

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:

**«Використання засобів машинного навчання для
покращення сервісу для знайомств та пошуку
друзів»**

Виконав студент групи ІПