, технічним завданням даної роботи є розробка системи автоматичного тегування, яка дозволяє автоматично класифікувати та тегувати навчальний контент (завдання, тести) та її інтеграція до платформи виконання завдань та тестів, що дозволить оцінювати успішність учнів у розрізі кожного сформованого тегу.

Метою даної розробки є створення інструменту для моніторингу навчального процесу. Система дозволить формувати деталізований рейтинг успішності учня по кожній темі (тегу) на основі агрегації оцінок за виконанні завдання. Використовуючи гібридний підхід до обробки природної мови система надаватиме персоналізований фідбек, допомагаючи виявити слабкі та сильні сторони учня. Такий підхід зміщує фокус із загальної оцінки за предмет на компетентнісний підхід, дозволяючи вчителям адаптувати навчальну програму під реальні потреби класу.

Основними функціональними вимогами системи є:

- Автоматичний аналіз текстового вмісту завдань та присвоєння їм релевантних тегів з урахуванням морфології української мови.
- Розрахунок рейтингу учня окремо за кожним тегом на основі історії оцінювання.
- Формування рекомендацій для учнів із чітким виділенням тем, що потребують додаткового опрацювання.

- Генерація зрізів знань класу по конкретних тегах для коригування навчального плану.

Основними технічними вимогами проекту, з урахуванням обраної стратегії розробки, є:

- Модуль тегування тексту повинен бути реалізований як окремий мікросервіс, що взаємодіє з основною системою через REST API, забезпечуючи гнучкість та відмовостійкість.

- Система повинна виконувати весь цикл обробки тексту локально, без передачі даних на сторонні хмарні сервіси.

- Використання оптимізованих моделей, що дозволяють масштабувати систему без лінійного зростання витрат на інфраструктуру при збільшенні навантаження.

- Забезпечення низької затримки при обробці запитів на тегування, щоб не уповільнювати роботу інтерфейсу користувача.

Виконання даної роботи передбачає такі кроки:

- Вивчення існуючих підходів до класифікації тексту, обґрунтування вибору власної реалізації проти готових хмарних рішень [10].

- Збір навчального датасету, попередня обробка тексту та проектування API мікросервісу.

- Реалізація програмного модуля тегування, тренування нейронної мережі та інтеграція статистичних алгоритмів.

- Вбудовування мікросервісу в загальну веб-систему, перевірка точності класифікації на реальних завданнях та навантажувальне тестування.

Результатом виконання роботи повинна бути автоматична система тегування тексту, інтегрована в платформу виконання завдань та тестів. Система повинна відповідати функціональним та технічним вимогам, забезпечувати високу точність класифікації україномовного контенту та надавати аналітичні звіти для підвищення якості освітнього процесу.

1.4 Аналіз засобів тегування тексту

У сфері обробки природної мови існують три основні підходи до тегування тексту: символічний, статистичний та нейронний. Кожен з них має свої переваги, недоліки та сфери застосування. Буде виконано детальний аналіз цих підходів та їх методів для вирішення завдання тегування тексту завдань та формування рейтингу учнів за цими тегами.

Символічні методи спираються на заздалегідь визначені лінгвістичні правила, словники та граматичні структури для визначення ключових слів. Типовий процес тегування тексту використовуючи символічний підхід починається з токенізації вхідного тексту (розбиття його на окремі слова), за чим слідує видалення стоп-слів (фільтрація загальних слів, які не несуть особливого значення, наприклад, "і", "або") та лематизація/стемінг (зведення слів до їх базової (називної) форми) [11]. Наступним кроком є категоризація слів за частинами мови (POS tagging), з пріоритетом іменників. Завершальним етапом є фільтрація ключових слів за допомогою визначених лексичних та граматичних правил і словників.

Перевагою символічних методів є їх легка інтерпретованість. Користувач чи розробник може легко простежити, чому те чи інше слово було обрано як ключове, оскільки процес базується на чітко визначених правилах та словниках. Такі методи гарно працюють зі структурованими та «чистими» текстами, такими як технічні посібники чи підручники, де термінологія є стандартизованою. Однак, незважаючи на те, що символічні методи в теорії повинні гарно працювати зі структурованими текстами завдань (наприклад з математики), їхня негнучкість значно ускладнює їх використання в динамічних системах де текст генерований різними людьми та відрізняється за стилем і якістю.

Статистичні методи, в свою чергу, аналізують частоту використання певних слів та моделі їх розподілу в тексті. Цей підхід включає такі поширені

для тегування тексту техніки як: TF-IDF (Term Frequency-Inverse Document Frequency) – метод, що оцінює слова на основі їх частоти в тексті відносно їх рідкості в загальному наборі даних [12] та N-gram аналіз – метод, який ідентифікує повторювані групи слів, що використовуються в комбінації (наприклад «лінійна алгебра» чи «хімічна реакція»).

Статистичний підхід, на відміну від символічного, є більш масштабованим, незалежним від мови тексту та вимагає мінімального ручного втручання. Ці переваги роблять його гарним вибором для обробки великих обсягів даних швидко, без серйозних підготувань. Проте статистичний метод має суттєвий недолік – відсутність розуміння контексту. Для прикладу, TF-IDF метод може однаково оцінити слово «функція» в контексті математики та програмування, при цьому не враховуючи різниці між цими термінами. Аналогічно, синоніми (наприклад, «задача» та «завдання») розглядаються статистичними методами як окремі сутності, що призводить до формування надлишкових тегів. Статистичні методи можуть бути ефективними для початкової обробки тексту для пошуку найбільш часто вживаних термінів, однак їх нездатність вловлювати розуміти обмежує можливість їх самостійного використання в системі для аналізу успішності учнів.

Нейронний підхід до тегування тексту використовує моделі глибокого навчання, для розуміння значення та контексту тексту, що мають такі основні архітектури як рекурентні нейронні мережі та трансформери.

Рекурентні нейронні мережі (RNN), що оброблюють текст послідовно, фіксуючи залежності між словами. Такі варіанти як LSTM та GRU частково вирішують проблему «зникаючого градієнту» («vanishing gradient problem»), яка притаманна таким мережам, що дозволяє їм мати більш довготривалу пам'ять [13].

Трансформери – моделі, як BERT (Bidirectional Encoder Representations from Transformers, двоспрямовані кодувальні представлення з трансформерів) та GPT (Generative Pre-trained Transformer, породжувальний попередньо

тренований трансформер), що використовують механізми самоусвідомлення для двостороннього аналізу зв'язку між словами в тексті, які дозволяють мати гарний рівень розуміння контексту [14].

Нейронні моделі чудово розуміють контекст, розрізняють слова, що мають декілька значень та виявляють загальні теми. Принцип їх роботи дозволяє їм добре адаптуватися до неформального чи граматично невірного тексту, що робить їх гнучкими й ідеальними для аналізу тексту, що динамічно згенерований людьми в контексті навчального процесу. Проте, незважаючи на значні переваги даного підходу, ці моделі вимагають великих оброблених наборів даних і значних обчислювальних ресурсів для навчання. Крім того, їх натура «чорної скриньки» знижує інтерпретованість, ускладнюючи розробку та валідацію. Отже, нейронні методи гарно підходять для виявлення специфічних тегів із тексту завдань, однак при розробці потрібно враховувати вимоги такого підходу до якості навчання та обчислювальної потужності системи для аналізу успішності учнів.

Для вирішення поставленої задачі багатокласової класифікації текстів, а саме присвоєння тегів тексту, у роботі буде застосовано гібридний підхід. Цей підхід поєднуватиме класичні методи обробки природної мови для виділення ключових слів та сучасні архітектури глибокого навчання для вловлювання семантичних зв'язків у текстах. Основними інструментами аналізу виступатимуть алгоритм TF-IDF та рекурентна нейронна мережа з архітектурою Long Short-Term Memory (LSTM).

Для первинного аналізу та вилучення найбільш значущих термінів із текстів буде використано метод TF-IDF (Term Frequency-Inverse Document Frequency). Це статистичний метод, який дозволяє оцінити важливість слова в контексті документа, що є частиною загальної колекції документів (корпусу). Його ключова перевага полягає у здатності зменшувати вагу слів, які часто зустрічаються в усьому корпусі, і, навпаки, підвищувати вагу термінів, що є характерними для конкретного документа.

Розрахунок ваги за цим методом складається з двох компонентів:

Term Frequency (TF) – частота терміна. Цей показник визначає, наскільки часто певне слово зустрічається в документі. Розраховується як відношення кількості входжень слова до загальної кількості слів у документі. Це дозволяє визначити ключові слова в межах одного тексту.

Inverse Document Frequency (IDF) – обернена частота документа. Цей компонент оцінює унікальність слова в масштабах усього корпусу текстів. Він розраховується як логарифм відношення загальної кількості документів до кількості документів, у яких зустрічається дане слово. Чим у більшій кількості документів зустрічається слово, тим меншим буде його показник IDF. Це дозволяє нівелювати важливість загальновживаних слів (наприклад, сполучників або займенників, якщо вони не були видалені на етапі попередньої обробки).

Кінцева вага слова w_{ij} для терміна i в документі j обчислюється шляхом перемноження цих двох показників: $w_{ij} = TF_{ij} \times IDF_i$. Таким чином, кожен текст буде представлений у вигляді числового вектора, де кожне значення відповідає вазі конкретного слова з глобального словника. Цей підхід дозволить створити базовий набір тегів, ідентифікуючи слова з найвищими показниками TF-IDF як ключові.

Ключовою перевагою методу TF-IDF є його простота та висока швидкість обчислень. Він не потребує попереднього навчання моделі, оскільки є статистичною мірою, що дозволяє швидко отримати числове представлення тексту та виділити потенційно важливі ключові слова. Цей підхід є інтуїтивно зрозумілим і забезпечує хорошу базову ефективність для завдань інформаційного пошуку та первинної класифікації.

Водночас головним недоліком TF-IDF є повне ігнорування семантичного контексту та порядку слів. Алгоритм не розуміє синонімів або смислових зв'язків; для нього слова "сонце" та "зірка" є абсолютно різними сутностями. Це призводить до втрати значної частини сенсу тексту. Крім того,

цей метод створює вектори дуже великої розмірності та високої розрідженості, що може ускладнити роботу наступних алгоритмів машинного навчання.

Як потенційні аналоги методу TF-IDF можна розглядати простіші підходи, такі як Bag-of-Words (Мішок слів), який лише підраховує частоту входження слів, ігноруючи їхню значущість у корпусі, або більш складні методи, засновані на розподіленій семантиці.

Для глибинного семантичного аналізу структури та змісту й класифікації тексту по темах буде застосовано нейронну мережу з архітектурою Long Short-Term Memory (LSTM). LSTM є особливим видом рекурентних нейронних мереж (RNN), спеціально розробленим для ефективної роботи з послідовними даними, такими як текст. На відміну від стандартних RNN, які страждають від проблеми швидкого згасання або вибуху градієнта, LSTM здатні вивчати довгострокові залежності між елементами послідовності [15].

Ядром такої архітектури є LSTM комірка пам'яті, що керує потоком інформації за допомогою трьох механізмів: вхідний клапан, клапан забування та вихідний клапан. Ці клапани використовують сигмоїдні функції активації для видачі значень від нуля до одиниці.

Клапан забування аналізує попередній стан мережі та поточний вхідний сигнал, щоб вирішити, яку інформацію з минулого варто "забути", тобто видалити з пам'яті осередку. Вхідний клапан визначає, яка нова інформація з поточного вхідного сигналу є важливою і має бути збережена в стані осередку, а вихідний клапан, на основі поточного стану осередку, контролює, яка частина інформації буде використана для генерації вихідного сигналу на поточному кроці.

Проте лінійна обробка тексту «зліва направо» не є найкращою для завдань класифікації, де весь документ доступний одночасно. У складних текстах процес визначення значення слова часто залежить від контексту навколо нього. Тому для вирішення даної задачі буде використано

двонапрямлену LSTM (Bi-LSTM). Вона передбачає тренування двох окремих шарів LSTM: прямого та зворотного LSTM, які обробляють послідовність від початку до кінця й від кінця до початку відповідно [16]. На кожному конкретному часовому кроці приховані стани цих двох шарів об'єднуються. Це гарантує, що кінцеве представлення слова містить повний контекст до та після нього.

Оскільки мережа стискає семантичний вміст усього документа в кінцевий вектор прихованих станів, що призводить до втрати деталей, буде використано механізм уваги, який дозволяє моделі брати до уваги усі попередні приховані стани, а не лише кінцевий. Механізм працює шляхом обчислення оцінки для кожного часового кроку, які потім нормалізуються для отримання вагових коефіцієнтів уваги. Це дозволяє моделі присвоювати високу значущість найбільш релевантним словам.

В ролі активації буде використана сигмоїдна функція, що обчислює кінцеві значення незалежно для кожного класу від нуля до одиниці. Щоб зменшити ризик перенавчання алгоритм використовуватиме шари dropout для випадкового вимкнення нейронів під час навчання [17].

Завдяки такій архітектурі LSTM-мережа може вибірково зберігати інформацію протягом тривалих періодів часу, що робить її ідеальною для аналізу текстів, де значення слів та фраз часто залежить від широкого контексту, розкритого у попередніх рядках [18]. Перед подачею в LSTM-мережу слова будуть перетворені на щільні числові вектори за допомогою методів слівних вкладень (word embeddings), що дозволить моделі працювати не з окремими словами, а з їхніми семантичними представленнями.

Головною сильною стороною архітектури LSTM є її здатність вловлювати послідовну природу даних та моделювати довгострокові залежності в тексті. Завдяки своїм вентилям механізмам, мережа може "пам'ятати" важливу інформацію з попередніх частин речення чи абзацу, що є критично важливим для розуміння контексту. Працюючи з векторними

представленнями слів (word embeddings), LSTM здатна оперувати їхніми семантичними значеннями, а не просто ідентифікаторами.

Проте, така складна архітектура має і свої недоліки. Моделі на основі LSTM є обчислювально затратними, а їхнє навчання потребує значних ресурсів та часу, особливо на великих обсягах даних. Для досягнення високої точності LSTM-мережі зазвичай вимагають великих тренувальних датасетів, інакше вони схильні до перенавчання. Крім того, через послідовну обробку даних (слово за словом) вони можуть бути повільнішими за сучасні паралельні архітектури, як, наприклад, трансформери.

Альтернативами для LSTM у вирішенні цієї задачі можуть слугувати інші варіанти рекурентних мереж, наприклад, Gated Recurrent Unit (GRU), що має простішу архітектуру, але демонструє схожу ефективність. Більш сучасним та потужним аналогом є архітектури на основі механізму уваги, зокрема трансформери, такі як BERT або GPT, які дозволяють моделювати залежності між словами незалежно від їхньої відстані одне від одного в тексті.

Поєднання цих двох підходів дозволить частково нівелювати їх недоліки, у вигляді неможливості розуміння контексту для TF-IDF та значні затрати ресурсів й залежність від якості датасету для LSTM мережі.

Висновки до розділу 1

У даному розділі було проведено комплексний аналіз проблеми автоматичної класифікації навчального контенту, який підтвердив критичну важливість впровадження автоматизованих методів обробки природної мови для забезпечення адаптивності навчального процесу та формування об'єктивної звітності успішності.

Здійснено порівняльний огляд існуючих рішень, включаючи хмарні API, SaaS-платформи та великі мовні моделі, за результатами якого обґрунтовано

економічну та безпекову доцільність розробки власної системи тегування, незалежної від зовнішніх провайдерів. На основі цього сформульовано технічне завдання на проектування автономного мікросервісу, визначено його функціональні вимоги та архітектурні особливості інтеграції з основною платформою.

За підсумками аналізу символічних, статистичних та нейронних методів обробки тексту здійснено обґрунтований вибір засобів реалізації у вигляді гібридного підходу, що поєднує алгоритм TF-IDF для екстракції ключових слів та рекурентну нейронну мережу архітектури Bi-LSTM з механізмом уваги для визначення семантичних категорій.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ АВТОМАТИЧНОГО ТЕГУВАННЯ ТЕКСТУ

2.1 Вибір програмних засобів для побудови моделі тегування тексту

Оскільки система передбачає вирішення задачі класифікації з декількома мітками, для її розробки буде використано мову програмування Python, оскільки її екосистема має безліч інструментів для побудови систем на базі моделей глибокого навчання [19].

Основна обчислювальна логіка буде побудована з використанням TensorFlow та Keras API, поєднання яких є популярною платформою для створення моделей машинного навчання [20]. Для аналізу залежностей, що властиві навчальному матеріалу, де контекст теми часто залежить попередніх та наступних слів, буде реалізовано двонаправлену LSTM з модуля TensorFlow. Вона дозволить мережі обробляти текстові послідовності одночасно в прямому і зворотному напрямках, зберігаючи контекст з усього речення перед виконанням класифікації. Крім того, щоб усунути обмеження LSTM в обробці довгострокових залежностей, буде інтегровано шар Multi-Head Attention з того ж модуля. Цей компонент дозволить моделі динамічно зважувати важливість різних слів у вхідній послідовності, забезпечуючи пріоритетність ключових термінів незалежно від їх відстані один від одного [21]. Для стабілізації навчання цих рекурентних зв'язків буде використано Layer Normalization [22]. Шар вбудовувань буде ініціалізовано за допомогою попередньо навчених векторів GloVe (300-вимірних) news.lowercased.lemmatized.glove.300d від команди lang-uk, щоб надати моделі базові семантичні знання, які потім будуть уточнені під час процесу навчання [23].

Для попередньої обробки тексту, а саме видалення стоп слів та лематизації, буде використано модуль SpaCy та його uk_core_news модель для

роботи з україномовним текстом [24]. SpaCy аналізує контекст речення та визначає лему (просту форму слова). Це дозволить системі відображати слова в різних варіантах відмінювання як одну ознаку, що зменшує розрідженість словника.

Для побудови TF-IDF алгоритму буде використано Scikit-learn, а саме його TfidfVectorizer для обчислення показників частоти терміна та оберненої частоти документа [25].

Сервіс буде розроблено з використанням FastAPI, який був обраний замість таких фреймворків, як Flask або Django, завдяки вбудованій підтримці асинхронного програмування [26]. Ця можливість є важливою, оскільки вона дозволяє серверу обробляти одночасні вхідні запити без блокування основного потоку під час очікування завершення важких обчислень інференції, тим самим максимізуючи пропускну здатність.

Для забезпечення цілісності даних буде впроваджено перевірку вхідних даних за допомогою моделей Pydantic. Система автоматично відхилятиме неправильно сформовані запити, як дані досягнуть логіки інференції, запобігаючи збоєм під час виконання [27].

Додаток буде обслуговуватися Uvicorn, сервером ASGI, який керує циклом подій, необхідним для FastAPI.

Для управління даними система буде покладатися на Pandas і NumPy для матричних операцій та маніпулювання наборами даних [28]. Крім того, для ефективної обробки вбудованих даних GloVe буде використано вбудований модуль Python bz2 для читання стиснених потоків, а також pickle для серіалізації не нейронних компонентів (таких як векторизатор TF-IDF).

2.2 Підготовка даних для навчання моделі

Оскільки модель буде використана для класифікації навчальних завдань зі шкільної програми, нам потрібні релевантні дані. Було прийнято рішення

використовувати дані з ресурсу zno.osvita.ua, що містить масив завдань з зовнішнього незалежного оцінювання усіх років, що є тематично релевантним для навчання моделі, що класифікуватиме завдання в процесі навчання у школі.

Для використання цих даних, потрібно перетворити необроблений, неструктурований веб-контент у структурований, маркований формат, придатний для машинного навчання [29]. Процес починається з автоматизованого механізму скрейпінгу, який систематично проходить вміст веб-сайту. Оскільки система призначена для обробки тільки текстових вхідних даних, скрейпер аналізує HTML-структуру карток завдань і відкидає всі записи, що містять зображення, забезпечуючи сумісність отриманого набору даних з архітектурою обробки тексту мережі Long Short-Term Memory (LSTM). Для дійсних записів система витягує як текст питання, так і відповідні варіанти відповідей, комбінуючи їх, щоб надати моделі повний семантичний контекст, необхідний для класифікації.

Після отримання необробленого тексту система переходить до етапу формування набору даних, який вирішує проблему маркування. Оскільки необроблені дані, зібрані за допомогою скрейпінгу, не мають вбудованих тегів тем, система використовує евристичну стратегію маркування для їх генерації. Це досягається шляхом зіставлення конкретних тем із заздалегідь визначеними словниками регулярних виразів і ключових слів. Під час обробки необробленого тексту система сканує його на наявність цих шаблонів; коли знаходить збіг, до завдання присвоюється відповідний тег теми. Такий підхід дозволяє автоматично створювати набір даних із декількома мітками, де одне завдання може бути пов'язане з декількома відповідними темами. Такий підхід можливий через високий рівень типовості завдань ЗНО, більшість з них починається з стандартизованого окреслення завдання, що містить слова мітки, такі як: «кома», «синонім», «фразеологізм», тощо.

В процесі використання набору даних в навчанні він проходить ретельну попередню обробку та векторизацію, призначену для нормалізації української мови. Текст очищується шляхом видалення шуму, такого як розділові знаки, цифри та стоп-слова, які не впливають на семантичне значення завдання класифікації. Крім того, система проводить лематизацію слів, зводячи їх до канонічних форм. Цей крок скорочує розмір словника і гарантує, що варіації одного і того ж слова обробляються однаково, тим самим покращуючи здатність моделі до узагальнення. Далі, цей нормалізований текст перетворюється на числові послідовності на основі попередньо навчених вбудовувань слів, надаючи нейронній мережі щільне представлення навчального контенту.

2.3 Побудова моделі тегування тексту

LSTM був розроблений як градієнтно-орієнтований метод, який може вивчити, як зберігати інформацію протягом тривалих інтервалів часу. Цей метод забезпечує постійний потік помилки (CEC) через спеціальні внутрішні одиниці, ефективно запобігаючи руйнуванню градієнта та дозволяючи моделі долати мінімальні часові затримки, що перевищують 1000 дискретних кроків.

Ядром архітектури LSTM є її комірка (рис. 2.1), яка вміщує клітинний стан та систему з трьох мультиплікативних вентилів, які регулюють, як інформація зберігається та обробляється. Клітинний стан C_t діє як конвеєр пам'яті, що проходить через комірку. Він здатний зберігати інформацію протягом довільних часових інтервалів. На відміну від прихованого стану традиційних RNN, оновлення клітинного стану відбувається здебільшого через додавання, а не множення. Саме ця адитивна взаємодія реалізує CEC, оскільки градієнт може текти безперешкодно (з вагою 1), якщо відповідний вентиль забування відкритий.

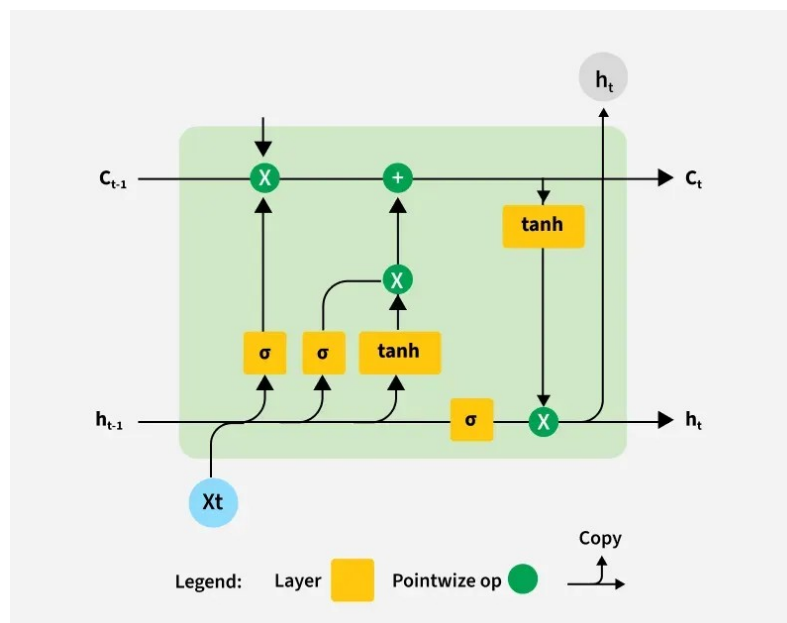


Рис. 2.1. Структура комірки пам'яті LSTM

У LSTM використовуються дві основні нелінійні функції: сигмоїда та гіперболічний тангенс. Сигмоїда (σ) використовується в усіх трьох вентилях (забування, входу та виходу). Її вихідний діапазон дозволяє їй діяти як керуючий "вентиль", визначаючи, яка частка інформації має бути пропущена (значення ≈ 1) або відкинута (значення ≈ 0). Гіперболічний тангенс ($\tanh$) використовується для генерації кандидатського стану. $\tanh$ обмежує значення між $[-1, 1]$, що допомагає регулювати та нормалізувати нелінійні характеристики нового контенту, який потенційно буде доданий до клітинного стану. Перевага такої архітектури полягає у стратегічному поділі: нелінійні вентилях (σ) контролюють потік, тоді як основний механізм оновлення пам'яті залишається лінійним, забезпечуючи стабільність градієнта.

Роботу LSTM-комірки можна описати п'ятьма ключовими рівняннями, які визначають, як вхідний сигнал x_t та попередні стани h_{t-1} та C_{t-1} трансформуються у поточний стан C_t та прихований стан h_t (табл. 2.1). У формулах W позначає матриці ваг, b — вектори зміщення, а $\odot$ поелементне (Адамарове) множення.

Формули, що описують роботу LSTM комірки

Вентиль/Оновлення	Формула	Пояснення
Вентиль забування (f_t)	$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$	Регулює збереження старого клітинного стану C_{t-1} . Значення f_t вирішує, яку інформацію слід забути
Вентиль входу (i_t)	$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$	Визначає, які нові дані є достатньо важливими для оновлення клітинного стану.
Кандидатський стан ($\hat{C}_t$)	$\hat{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$	Створює новий потенційний контент для додавання до пам'яті. Використання $\tanh$ обмежує його діапазон, стабілізуючи систему.
Оновлення клітинного стану (C_t)	$C_t = f_t \odot C_{t-1} + i_t \odot \hat{C}_t$	Старий стан C_{t-1} фільтрується f_t , а новий фільтрований контент $i_t \odot \hat{C}_t$ додається. Операція додавання є механізмом, що забезпечує стабільний потік градієнта.
Вентиль виходу (o_t)	$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$	Регулює, яку частину внутрішньої пам'яті C_t слід викрити та використати для формування прихованого стану.
Прихований стан (h_t)	$h_t = o_t \odot \tanh(C_t)$	Фінальний вихід комірки, що передається до наступного шару або наступного часового кроку.

Оскільки поставлена задача вимагає глибокого аналізу контексту всього речення, модель має архітектуру двонапрявленої обробки (Bi-LSTM). Bi-LSTM складається з двох незалежних рекурентних підмереж: одна обробляє вхідну послідовність у прямому напрямку (від t_1 до t_T), а інша у зворотному напрямку (від t_T до t_1) (рис. 2.2). На кожному часовому кроці виходи обох напрямків (приховані стани) об'єднуються, зазвичай шляхом конкатенації, суми або множення.

Така архітектура дозволяє розуміти як минулий, так і майбутній контекст, що корисно для поставленої задачі. На відміну від стандартного LSTM, який може використовувати лише попередній контекст, Bi-LSTM забезпечує повний контекстний огляд. Це підвищує якість представлення даних, хоча і подвоює розмірність прихованого стану, збільшуючи загальну ємність моделі.

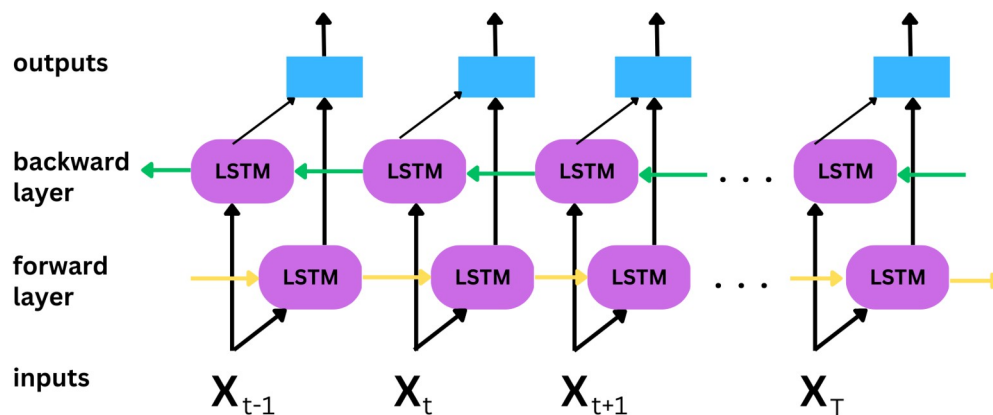


Рис. 2.2. Логіка роботи Bi-LSTM

LSTM-шари вимагають, щоб вхідні дані були представлені у вигляді тривимірного тензора, що відповідає формату: розмір пакета (batch size), довжина послідовності (sequence length), розмірність ознак (feature dimension). Розмірність цих ознак визначає складність вхідного вектора на кожному часовому кроці. Ці ознаки передаються в механізм уваги, що був

імplementований, щоб запобігти проблемі «вузького місця», коли LSTM-методи не можуть зберегти певні деталі в довгих послідовностях. Цей механізм дозволяє моделі динамічно зважувати важливість різних слів у послідовності незалежно від їхньої відстані, ефективно дозволяючи моделі "фокусуватися" на ключових словах, що стосуються певних тем.

Для числових часових рядів або будь-яких вхідних ознак, що не є вбудовуваннями, обов'язковим кроком є масштабування або нормалізація даних. Оскільки LSTM використовує загальні матриці ваг, що обробляють ознаки з потенційно різними діапазонами, велика різниця у масштабі вхідних даних може ускладнити збіжність і призвести до нестабільності градієнтів. Масштабування, забезпечує, що всі ознаки рівномірно впливають на процес навчання, сприяючи більш швидкому та стабільному досягненню глобального мінімуму.

Розмір прихованого стану (units або n_h) у LSTM-шарі прямо впливає на кількість параметрів та, відповідно, на ємність моделі. Висока кількість прихованих одиниць (наприклад, 128 або 256) дозволить моделі захоплювати значну кількість складної інформації та довготривалих залежностей у послідовності. Це збільшує потенціал для отримання точних результатів, особливо для складних наборів даних. Однак, більша кількість одиниць також збільшує обчислювальну вартість, а також підвищує ризик перенавчання, якщо не застосовуються належні механізми регуляризації. З оглядом на розмірність та комплексність даних для навчання, було визначено, що найоптимальнішим розміром прихованого стану для даної задачі є 64.

Дані 3D-послідовності згортаються у 2D-вектор, що усереднює вектори ознак за всіма часовими кроками, створюючи стислий виклад усього тексту. Цей виклад проходить через повністю зв'язаний (щільний) прихований шар з активацією relu , а потім через кінцевий вихідний щільний шар з сигмоїдною активацією.

Регуляризація є необхідною частиною архітектури глибоких моделей, для підвищення їхньої здатності до узагальнення. Архітектура моделі застосовує механізм dropout, що застосовує маскування (скидання) до одиниць, які беруть участь у лінійному перетворенні вхідного тензора (x_t). Це стандартна форма регуляризації, яка допомагає запобігти коадаптації нейронів щодо вхідних ознак.

В ролі оптимізатора було обрано оптимізатор Adam (Adaptive Moment Estimation), що є стандартним вибором для глибоких мереж завдяки його адаптивному підходу. Adam поєднує переваги Momentum (прискорення збіжності) та RMSprop (адаптивне коригування швидкості навчання для кожного параметра). Adam ефективний у контексті RNN, оскільки він розраховує адаптивні швидкості навчання, ґрунтуючись на оцінках першого (середнє значення) та другого (дисперсія) моментів градієнтів. Ця властивість робить його стійким до шумних градієнтів, які часто виникають у складних рекурентних моделях. Його здатність до швидкої адаптації до нестационарних проблем, характерних для послідовних даних, забезпечує швидку та стабільну збіжність [30].

Функцією втрат моделі є Binary Cross-Entropy, що вимірює розбіжність між передбаченою ймовірністю та істинною міткою. Функція втрат порівнює точність моделі на кожному кроці навчання та передає інформацію в наступний крок.

Перед початком навчання даної моделі, вихідні дані проходять попередню обробку. Основною метою цього процесу є зменшення розмірності вхідних даних із збереженням семантичного сигналу, необхідного для точної класифікації тем. Необроблений текст не можна безпосередньо вводити в модель, оскільки це призведе до розрідженого та некерованого розміру словника.

Процес починається з нормалізації, система спочатку перетворює весь текст у нижній регістр і систематично видаляє «шумові» символи, включаючи

розділові знаки та цифри. Після цього очищення система застосовує морфологічний аналіз для лематизації тексту. На відміну від англійської, українські слова значно змінюють форму залежно від відмінка, роду та числа; без лематизації модель розглядала б варіації одного і того ж слова (наприклад, «атом», «атоми», «атомний») як повністю не пов'язані між собою лексеми, що розмивало б сигнал навчання. Зводячи кожне слово до його канонічної форми (лемми) і відфільтровуючи стоп-слова, система значно ущільнює словник і покращує співвідношення сигнал/шум. Також відкидаються рядки, що не містять тегів.

Після лінгвістичної нормалізації текст проходить векторизацію для перетворення лінгвістичних лексем у послідовності цілих чисел, які здатна сприймати модель. Система адаптує векторизаційний шар до навчального корпусу, ідентифікуючи найчастіші терміни для побудови словника фіксованого розміру. Обмеження розміру словника запобігає надмірному пристосуванню моделі до рідкісних, неінформативних слів. Оскільки нейронні мережі вимагають вхідних тензорів однорідної форми для ефективної пакетної обробки, ці послідовності цілих чисел стандартизуються до фіксованої довжини. Послідовності, коротші за поріг, доповнюються нулями, а довші послідовності обрізаються, що забезпечує LSTM отримання послідовної структури даних.

Після цих кроків система готує цільові змінні та семантичні особливості за допомогою спеціалізованих технік кодування. Оскільки проблема відноситься до проблем класифікації з декількома мітками (де одне завдання може належати до декількох тем), текстові теги перетворюються у числові формати, сумісні з нейронною мережею – бінарні вектори. Це дозволяє вихідному шару одночасно прогнозувати кілька класів, що не виключають одне одного. Щоб покращити семантичне розуміння моделі, слова, закодовані цілими числами, проєктовані на попередньо навчені вбудовування (вектори GloVe). Цей крок використовує трансферне навчання, вводячи в модель

зовнішні семантичні знання і дозволяючи їй розуміти відносини між словами, навіть якщо вони рідко зустрічаються в конкретному наборі даних. При навчанні, дані розділяються на тренувальну та валідаційну вибірку. Тренувальна вибірка становить 80% від загального розміру вибірки та використовується для тренування моделі. Валідаційна вибірка, в свою чергу, становить 20% від загального розміру вибірки та використовується для валідації результатів та внесенні змін в процесі тренування.

Після підготовки даних і завантаження попередньо навчених вбудованих елементів GloVe для ініціалізації розуміння моделлю семантичних відносин, система розпочинає процес навчання за допомогою оптимізатора Adam. Щоб запобігти запам'ятовуванню моделі навчальних даних, або перенавчанню, цикл навчання включає специфічні механізми зворотного виклику, які в режимі реального часу контролюють продуктивність на валідаційному наборі. Для спостереження за валідаційними втратами використовується зворотний виклик раннього зупинення; якщо модель не покращується після визначеного періоду у п'ять епох, навчання автоматично зупиняється, а ваги повертаються до свого найкращого стану.

Доповненням до механізму раннього зупинення є механізм зменшення швидкості навчання. Якщо втрата валідації стабілізується, але не погіршується настільки, щоб викликати повне зупинення, цей зворотний виклик зменшує швидкість навчання в 0,5 рази. Це дозволяє оптимізатору робити менші, більш точні кроки в просторі ваг, допомагаючи моделі досягти більш глибокого або стабільного локального мінімуму, який міг бути пропущений при більш високій швидкості навчання. Після завершення циклу навчання логіка забезпечує відтворюваність і готовність до розгортання, серіалізуючи не тільки навчені ваги моделі, але й конкретні екземпляри токенизатора та бінаризатора міток, що використовуються, гарантуючи, що дані виведення проходять точно такі самі перетворення, як і дані навчання.

Процес навчання візуалізується за допомогою кривих втрат і точності, які показують, як мережа Long Short-Term Memory (LSTM) адаптується до даних протягом навчання. Також формується звіт про класифікацію, що показує ефективність моделі за допомогою трьох ключових показників: точності, відтворюваності та F1-балу.

Розглянемо приклад кривих втрат (рис. 2.1) й точності (рис. 2.2) та звіту про класифікацію (рис. 2.3), що були сформовані при навчанні моделі на даних з предмету «Українська мова та література».

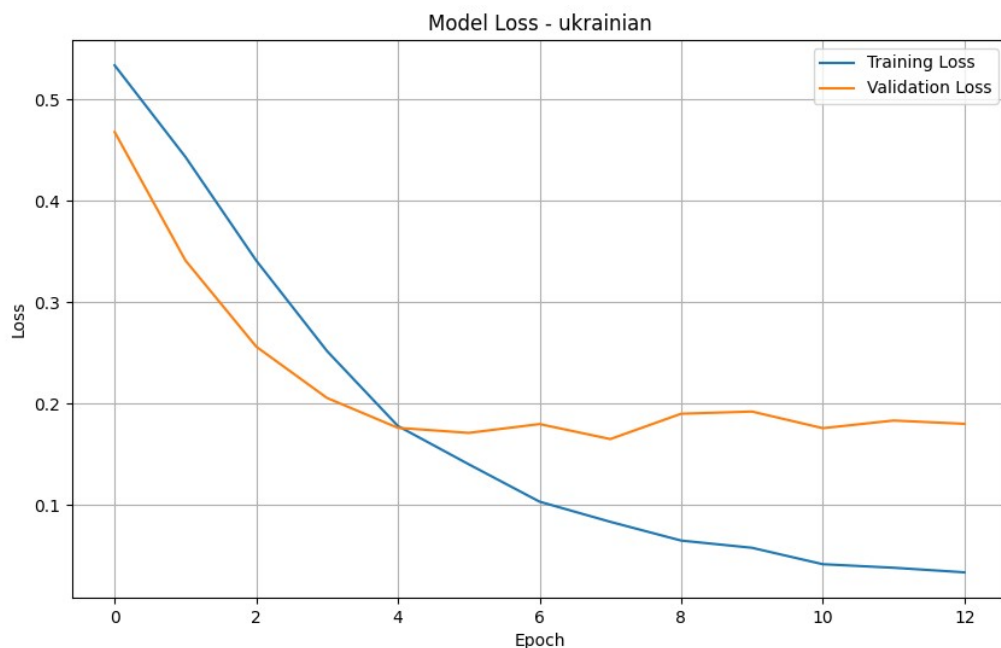


Рис. 2.3. Крива втрат

Крива втрат вказує на те, що модель ефективно навчається на початковому етапі, при цьому втрати під час навчання і валідації швидко зменшуються. Однак близько сьомої епохи відбувається зсув. Хоча втрати під час навчання продовжують наближатись до нуля, що вказує на те, що модель запам'ятовує дані навчання, втрати під час валідації, що стабілізувались близько четвертої епохи, починають збільшуватися. Це вказує на початок перенавчання, коли модель починає втрачати здатність узагальнювати нові, невидимі дані, незважаючи на те, що продовжує вдосконалюватися на навчальному наборі.

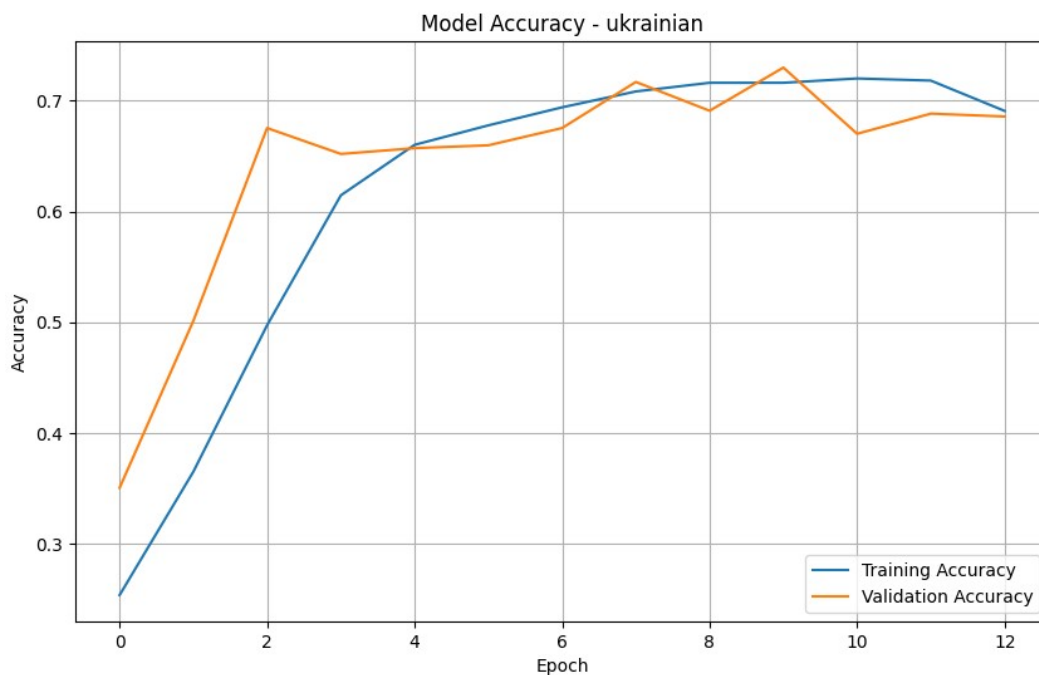


Рис. 2.4. Крива точності

Крива точності підтверджує це спостереження. Точність навчання стабільно зростає протягом сесії, досягаючи приблизно 70%, тоді як точність валідації стабілізується, коливаючись в межах 65-70%. Така поведінка свідчить про те, що модель швидко вивчає відмінні риси категорій і зберігає свою ефективність на невидимих даних, не плутаючись з порожніми або шумними прикладами.

Subject: ukrainian

	precision	recall	f1-score	support
Лексика та фразеологія	0.83	0.71	0.77	28
Література: Твори та автори	0.94	0.87	0.90	161
Література: Теорія	0.86	0.87	0.86	127
Морфологія	0.89	0.88	0.88	73
Синтаксис та пунктуація	0.98	0.75	0.85	76
Стилістика та текст	1.00	0.95	0.97	115
Фонетика та правопис	0.93	0.95	0.94	114
micro avg	0.93	0.88	0.90	694
macro avg	0.92	0.85	0.88	694
weighted avg	0.93	0.88	0.90	694
samples avg	0.89	0.87	0.87	694

Рис. 2.5. Звіт про класифікацію

Звіт про класифікацію описує ефективність моделі за допомогою трьох ключових показників: точності, відтворюваності та F1-балу. Точність вимірює надійність моделі при прогнозуванні конкретного тегу; високий бал точності означає, що коли модель стверджує, що текст належить до певної категорії, вона зазвичай має рацію. Відтворюваність оцінює здатність моделі знаходити всі відповідні екземпляри категорії в наборі даних. F1-Score надає гармонійне середнє цих двох показників, пропонуючи єдиний показник.

У цьому конкретному експерименті модель демонструє гарну продуктивність у більшості категорій. Категорія «Стилістика та текст» досягла найвищих результатів, з ідеальною точністю 1 та F1-показником 0,97, що вказує на те, що модель легко розрізняє ці тексти. Категорії «Література: твори та автори» і «Фонетика та правопис» також демонструють високі результати з F1-scores 0,90 і вище. Найнижчі результати, хоча й прийнятні, спостерігаються в категорії «Лексика та фразеологія» з F1-показником 0,77. Однак це безпосередньо пов'язано з низькою кількістю елементів з цією категорією в навчальних даних – лише 28 зразків, що не достатньо для повного узагальнення.

Особливу поведінку слід відзначити в категорії «Синтаксис і пунктуація». Хоча вона має винятково високу точність 0,98, її відтворюваність нижча – 0,75. Це вказує на консервативну поведінку, коли модель рідко помилково позначає текст як пов'язаний із синтаксисом, але пропускає близько 25% фактичних прикладів синтаксису, ймовірно, через перетинання з іншими категоріями.

Натренувавши модель, можна перейти до виконання передбачення тегів. Для цього система завантажує артефакти конкретного предмету, створені під час навчання, включаючи навчену модель Keras, мультилейбовий бінаризатор для відображення бінарних векторів назад у текстові мітки та векторизатор TF-IDF для вилучення ключових слів. Щоб забезпечити отримання моделлю даних у форматі, ідентичному тому, який вона бачила під

час навчання, необроблений вхідний текст проходить точно таку саму попередню обробку: очищення, переведення в нижній регістр та лематизацію, перед перетворенням у тензор.

Після попередньої обробки та векторизації текст проходить через нейронну мережу. Модель видає вектор значень з плаваючою комою від 0 до 1, що відповідають ймовірностям кожного тегу. Цей формат виводу визначається використанням сигмоїдної функції активації в кінцевому щільному шарі. Потім система інтерпретує ці ймовірності. Спочатку вона намагається ідентифікувати релевантні теми, застосовуючи поріг впевненості (налаштований на 0,4) до вектора виходу; будь-який клас із ймовірністю, що перевищує це значення, додається до набору результатів. Однак якщо жодна тема не перевищує поріг впевненості, що може статися з коротким або зашумленим текстом, логіка за замовчуванням вибирає єдину тему з найвищою ймовірністю. Це гарантує, що API завжди повертає корисний прогноз, а не порожній набір.

Паралельно з нейронною класифікацією система виконує окремий процес вилучення для ідентифікації конкретних ключових слів. Вона використовує попередньо навчений векторний перетворювач TF-IDF для перетворення лематизованого тексту в розріджену матрицю та обчислення балів важливості для окремих слів. Логіка фільтрує ці слова за вторинним порогом і сортує їх за балами, повертаючи п'ять найважливіших термінів. Цей подвійний підхід надає кінцевому користувачеві як високий рівень категоріальної класифікації (за допомогою LSTM), так і детальні дескриптори вмісту (за допомогою TF-IDF), що дає в результаті комплексний набір семантичних тегів.

Висновки до розділу 2

У цьому розділі було виконано проєктування та програмну реалізацію системи автоматичного тегування навчального контенту, починаючи від вибору технологічного стеку і завершуючи інтеграцією мікросервісу в робоче середовище платформи. Для вирішення задачі класифікації текстів було обґрунтовано використання мови Python та бібліотек глибокого навчання TensorFlow і Keras, що дозволило побудувати архітектуру Bi-LSTM та використовувати в поєднанні з нею попередньо навчені векторних представлення GloVe та лематизацією через SpaCy. Побудована модель здатна враховувати семантичний контекст та довгострокові залежності у реченнях українською мовою.

Було описано процес підготовки спеціалізованого навчального набору даних на основі матеріалів ЗНО, для чого було реалізовано алгоритми скрейпінгу, очищення від шуму та евристичної розмітки. Навчання моделі супроводжувалося застосуванням стратегій регуляризації та оптимізації гіперпараметрів, що в результаті дозволило досягти високих показників F1-score для ключових категорій завдань. Для забезпечення повноти аналізу було впроваджено гібридний підхід до інференції, який поєднує нейромережеву класифікацію тем із статистичним виділенням ключових слів методом TF-IDF.

РОЗДІЛ 3. ОПИС ПРОГРАМНОГО ПРОДУКТУ

3.1 Інтеграція системи автоматичного тегування тексту до платформи виконання завдань та тестів

Для інтеграції функціональності автоматичного тегування тексту в платформу виконання завдань та тестів необхідно реалізувати відокремлену систему звернення до цієї функції. З огляду на невелику комплексність самої функції тегування, але високі обчислювальні затрати ресурсів, що виникають через необхідність завантаження та роботи з моделями машинного навчання, найбільш доцільним архітектурним рішенням є побудова системи як окремого мікросервісу. Мікросервісна архітектура являє собою стиль, де системи організовано як невеликі, автономні сервісів, кожен з яких зосереджений на одній конкретній функції, а взаємодія між ними відбувається за допомогою легких протоколів, як HTTP [31].

Такий підхід дозволяє ізолювати важкі обчислювальні вимоги та складні програмні залежності, характерні для середовищ TensorFlow та Scikit-learn, від основної логіки платформи. Ця ізоляція забезпечує стабільність, масштабованість та незалежне розгортання сервісу тегування. Функціональність мікросервісу буде надаватися через стандартизований RESTful-інтерфейс, що буде розроблений на базі високопродуктивного фреймворку FastAPI, що гарантує швидку та надійну комунікацію з основним застосунком платформи [32].

Сервіс забезпечує дотримання цілісності даних за допомогою моделей Pydantic, які слугують формальною схемою взаємодії між сервером і мікросервісом. Вхідні дані перевіряються, щоб переконатися, що вони містять необхідний текст і дійсний предмет з конфігурації, перш ніж запит досягне логіки обробки. Цей рівень перевірки запобігає марнуванню обчислювальних ресурсів системи на неправильно сформовані запити і повертає негайний

структурований відгук, якщо вхідні дані порушують визначені обмеження. Вихідні дані серіалізуються у передбачуваний формат JSON, що містить лематизований текст, передбачувані теми та ключові слова, що дозволяє сервісам нижчого рівня надійно використовувати дані без знання внутрішніх структур класів.

Для управління значними витратами ресурсів, властивим моделям машинного навчання, мікросервіс реалізує інтелектуальну стратегію управління життєвим циклом за допомогою асинхронності. При запуску програми система попередньо завантажує в пам'ять артефакти з високим пріоритетом, зокрема українські тематичні моделі, щоб запобігти затримці, пов'язаній з «холодним запуском» під час початкових запитів. Це поєднується з патерном синглтона в службі інференції, яка підтримує постійний словник завантажених ресурсів, дозволяючи іншим предметам завантажуватися ліниво тільки за запитом [33]. Ця конструкція оптимізує обсяг використаної пам'яті, що є важливим фактором при розгортанні мікросервісів в середовищах з обмеженими ресурсами.

Операційна спостережуваність і відмовостійкість забезпечуються за допомогою спеціальних кінцевих точок і структурованої обробки винятків. Сервіс надає спеціальну кінцеву точку перевірки працездатності, яка повертає стан системи і список завантажених на даний момент моделей. Внутрішньо API абстрагує складність конвеєра інференції, виявляючи конкретні помилки, такі як відсутність файлів моделей або порожні вхідні дані, і зіставляючи їх з відповідними кодами статусу HTTP, такими як 404 або 400. Ця інкапсуляція гарантує, що складна логіка попередньої обробки, векторизації та прогнозування LSTM залишається повністю прихованою.

Загальна система тепер включає розроблений мікросервіс, що відображено на діаграмі розгортання (рис. 3.1). Він є повністю автономним і його єдиною відповідальністю є отримання текстових даних

(завдання/питання) через RESTful-інтерфейс та повернення згенерованих тем і ключових слів.

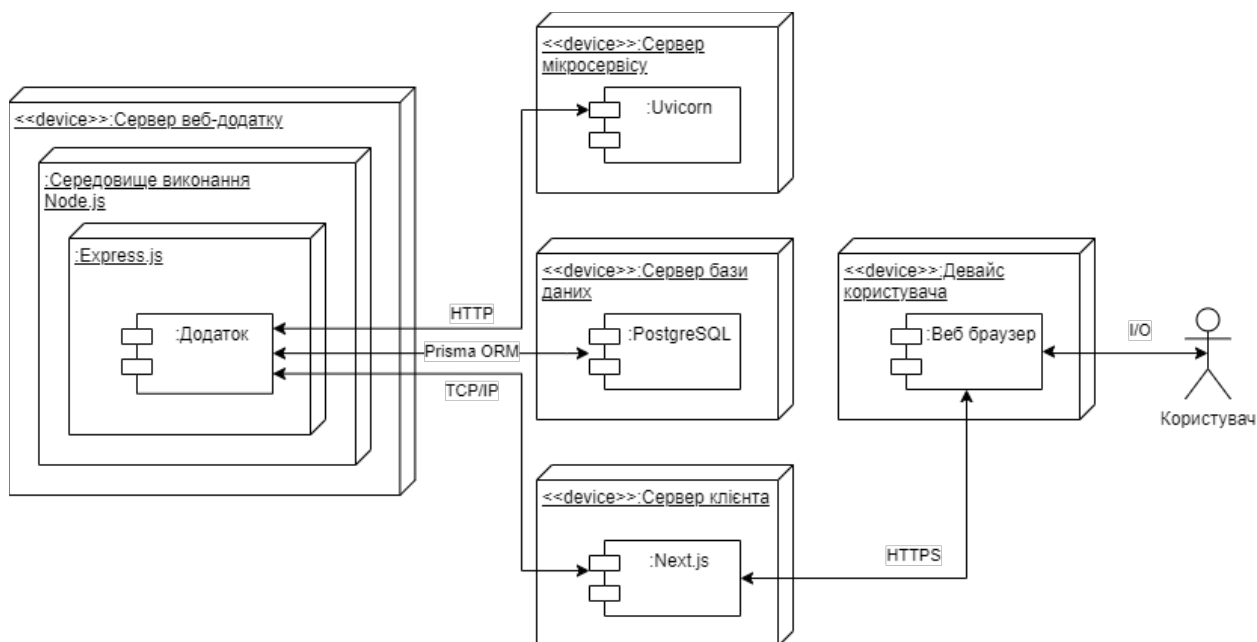


Рис. 3.1. Діаграма розгортання

Центральною ланкою серверної логіки виступає сервер веб-додатку на базі середовища виконання Node.js [34] та фреймворку Express.js, який координує обробку запитів та взаємодіє з іншими частинами інфраструктури. Зберігання та управління даними покладено на окремий сервер бази даних PostgreSQL [35], доступ до якого здійснюється через Prisma ORM [36], тоді як важкі обчислювальні задачі делегуються автономному серверу мікросервісу під управлінням Uvicorn через внутрішні HTTP-запити. Клієнтська частина реалізована на виділеному сервері з технологією Next.js [37], що комунікує з основним бекендом по протоколу TCP/IP, надаючи інтерфейс кінцевому користувачеві, який взаємодіє із системою через веб-браузер свого пристрою за допомогою захищеного протоколу HTTPS.

Перед тим як безпосередньо інтегрувати та застосовувати систему автоматичного тегування тексту на платформі виконання завдань та тестів, необхідно чітко визначити, які саме задачі ця система автоматичного тегування має виконувати. Це не просто технічна інтеграція, а стратегічний крок, що має вирішувати конкретні бізнес-потреби. Для цього я проаналізую

поточну діаграму варіантів використання платформи виконання завдань та тестів (рис. 3.2), що допоможе зрозуміти ключових акторів системи, їхні основні взаємодії з контентом та точки інтеграції, де функція автоматичного тегування буде найбільш корисною та ефективною. Аналіз діаграми дозволить ідентифікувати, у який момент життєвого циклу завдання (від його створення викладачем до використання учнем) потрібно викликати сервіс тегування, а також яким чином результати тегування (теми та ключові слова) вплинуть на інші варіанти використання, такі як "Пошук завдання", "Формування персоналізованого рейтингу" або "Налаштування тесту".

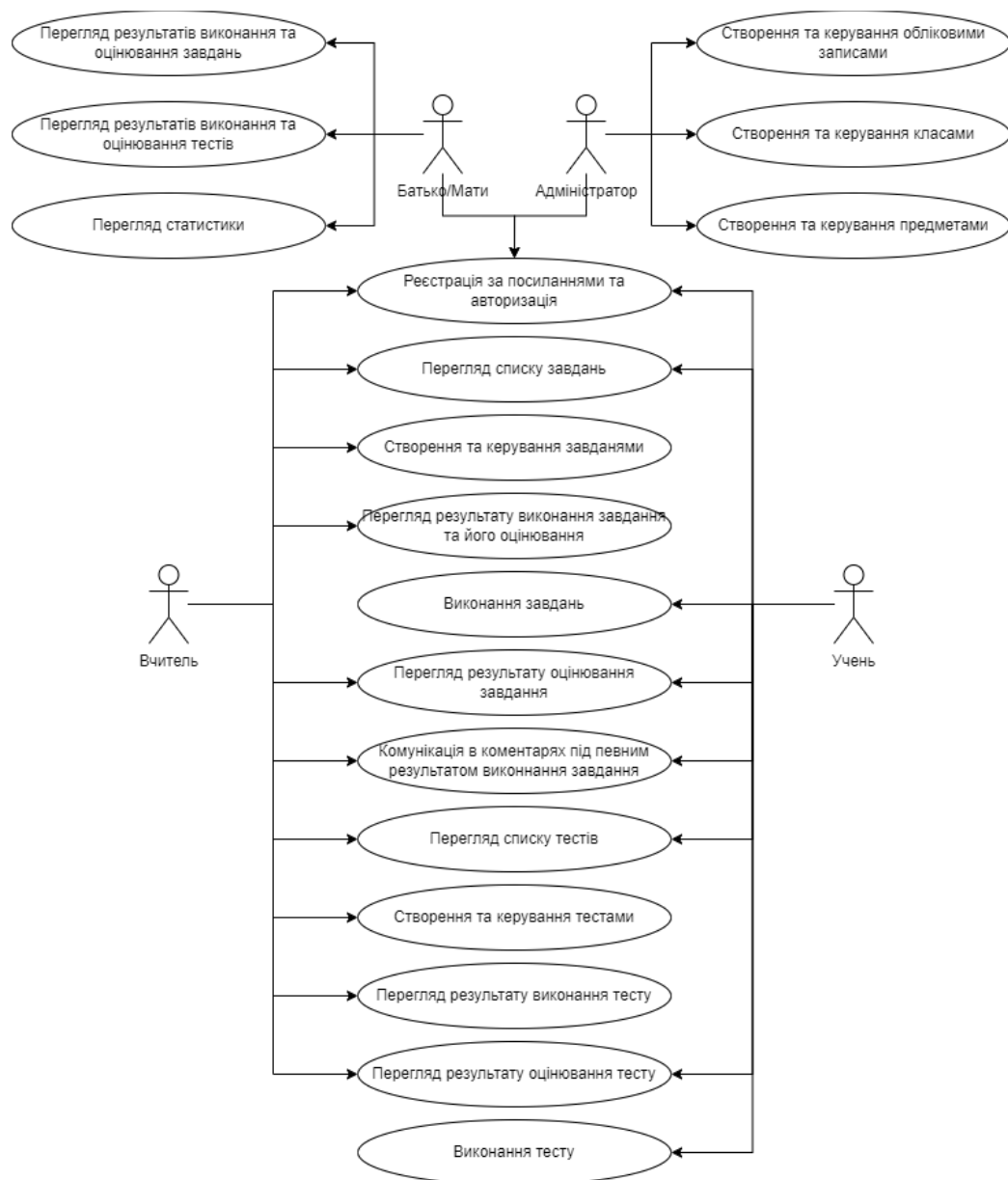


Рис. 3.2. Діаграма варіантів використання платформи

Розуміння цих взаємозв'язків забезпечить, що нова система автоматизації буде органічно вбудована у вже існуючу логіку роботи платформи, уникаючи конфліктів та максимально підвищуючи зручність і функціональність для всіх користувачів.

У цій діаграмі фігурує 4 учасника процесу: Адміністратор, Вчитель, Учень та Батько/Мати. Проаналізувавши цю діаграму, можна зробити висновок, що використання системи автоматичного тегування тексту найбільш релевантне у випадку створення завдань та тестів (для передбачення теми та присвоєння ключових слів), перегляду зведеної успішності за темами та при пошуку завдань, використовуючи для цього сформовані ключові слова. Сформулю діаграму варіантів використання платформи ексклюзивно для дій, що передбачають використання системи автоматичного тегування тексту (рис. 3.3).

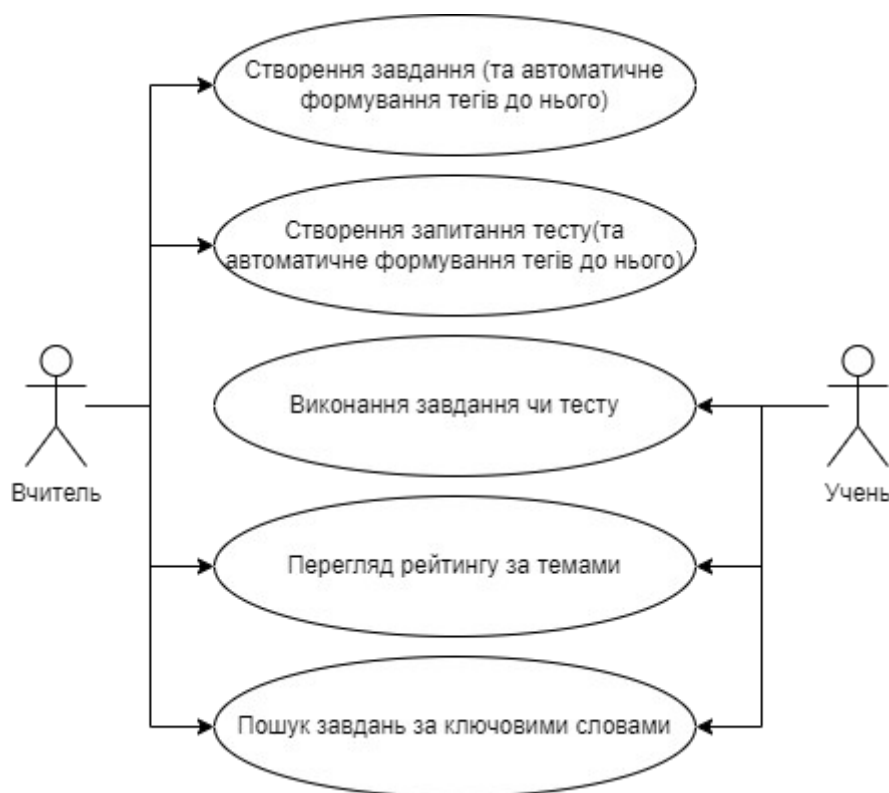


Рис. 3.3. Діаграма варіантів використання для задач пов'язаних з використанням системи тегування

Центральною функцією є автоматичне формування тем та ключових слів в момент створення нового завдання чи запитання тесту. Сформовані теги використовуються для побудови персоналізованих рейтингів за темами для кожного учня, що дозволяє відстежувати успішність кожного учня не лише за загальним балом, але й за конкретними предметними областями. Цей рейтинг може бути переглянутий як самим учнем для виявлення своїх сильних сторін та прогалин у знаннях, так і вчителем для цільового коригування навчального плану та надання допомоги.

Сформовані ключові слова дозволяють покращити функціонал пошуку на платформі. Користувачі можуть використовувати автоматично присвоєні ключові слова при пошуку завдань, що дає можливість виконувати пошук не лише за назвою завдання, а й за вмістом, хоч і опосередковано через теги.

Для формування сутностей, що будуть брати участь в процесі генерації тегів та їх використання, проаналізуємо наявну діаграму сутностей (рис. 3.4).

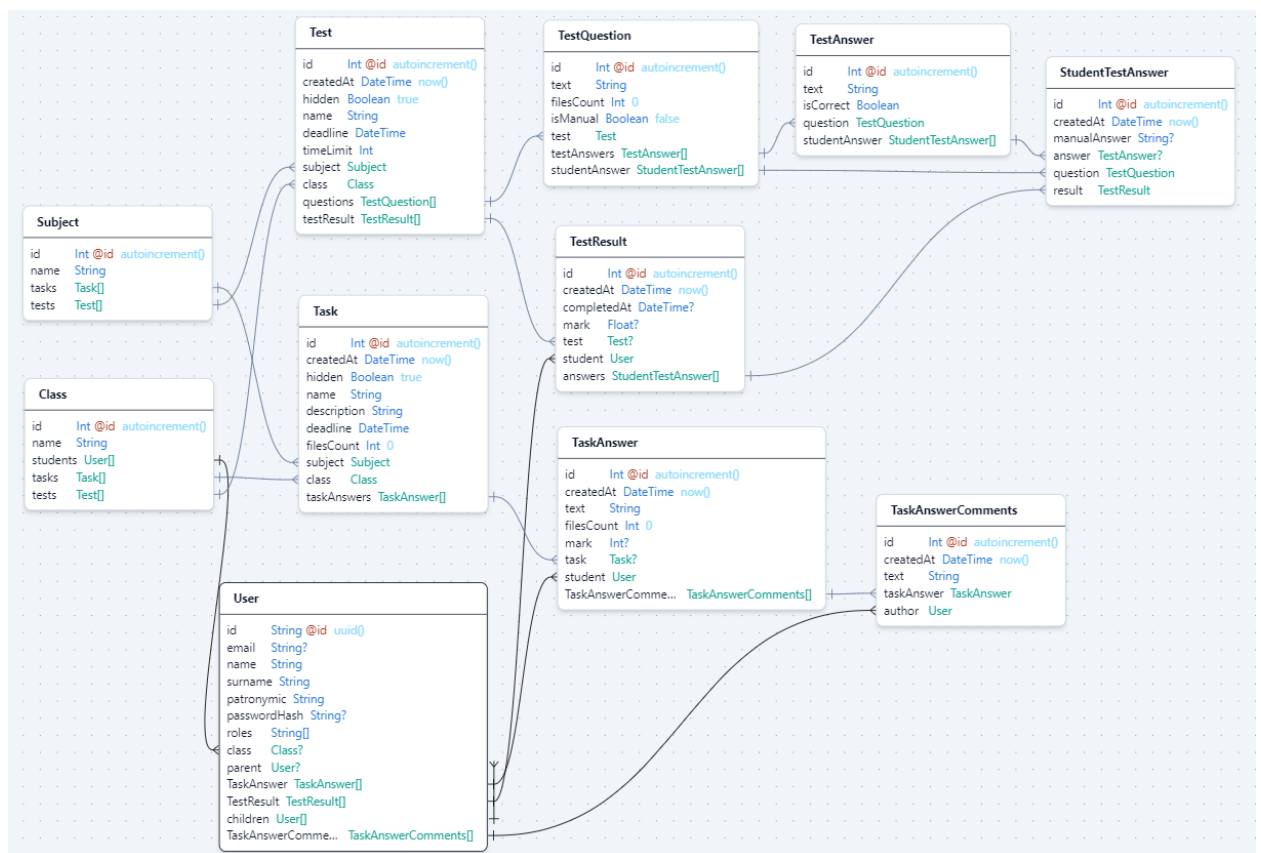


Рис. 3.4. Діаграма сутностей системи

Для інтеграції системи автоматичного тегування тексту в архітектуру платформи, потрібно внести зміни до її моделі даних (рис. 3.5). Ключові сутності, що оперують текстовим контентом Task (Завдання) та TestQuestion (Питання тесту) мають бути розширені новою властивістю, призначеною для збереження згенерованих ключових слів. Це забезпечить швидкий пошук та навігацію, як було визначено раніше.

Крім того, буде введена нова незалежна сутність Topic (Тема). Ця сутність буде слугувати центральним репозиторієм для всіх унікальних тем, які ідентифікує система тегування. Важливою складовою сутності Topic є властивість, що визначає предмет, до якого ця тема належить. Таке доповнення є важливим для забезпечення фільтрації персоналізованого рейтингу за темами в розрізі навчальних предметів, що значно покращить аналітичні можливості для викладачів.

Для реалізації гнучкого зв'язку, що дозволяє одному завданню або питанню мати багато тем, і навпаки, будуть створені допоміжні сутності. Це будуть TopicsOnTasks та TopicsOnTestsQuestions. Ці проміжні сутності реалізують відношення типу "багато до багатьох" між сутністю Topic та відповідними сутностями Task і TestQuestion.

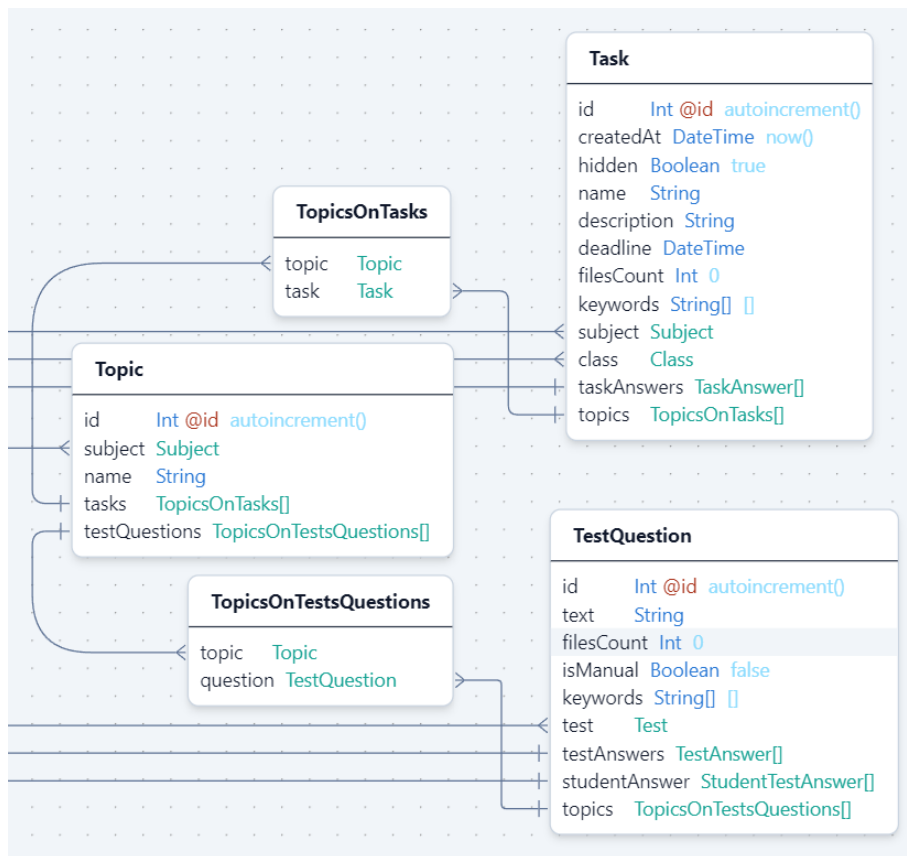


Рис. 3.5. Внесені зміни до моделі даних

Інтеграція системи автоматичного тегування в освітню платформу реалізована на рівні бекенд-контролерів, що забезпечує обробку даних під час створення навчального контенту. Коли викладач ініціює створення нового завдання або питання для тесту, система перехоплює текстовий вміст (опис завдання або текст питання) та виконує його попередню обробку, зокрема очищення від HTML-тегів. Очищений текст разом із внутрішньою назвою навчального предмету передається до зовнішнього мікросервісу через утиліту `taggingServiceCaller`, яка повертає набір релевантних ключових слів та тем.

Отримані дані зберігаються в базі даних із використанням різних стратегій для ключових слів і тем. Ключові слова записуються безпосередньо у відповідні поля сутностей `Task` або `TestQuestion`, що забезпечує швидкий доступ до них. Натомість обробка тем передбачає перевірку їх існування в базі даних для конкретного предмету: якщо тема вже існує, система використовує її, якщо ні, створює нову сутність `Topic`. Зв'язок між контентом і темами

реалізується через проміжні таблиці TopicsOnTasks та TopicsOnTestsQuestions.

Використання збережених метаданих значно розширює функціональність платформи для кінцевих користувачів, зокрема в аспектах пошуку та аналітики. Функція пошуку завдань модифікована таким чином, що запит користувача порівнюється не лише з назвою завдання, але й з масивом збережених ключових слів, що дозволяє знаходити релевантний контент навіть за відсутності прямих входжень у заголовок.

Крім того, система використовує сформовані зв'язки з темами для побудови детальної аналітики успішності студентів. Спеціалізований контролер TopicController агрегує результати виконання завдань та тестів, співставляючи отримані оцінки з темами, до яких належать відповідні питання чи завдання. Алгоритм розраховує середній рейтинг студента по кожній темі, враховуючи як оцінки за практичні завдання, так і правильність відповідей у тестах, що дозволяє формувати персоналізовану статистику компетентностей.

3.2 Структура інтерфейсу веб-додатку та порядок роботи

При зверненні до ресурсу через веб-браузер, користувач потрапляє на вітальну сторінку, яка ініціює процес входу до системи. Головну сторінку зображено на рис. 3.6.

Вітаємо!

Щоб продовжити роботу, будь-ласка увійдіть.

Вхід

Рис. 3.6 Головна сторінка додатку

Перехід до сторінки аутентифікації відкриває форму входу з інтегрованою валідацією даних (рис. 3.7), де необхідно ввести облікові дані

(email та пароль). Реєстрація нових облікових записів здійснюється адміністраторами системи, тому можливість самостійної реєстрації для користувачів відсутня.

Вхід

Введіть ваш email та пароль нижче, для входу.

Email

exapmle

Невірний формат email

Пароль

.....

Пароль повинен містити мінімум 8 символів

Увійти

Рис. 3.7 Форма входу

Після успішної авторизації користувач перенаправляється на головну робочу панель системи (рис. 3.8), яка слугує точкою доступу до всіх функціональних модулів системи. Інтерфейс цієї сторінки динамічно адаптується відповідно до призначених користувачеві ролей, відображаючи відповідний набір навігаційних елементів. У випадку наявності єдиної ролі, система автоматично перенаправляє користувача до відповідного компоненту.

Оберіть роль

Оберіть до робочої зони якої з ваших ролей ви перейдете

Адмін Учень Вчитель Батько/мати

Рис. 3.8 Компонент вибору сторінки ролі

Панель адміністратора надає доступ до управління трьома ключовими сутностями: Класами, Предметами та Користувачами. Вкладка «Класи» (рис. 3.9) забезпечує функціонал для створення та керування класами. Формування класів є важливим для агрегації аналітичних даних, зокрема для відображення зведеного рейтингу успішності групи за автоматично згенерованими тегами.

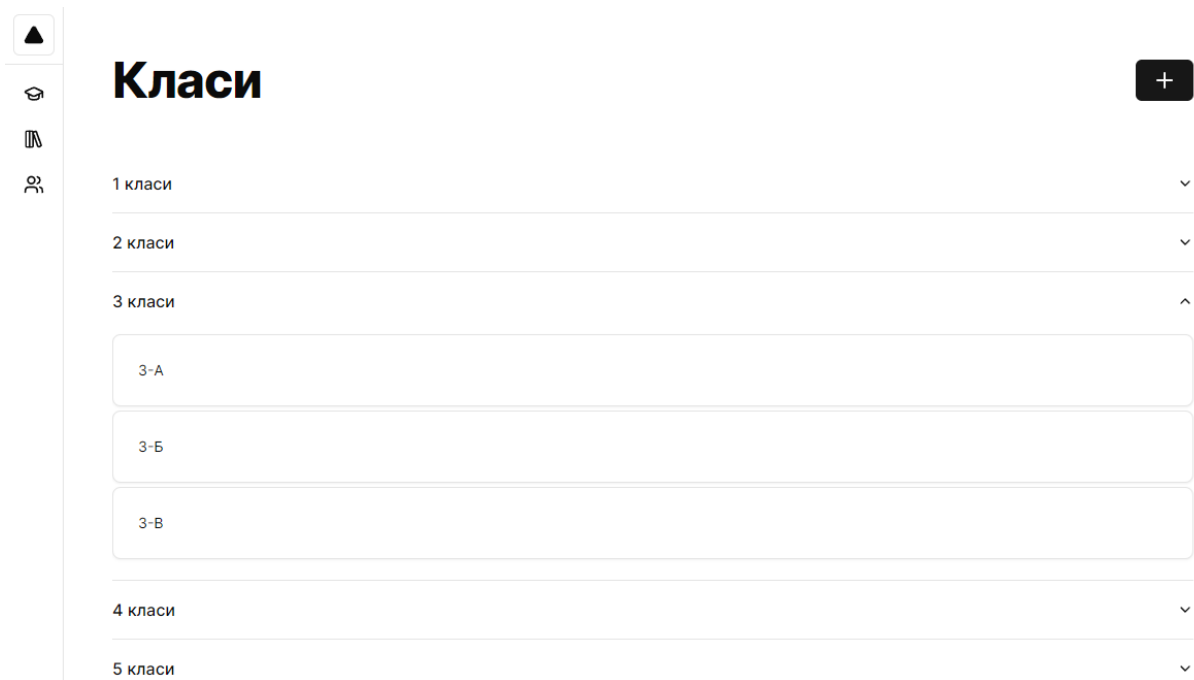


Рис. 3.9 Вкладка класів

Розділ «Предмети» (рис. 3.10) дозволяє переглядати існуючі предмети та додавати нові.

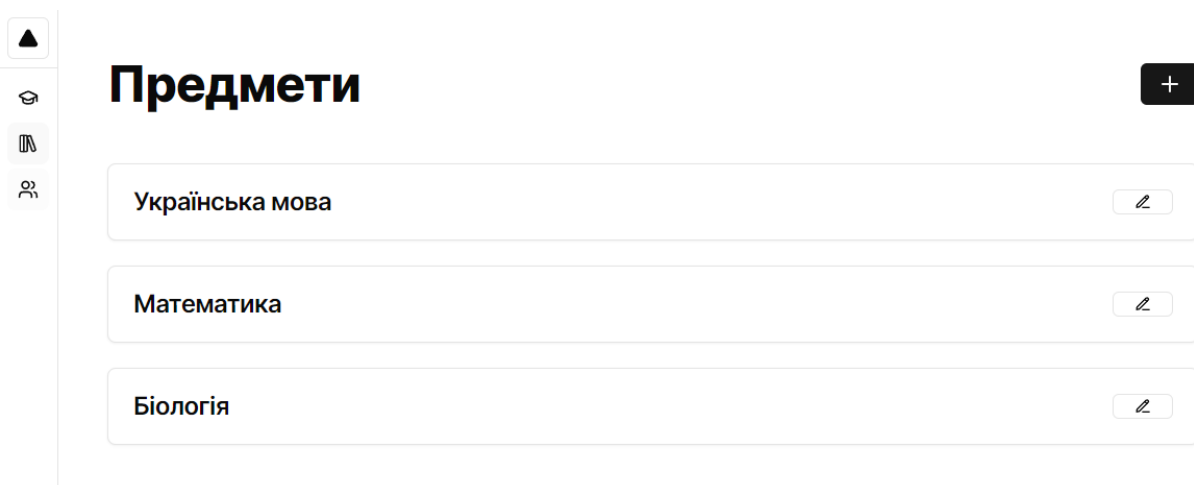


Рис. 3.10 Сторінка предметів

Форма створення предмету (рис. 3.11), окрім стандартної назви, включає спеціалізоване поле для внутрішнього ідентифікатора. Цей ідентифікатор виступає ключем для комунікації з мікросервісом тегування, визначаючи конкретну модель обробки тексту для кожного предмету.

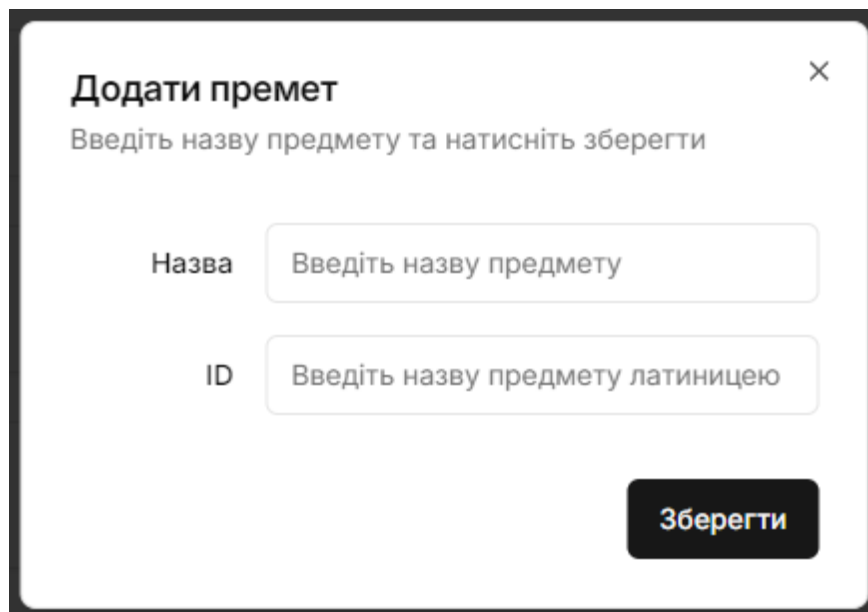


Рис. 3.11 Форма створення предмету

Модуль керування користувачами (рис. 3.12) надає інструменти для адміністрування облікових записів з можливістю фільтрації за ролями та пошуку.

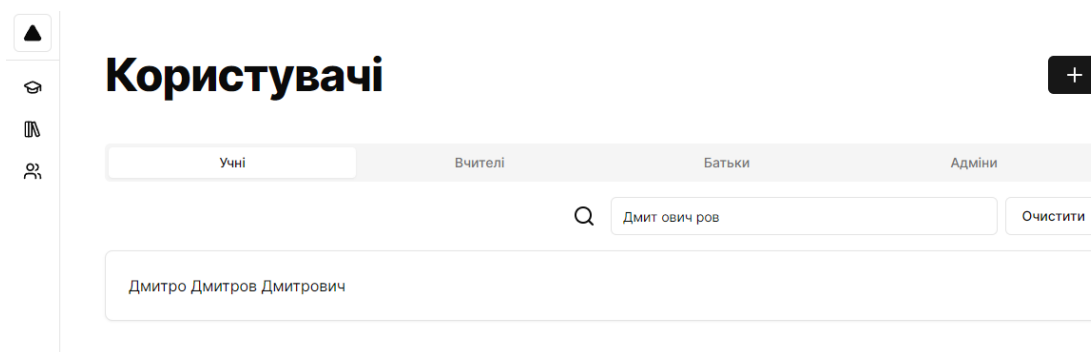


Рис. 3.12 Сторінка керування користувачами

Процес створення нового користувача (рис. 3.13) передбачає заповнення форми та призначення однієї або кількох ролей. Особлива увага приділяється ролям «Учень» та «Вчитель», оскільки їхня взаємодія генерує основний потік

даних для системи автоматичного тегування: вчителі створюють контент, ініціюючи аналіз тексту, а учні формують статистичну базу рейтингів через виконання завдань.

Додати користувача

Введіть дані користувача та натисніть зберегти

Ім'я

Іван

Прізвище

Іванов

По-батькові

Іванович

Ролі

Оберіть які ролі матиме користувач

☒ Учень

☐ Вчитель

☐ Батьки

☐ Адміністратор

Клас

Оберіть до якого класу буде відноситись учень

Обраний клас: 3-А

Батько/Мати

Оберіть користувача, що буде вважатись батьком/мати учня та зможе переглядати його успішність

Обраний користувач: Name Surname Patronymic

Зберегти

Рис. 3.13 Форма створення користувачами

Для завершення активації акаунту адміністратор генерує унікальне посилання, перейшовши за яким користувач має змогу зареєструватись та встановити персональні дані для входу.

Робоче середовище вчителя зосереджене на створенні навчального контенту та моніторингу успішності. Сторінка «Завдання» (рис. 3.14) відображає повний список завдань з можливостями пошуку, який індексує не лише заголовки, але й автоматично присвоєні мікросервісом ключові слова.

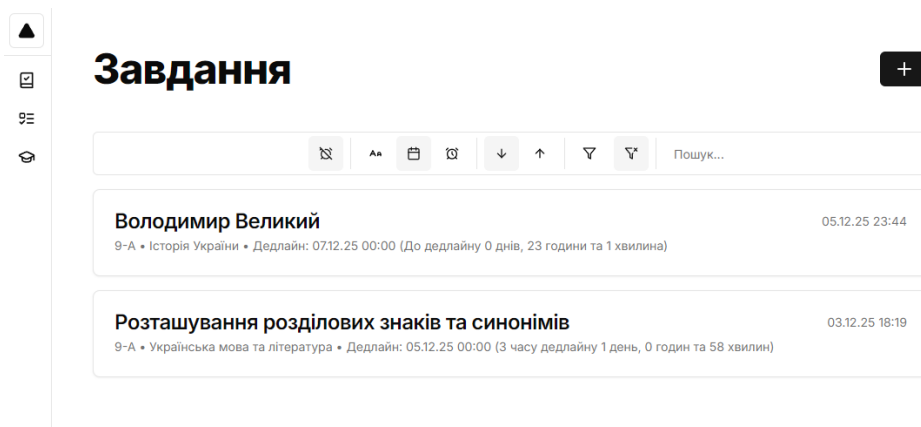


Рис. 3.14 Список завдань

Процес створення завдання (рис. 3.15) передбачає заповнення форми (назва, опис, дедлайн, клас, предмет). Сервер передає ці текстові дані опису завдання до мікросервісу тегування. Отримані результати аналізу (теми та ключові слова) зберігаються в базі даних разом із завданням.

Рис. 3.15 Форма створення завдання

На сторінці перегляду завдання (рис. 3.16) викладач має доступ до згенерованих метаданих, а також інструментів редагування, видалення завдання. Сторінка оцінювання відповіді учня дозволяє переглянути текст відповіді, вказати або змінити оцінку. Внесені оцінки автоматично оновлюють рейтинги успішності учня та класу в розрізі відповідних тем.

▲

📅

☰

🎓

Розташування розділових знаків та синонімів

✎ 🗑 03.12.25 18:19

9-А • Українська мова та література • Дедлайн: 05.12.25 00:00 (3 часу дедлайну 1 день, 1 година та 49 хвилин)

Приховане: ☐

Лексика та фразеологія

Синтаксис та пунктуація

#визначати #синонім #означати

Чи правильно виставлені розділові знаки? Також напишіть що означають фразеологізми, визначіть значення слів та підберіть до них синоніми та антоніми.

Зображення

+

Відповіді учнів:

Василь Іваненко Іванович

Оцінка: 10/12

03.12.25 20:25

Дмитро Петренко Олегович

Оцінка: 4/12

06.12.25 00:12

Рис. 3.16 Сторінка завдання

Аналогічний підхід застосовано до модуля тестування. Сторінка «Тести» відображає повний список тестів та містить форму створення. На сторінці тесту, вчитель отримує доступ до редактору питань (рис. 3.17), що дозволяє переглянути всі запитання тесту, додати нові та переглянути результати проходження тесту учнями. При створенні, кожне запитання, схожим до завдання чином, проходить процес автоматичного тегування, в результаті якого воно отримує теми та ключові слова, які можна побачити в секції питання. Відповіді учнів на ці питання безпосередньо впливають на їхній тематичний рейтинг.

54

▲

☑

☰

🎓

Розділові знаки та фразеологія

✎

🗑

03.12.25 19:48

9-А • Українська мова та література • Ліміт часу: 30 хв • Дедлайн: 05.12.25 00:00 (3 часу дедлайну 1 день, 1 година та 20 хвилин)

Приховане: ☐

Питання

✎

🗑

Лексика та фразеологія

#фразеологізм #пояснити #даний #означати #важливий

Поясніть що означають фразеологізми у даному реченні. Ці фразеологізми є важливими.

Варіанти відповідей:

true

🗑

false

🗑

Новий варіант відповіді...

☑ ☐

+

Зображення

+

Список питань

1

+

Результати учнів:

Василь Іваненко Іванович

01.01.70 03:00

Результат: 0.00% (0)

👤

🔗

⚙

Рис. 3.17 Сторінка тесту та редактор питань

Останнім сегментом функціоналу вчителя є сторінка «Класи». Загальна сторінка класів є ідентичною до сторінки класів Адміністратора. Детальна сторінка класу (рис. 3.18) візуалізує агрегований рейтинг за темами з можливістю фільтрації по предметах. Рейтинг представлений у відсотковому форматі та відображає кожну задіяну тему, кількість результатів виконання завдання чи відповіді на запитання тесту за цією темою та шкалу рейтингу. Інтерфейс використовує колірну індикацію для швидкої ідентифікації проблемних зон або високих досягнень.

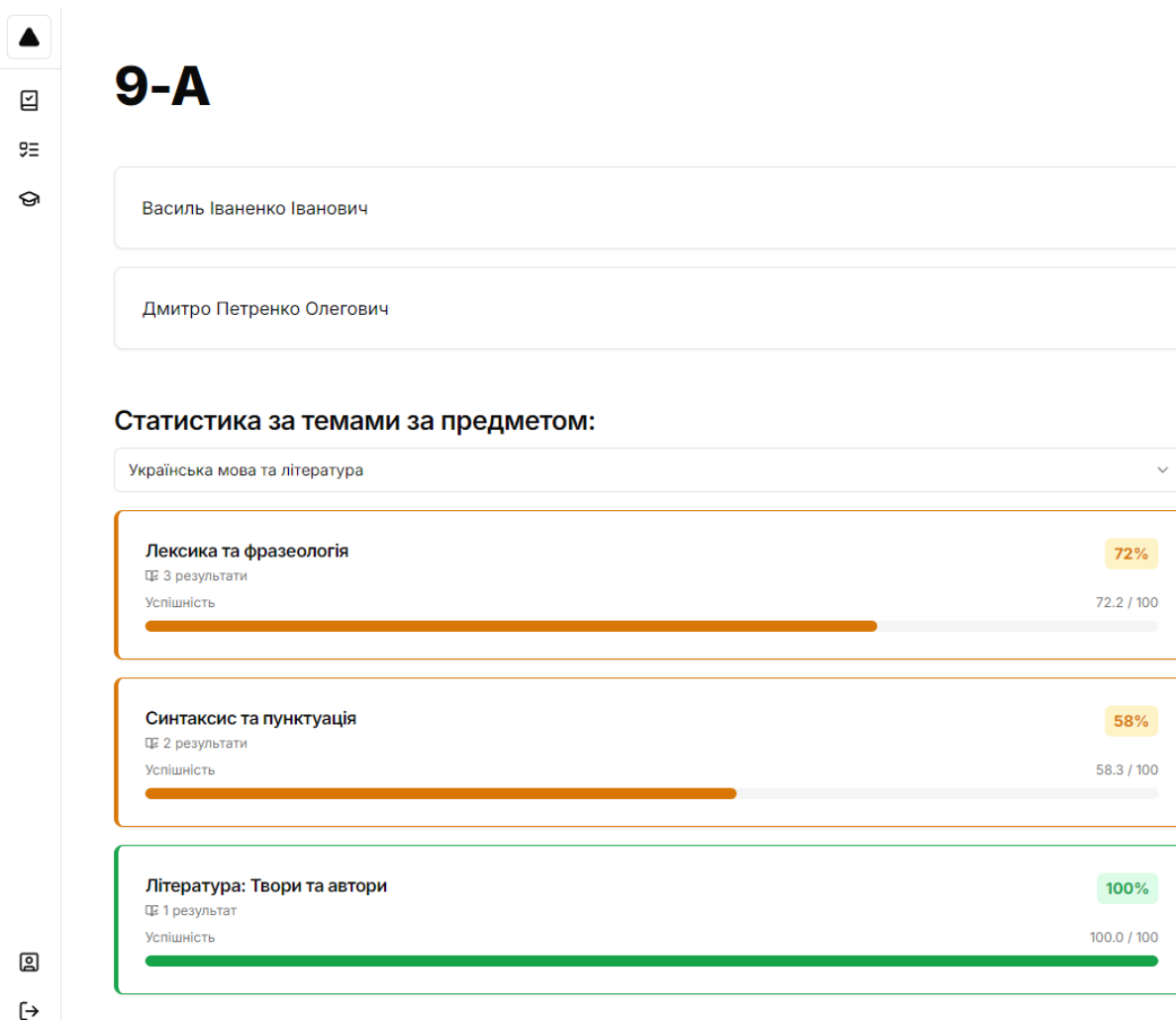


Рис. 3.18 Сторінка класу з рейтингом

Наступною частиною додатку є перелік сторінок, пов'язаних з діяльністю учня. Інтерфейс учня оптимізовано для виконання навчальних активностей та самоконтролю.

Доступ до завдань та тестів організовано аналогічно до викладацького інтерфейсу, проте обмежено режимом перегляду та виконання. Перейшовши на сторінку конкретного завдання, учень бачить присвоєні теми та ключові слова, що сприяє кращому розумінню контексту. Для виконання завдання, учень має відправити текст відповіді. Після перевірки роботи вчителем, сторінка завдання (рис. 3.19) відображає отриману оцінку та коментарі. Отримана оцінка за завдання автоматично враховується в індивідуальний рейтинг учня.

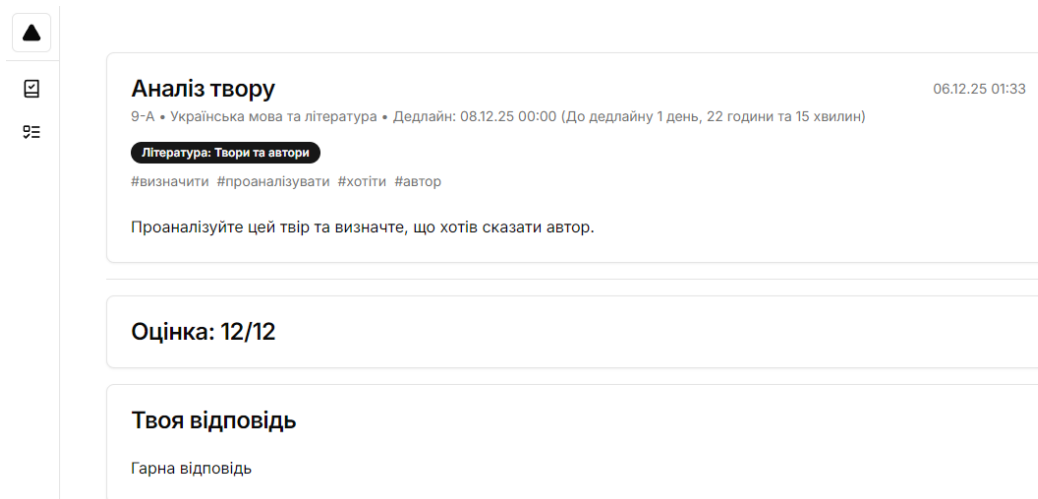


Рис. 3.19 Сторінка завдання та відповіді на нього

Проходження тестів реалізовано в інтерактивному режимі з контролем часу. Результати відповідей на окремі питання автоматично інтегруються в систему рейтингування, оновлюючи показники компетентності за відповідними темами.

Персональний профіль учня (рис. 3.20) містить аналітичну панель, що дублює структуру рейтингу класу, але відображає виключно особисті досягнення. Це дозволяє учневі відстежувати власний прогрес у розрізі конкретних тем та предметів.

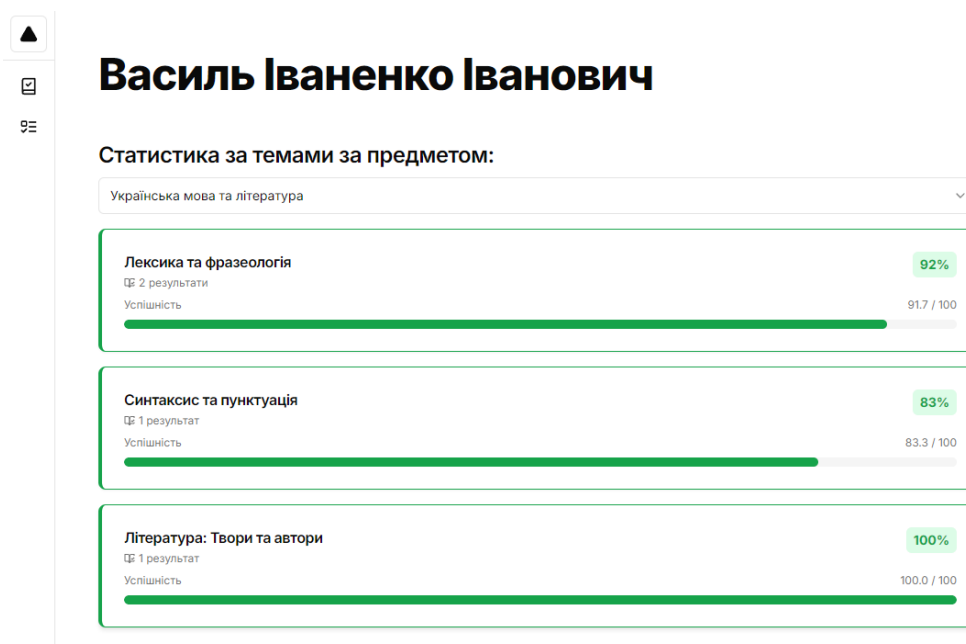


Рис. 3.20 Сторінка завдання та відповіді на нього

3.3 Аналіз результатів роботи системи автоматичного тегування тексту

Для оцінки ефективності розробленого мікросервісу та алгоритмів класифікації було проведено серію тестувань на довільних даних, що не є частиною датасету. Тестування охоплювало п'ять основних предметів: українську мову та літературу, історію України, математику, фізику та хімію. Загальний обсяг навчальної та тестової вибірки суттєво варіювався залежно від предмету, що дозволило також дослідити вплив розміру датасету на якість навчання нейронної мережі. Найбільші обсяги даних були зібрані для гуманітарних дисциплін, зокрема для української мови було оброблено 1921 записи, а для історії України – 1578 записів. Технічні дисципліни мали меншу репрезентативність: хімія – 602 записи, фізика – 357 записів, а математика – 284 записи. Оцінка якості класифікації проводилася за допомогою стандартних метрик машинного навчання: точності (precision), повноти (recall) та F1-міри (середнього гармонійного між точністю та повнотою) [38].

Аналіз отриманих в процесі навчання моделей звітів класифікації демонструє чітку кореляцію між текстовою насиченістю предмету та успішністю роботи моделі. Найвищі показники ефективності були зафіксовані для предмету «Українська мова та література». Зважена середня F1-міра склала 0.90, що є відмінним показником для систем автоматичної обробки тексту. Особливо високу точність модель продемонструвала у категорії «Стилістика та текст», досягнувши F1-міри на рівні 0.97, та у категорії «Фонетика та правопис» з показником 0.94. Це пояснюється тим, що завдання з мови базуються на чітких лінгвістичних правилах та містять специфічну термінологію (наприклад, «звук», «наголос», «склад»), яку векторизатору та LSTM-мережі легко ідентифікувати та розмежувати.

Високі результати також було отримано для предмету «Історія України», де зважена F1-міра склала 0.75. Найкраще модель розпізнає теми, пов'язані з

конкретними подіями та персонами, наприклад, категорія «Революція та визвольні змагання (1917-1921)» отримала F1-міру 0.90. Водночас, категорія «Незалежна Україна» показала нижчий результат (0.61), що може бути викликано перетином сучасної політичної лексики з іншими історичними періодами або недостатньою кількістю унікальних ключових слів у навчальній вибірці для цього періоду.

Результати тестування точних наук показують певну неоднорідність. Для предмету «Хімія» загальна F1-міра склала 0.81, що свідчить про високу здатність системи до узагальнення навіть при середньому розмірі датасету. Категорія «Неорганічна хімія» була класифікована з точністю 0.89, що пояснюється наявністю унікальних назв хімічних елементів та сполук, які діють як сильні маркерні слова. Натомість результати з фізики виявилися помірними зі зваженою F1-мірою 0.66. Аналіз звіту показує значний дисбаланс у розпізнаванні категорій: якщо «Електродинаміка» має високий показник 0.86, то «Квантова фізика» продемонструвала критично низький результат 0.12. Це безпосередньо пов'язано з малим обсягом даних (лише 357 записів загалом) та недостатньою кількістю прикладів саме для квантової фізики у тестовій вибірці, що не дозволило моделі сформувати стійкі нейронні зв'язки для цієї теми [39].

Найскладнішим для автоматичного тегування виявився предмет «Математика», де було отримано незадовільний результат зі зваженою F1-мірою 0.04. Такий показник зумовлений двома факторами. По-перше, математичний датасет був найменшим (284 записи). По-друге, математичні завдання часто формулюються мовою символів та формул, а не природною мовою. LSTM-мережа та TF-IDF векторизатор орієнтовані на семантичний аналіз слів. Коли завдання виглядає як абстрактне рівняння без текстового опису, система не знаходить достатньої кількості лексем для класифікації. Це підтверджує звіт, де більшість категорій, таких як «Стереометрія» чи

«Планіметрія», мають нульові показники точності через неможливість виявити контекст.

Попри статистичні показники, якісний аналіз роботи системи на окремих прикладах демонструє її практичну придатність за умови наявності текстового опису в завданні. Для завдання з української мови та літератури «Визначіть наголоси слів, та їх правильне написання. Напишіть значення слів та визначте синоніми, антоніми, омоніми, пароніми, тлумачення, фразеологізми що фігурують у вислові» система коректно визначила теми «Лексика та фразеологія» та «Фонетика та правопис». Для завдання з історії України «Хто такий Володимир Великий, та що він зробив для Київської Русі» також було коректно визначені теми «Стародавня історія та Русь». Ці приклади підтверджують ефективність моделі при роботі з предметами з високою текстовою насиченістю.

При обробці завдань з хімії система також сформувала коректні теми: «Органічна хімія» для завдання «Опишіть якості етанолу, етилу та безену» та «Неорганічна хімія» з «Хімічні обчислення» для завдання «Обчисліть молярну масу електронів у молекулі заліза». В свою чергу, обробка завдань з фізики показала неідеальні результати. Після обробки завдання з фізики «Визначіть швидкість руху тіла, та його прискорення» коректно визначила тему «Механіка» проте невірно присвоїла тему «Електродинаміка». При тестування роботи системи з завданнями математики, не змогла сформувати коректні результати.

Проведене тестування підтвердило гіпотезу про те, що запропонована архітектура найкраще підходить для предметів з високою щільністю текстової інформації. Система демонструє високу точність та надійність для гуманітарних та природничих наук. Для ефективної роботи з математичними дисциплінами необхідно або суттєво збільшити навчальну вибірку, збагативши її текстовими описами задач, або інтегрувати додаткові методи

попередньої обробки [40], здатні трансформувати математичні формули у текстові токени.

Висновки до розділу 3

У даному розділі було виконано програмну інтеграцію розробленого мікросервісу автоматичного тегування в загальну архітектуру освітньої платформи, організувавши взаємодію через RESTful API з використанням асинхронних запитів та валідації даних. Було модифіковано модель бази даних та розроблено відповідні користувацькі інтерфейси для адміністраторів, викладачів та учнів, що забезпечило можливість автоматичної генерації метаданих при створенні контенту та формування персоналізованих рейтингів компетентності. Також було проведено комплексне тестування системи на навчальних матеріалах з п'яти дисциплін, за результатами якого проаналізовано метрики точності та повноти класифікації. Це дозволило підтвердити високу ефективність обраного підходу для текстомістких дисциплін, таких як українська мова та історія, та виявити необхідність додаткової адаптації алгоритмів для роботи з високоформалізованими математичними завданнями.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було успішно вирішено актуальну науково-прикладну задачу розробки та інтеграції інтелектуальної системи автоматичного тегування текстів для освітньої платформи. Основною метою дослідження стало створення інструментарію, який дозволяє автоматизувати процес класифікації навчального контенту та перейти від загального оцінювання успішності учнів до деталізованого компетентнісного аналізу в розрізі конкретних тем.

У результаті проведеного аналізу предметної області та існуючих аналогів, включаючи хмарні NLP-сервіси та великі мовні моделі, було виявлено їхні недоліки в контексті шкільної освіти, зокрема високу вартість, залежність від зовнішніх провайдерів та недостатню гнучкість у роботі зі специфічною термінологією. Це обґрунтувало необхідність розробки власного рішення, яке базується на гібридному підході до обробки природної мови. Запропонована та реалізована архітектура поєднує алгоритм TF-IDF для виділення ключових слів та рекурентну нейронну мережу Bi-LSTM з механізмом уваги для глибокого семантичного аналізу та визначення тем. Використання попередньо навчених векторних представлень GloVe та лематизації дозволило моделі ефективно враховувати морфологічні особливості української мови та контекстні залежності у текстах завдань.

Практична складова роботи полягає у створенні автономного мікросервісу на базі фреймворку FastAPI, який забезпечує ізоляцію ресурсоемних обчислень машинного навчання від основної логіки платформи. Для навчання системи було сформовано унікальний датасет на основі матеріалів зовнішнього незалежного оцінювання, що пройшов етапи скрейпінгу, очищення та евристичної розмітки. Інтеграція розробленого сервісу в платформу виконання завдань та тестів була реалізована через RESTful API, що супроводжувалося модифікацією бази даних для підтримки

зв'язків «багато до багатьох» між завданнями та темами, а також оновленням інтерфейсів користувача для відображення персоналізованої аналітики.

Експериментальна перевірка системи підтвердила високу ефективність обраного підходу для дисциплін із значним обсягом текстового контенту. Зокрема, для предмету «Українська мова та література» досягнуто показника F1-міри на рівні 0.90, а для «Історії України» – 0.75, що свідчить про високу точність класифікації. Водночас тестування виявило обмеження методу при роботі з високоформалізованими даними, такими як математичні завдання, де переважають формули, що потребує подальших досліджень у напрямку попередньої обробки математичної нотації. Впровадження розробленої системи дозволяє значно зменшити навантаження на викладачів за рахунок автоматизації рутинних процесів та підвищити якість освітнього процесу завдяки наданню детального зворотного зв'язку учням щодо їхніх прогалин у знаннях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Novichkov Y.M., Vakaliuk T.A. Comparison of node.js, deno and bun javascript runtimes. Сучасні інформаційні системи та технології: матеріали VII Всеукр. наук.-практ. інтернет-конф. за тематикою «Сучасні комп'ютерні системи та мережі в управлінні» (29 листопада 2024 р., м. Херсон, м. Хмельницький) / за ред. А. А. Григорової. – Херсон: Книжкове видавництво ФОП Вишемирський В. С., 2024. С. 12-14.
2. Novichkov Y.M., Talaver O.V., Vakaliuk T.A. Text tagging using natural language processing algorithms. Матеріали XVIII Всеукраїнської науково практичної WEB конференції аспірантів, студентів та молодих вчених (25-27 березня 2025 р.). – Кривий Ріг: Криворізький національний університет, 2025. С. 154-156.
3. Новічков Є.М., Чижмотря О.В. Використання bi-lstm моделі з механізмом уваги для автоматичного тегування тексту. Тези VIII Всеукраїнської науково-технічної конференції "Комп'ютерні технології: інновації, проблеми, рішення", 02-03 грудня 2025 року. Житомир: "Житомирська політехніка", 2025. (подано до друку).
4. Minaee, S., Kalchbrenner, N., Cambria, E., Nikzad, N., Chenaghlu, M., & Gao, J. Deep Learning Based Text Classification: A Comprehensive Review. ACM Computing Surveys (CSUR). 2021. T. 54. C. 1–40. URL: <https://doi.org/10.1145/3439726>.
5. Chen, X., Xie, H., Zou, D., & Hwang, G.-J. Application and Theory Gaps during the Rise of Artificial Intelligence in Education.. Computers and Education Artificial Intelligence. 2020. T. 1. URL: <https://doi.org/10.1016/j.caeai.2020.100002>.
6. D. W. Otter, J. R. Medina And J. K. Kalita. A Survey of the Usages of Deep Learning for Natural Language Processing. IEEE Transactions on Neural

Networks and Learning Systems. 2021. T. 32, C. 604–624. URL: <https://doi.org/10.1109/TNNLS.2020.2979670>.

7. Cloud Natural Language documentation. Google Cloud. 2024. URL: <https://docs.cloud.google.com/natural-language/docs> (дата звернення: 15.11.2025).

8. OpenAI, Josh A., Steven A. GPT-4 Technical Report. 2024. URL: <https://doi.org/10.48550/arXiv.2303.08774>.

9. Kasneci, E., Sessler, K., Küchemann, S., Bannert, M., Fischer, F. ChatGPT for Good? On Opportunities and Challenges of Large Language Models for Education. Learning and Individual Differences. 2023. T. 103, 102274. URL: <https://doi.org/10.1016/j.lindif.2023.102274>.

10. Xu, Y., Zhu, Y., Wang, J., & Cheng, Y. Build your own NLP system or use a cloud service? A cost-benefit analysis. Journal of Systems and Software. 2021. T. 174, 110892.

11. Jurafsky, D., & Martin, J. H. Speech and Language Processing (3rd ed. Draft). Stanford University, 2025.

12. Tunstall, L. , Von Werra, L. , Wolf, T. Natural Language Processing with Transformers, Revised Edition. O'Reilly Media, Inc., 2022.

13. Alexander R., Andreas K. On the vanishing and exploding gradient problem in Gated Recurrent Units. IFAC-PapersOnLine. 2020. T. 53. C. 1243–1248. URL: <https://doi.org/10.1016/j.ifacol.2020.12.1342>.

14. Gillioz, A., Casas, J., Mugellini, E., & Khaled, O. A. Overview of the Transformer-based Models for NLP Tasks. Federated Conference on Computer Science and Information Systems. 2020. C. 179–186. URL: <https://doi.org/10.15439/2020F20>.

15. Arfin S., Kalmer A., Nariswari R., Yudistra G. Long Short-Term Memory (LSTM): Trends and Future Research Potential. International Journal of Emerging Technology and Advanced Engineering. 2023. T. 13. C. 24–35. URL: https://doi.org/10.46338/ijetae0523_04.

16. Sazzed, S. Bi-LSTM-based approach for detecting toxic comments in texts. Proceedings of the 2021 IEEE International Conference on Big Data. 2021.
17. Labach, A., Salehinejad, H., & Valaee, S. Survey of Dropout Methods for Deep Neural Networks. 2019. URL: <https://doi.org/10.48550/arXiv.1904.13310>.
18. 3. Yu, Y., Si, X., Hu, C., & Zhang, J. Neural computation. A review of recurrent neural networks: LSTM cells and network architectures. 2019. T. 31. C. 1235–1270. URL: https://doi.org/10.1162/neco_a_01199.
19. Raschka, S., Mirjalili, V. Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2 (3rd ed.). Packt Publishing, 2019.
20. Chollet, F. Deep Learning with Python (2nd ed.). Manning Publications. 2021.
21. Brauwiers, G., & Frasincar, F. A General Survey on Attention Mechanisms in Deep Learning. IEEE Transactions on Knowledge and Data Engineering. 2023. C. 99. URL: <https://doi.org/10.1109/TKDE.2021.3126456>.
22. Xu, J., Sun, X., Zhang, Z., Zhao, G., & Lin, J. Understanding and improving layer normalization. Advances in Neural Information Processing Systems. 2019. URL: <https://doi.org/10.48550/arXiv.1911.07013>.
23. Almeida, F., & Xexéo, G. Word Embeddings: A Survey. Federal University of Rio de Janeiro. 2019. (Preprint. Federal University of Rio de Janeiro) URL: <https://doi.org/10.48550/arXiv.1901.09069>.
24. Available trained pipelines for Ukrainian. Spacy. URL: <https://spacy.io/models/uk> (дата звернення: 18.11.2025).
25. TfidfVectorizer. Scikit-learn documentation. URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html#sklearn.feature_extraction.text.TfidfVectorizer (дата звернення: 17.11.2025).
26. Voron F. Building Data Science Applications with FastAPI. Packt Publishing. 2023.

27. Data validation using Python type hints. Pydantic Documentation. URL: <https://docs.pydantic.dev/> (дата звернення: 17.11.2025).
28. Molin S. Hands-On Data Analysis with Pandas. Packt Publishing. 2021.
29. Sarkar D. Text Analytics with Python: A Practitioner's Guide to Natural Language Processing. Apress, 2019. URL: <https://doi.org/10.1007/978-1-4842-4354-1>.
30. Zhuang, J., Tang, T., Ding, Y., Tatikonda, S., & Li, X. AdaBelief optimizer: Adapting stepsizes by the belief in observed gradients. Advances in neural information processing systems. 2020. T. 33. C. 18795–18806.
31. Richardson, C. Microservices patterns: with examples in Java. Manning Publications. 2020.
32. Masse, M. REST API Design Rulebook. O'Reilly Media. 2021.
33. Percival, H., & Gregory, B. Architecture Patterns with Python. O'Reilly Media. 2020.
34. Wargnier, L. Node.js Design Patterns (3rd ed.). Packt Publishing. 2021.
35. Schöning, H. Mastering PostgreSQL 13. Packt Publishing. 2020.
36. Prisma Documentation: Next-generation ORM for Node.js. Prisma. URL: <https://www.prisma.io/docs> (дата звернення: 17.11.2025).
37. Next.js Documentation. Vercel. URL: <https://nextjs.org/docs> (дата звернення: 17.11.2025).
38. Grandini, M., Bagli, E., & Visani, G. Metrics for Multi-Class Classification: an Overview. 2020. URL: <https://doi.org/10.48550/arXiv.2008.05756>.
39. Paullada, A., Raji, I. D., Bender, E. M., Denton, E., & Hanna, A. Data and its (dis) contents: A survey of dataset development and use in machine learning research. Patterns. 2021. T. 2. 100336. URL: <https://doi.org/10.1016/j.patter.2021.100336>.

40. Zhong, Z., Li, H. A comprehensive survey on automatic knowledge graph construction. *ACM Computing Surveys*. 2020. T. 56. C. 1–62. URL: <https://doi.org/10.1145/3618295>.

ДОДАТКИ

Код попередньої обробки даних

```

class TextPreprocessor:

def __init__(self):
    try:
        self.nlp = spacy.load("uk_core_news_sm", disable=["parser", "ner"])
    except OSError:
        print("Downloading language model...")
        from spacy.cli import download

        download("uk_core_news_sm")
        self.nlp = spacy.load("uk_core_news_sm", disable=["parser", "ner"])

def clean_text(self, text: str) → str:
    """Basic cleanup: lowercase and remove numbers."""
    text = text.lower()
    text = re.sub(r"\d+", "", text)
    text = re.sub(f"[{re.escape(string.punctuation)}]", " ", text)
    text = re.sub(r"\s+", " ", text).strip()
    return text

def process(self, text: str) → str:
    """Full pipeline: Clean → Tokenize → Remove Stopwords →
    Lemmatize."""
    cleaned_text = self.clean_text(text)

    doc = self.nlp(cleaned_text)

    lemmatized_words = [
        token.lemma_
        for token in doc
        if not token.is_stop and not token.is_punct and len(token.text) > 1
    ]

    return " ".join(lemmatized_words)

```

Код побудови LSTM моделі

```

def build_lstm_model(
    vocab_size: int,
    embedding_dim: int,
    max_len: int,
    num_classes: int,
    embedding_matrix=None,
    vectorize_layer=None,
) → Model:
    if vectorize_layer is not None:
        inp = Input(shape=(1,), dtype=tf.string)
        x = vectorize_layer(inp)
    else:
        inp = Input(shape=(max_len,))
        x = inp

    if embedding_matrix is not None:
        x = Embedding(
            input_dim=vocab_size,
            output_dim=embedding_dim,
            weights=[embedding_matrix],
            trainable=True,
        )(x)

    x = Bidirectional(LSTM(config.LSTM_UNITS, return_sequences=True))(x)
    x = Dropout(0.4)(x)

    attention = MultiHeadAttention(num_heads=2, key_dim=32)(query=x,
value=x)

    x = LayerNormalization()(x + attention)
    x = GlobalAveragePooling1D()(x)
    x = Dense(64, activation="relu")(x)
    x = Dropout(0.3)(x)

    out = Dense(num_classes, activation="sigmoid")(x)

    model = Model(inputs=inp, outputs=out)
    model.compile(loss="binary_crossentropy", optimizer="adam",
metrics=["accuracy"])

    return model

```

Код тренування моделі

```

def train_subject(subject: str):
    if subject not in config.SUBJECT_DATASETS:
        raise ValueError(f"Subject '{subject}' not found in
config.SUBJECT_DATASETS")

    paths = config.get_subject_paths(subject)
    base_output_dir = os.path.dirname(paths["model"])

    print(f"--- Starting Training for: {subject} ---")

    if not os.path.exists(paths["data"]):
        raise FileNotFoundError(f"Dataset not found: {paths['data']}")

    df = pd.read_csv(paths["data"])

    print("Preprocessing text...")
    preprocessor = TextPreprocessor()
    df["processed_text"] = df["text"].apply(preprocessor.process)

    print("Training TF-IDF on full dataset (including untagged rows)...")
    tfidf = get_tfidf_vectorizer()
    tfidf.fit(df["processed_text"])

    df["tags_list"] = df["tags"].apply(lambda x: [t.strip() for t in
str(x).split(",")])

    print("Filtering rows without tags for LSTM training...")

    def is_valid_tag_list(tags):
        if not tags:
            return False
        if len(tags) == 1 and (tags[0] == "nan" or tags[0] == ""):
            return False
        return True

    df_clean = df[df["tags_list"].apply(is_valid_tag_list)].copy()

    print(f"Original rows: {len(df)}")
    print(f"Rows with valid tags (for LSTM): {len(df_clean)}")
    print(f"Dropped {len(df) - len(df_clean)} rows.")

```

```

mlb = MultiLabelBinarizer()
y = mlb.fit_transform(df_clean["tags_list"])
X = df_clean["processed_text"].values

X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.2, random_state=42
)

print("Adapting TextVectorization layer...")
vectorize_layer = tf.keras.layers.TextVectorization(
    max_tokens=config.MAX_VOCAB_SIZE,
    output_mode="int",
    output_sequence_length=config.MAX_SEQ_LEN,
)

vectorize_layer.adapt(X_train)

vocab = vectorize_layer.get_vocabulary()
embedding_matrix, vocab_size = load_embeddings(
    config.EMBEDDING_PATH, vocab, config.EMBEDDING_DIM
)

callbacks = [
    EarlyStopping(
        monitor="val_loss", patience=5, restore_best_weights=True,
verbose=1
    ),
    ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=2,
verbose=1),
]

print("Building and training model...")
model = build_lstm_model(
    vocab_size=vocab_size,
    embedding_dim=config.EMBEDDING_DIM,
    max_len=config.MAX_SEQ_LEN,
    num_classes=len(mlb.classes_),
    embedding_matrix=embedding_matrix,
    vectorize_layer=vectorize_layer,
)

history = model.fit(
    X_train,

```

```

        y_train,
        batch_size=config.BATCH_SIZE,
        epochs=config.EPOCHS,
        validation_data=(X_val, y_val),
        callbacks=callbacks,
    )

    print(f"Saving artifacts for {subject}...")
    model.save(paths["model"])

    with open(paths["mlb"], "wb") as f:
        pickle.dump(mlb, f)
    with open(paths["tfidf"], "wb") as f:
        pickle.dump(tfidf, f)

    save_training_plots(history, subject, base_output_dir)
    generate_statistics(model, X_val, y_val, mlb, subject, base_output_dir)

    print("Done.")

```

Код виконання передбачення

```

class InferenceService:
    def __init__(self):
        self.preprocessor = TextPreprocessor()
        self.resources = {}

    def load_subject_resources(self, subject: str):
        if subject in self.resources:
            return

        paths = config.get_subject_paths(subject)

        if not os.path.exists(paths["model"]):
            raise FileNotFoundError(f"Models for subject '{subject}' not trained yet.")

        print(f>Loading resources for {subject}...)

        resources = {}
        resources["model"] = load_model(paths["model"])

        with open(paths["mlb"], "rb") as f:
            resources["mlb"] = pickle.load(f)

        with open(paths["tfidf"], "rb") as f:
            resources["tfidf"] = pickle.load(f)

        self.resources[subject] = resources

    def extract_keywords(self, text: str, subject: str) → list:
        tfidf = self.resources[subject]["tfidf"]
        feature_array = np.array(tfidf.get_feature_names_out())

        tfidf_matrix = tfidf.transform([text])
        scores = tfidf_matrix.toarray().flatten()

        relevant_indices = np.where(scores > config.KEYWORD_SCORE_THRESHOLD)

        relevant_words = feature_array[relevant_indices]
        relevant_scores = scores[relevant_indices]

```

```

sorted_order = np.argsort(relevant_scores)[::-1]
sorted_keywords = relevant_words[sorted_order]

return sorted_keywords[: config.TOP_K_KEYWORDS].tolist()

def predict(self, text: str, subject: str):
    self.load_subject_resources(subject)
    res = self.resources[subject]

    processed = self.preprocessor.process(text)

    input_tensor = tf.constant([processed], dtype=tf.string)

    pred_probs = res["model"].predict(input_tensor, verbose=0)[0]

    threshold_indices = np.where(pred_probs >=
config.PREDICTION_THRESHOLD)[0]
    if len(threshold_indices) == 0:
        threshold_indices = [np.argmax(pred_probs)]

    topics = [res["mlb"].classes_[i] for i in threshold_indices]

    keywords = self.extract_keywords(processed, subject)

    return {
        "subject": subject,
        "topics": topics,
        "keywords": keywords,
        "lemmatized_text": processed,
    }

```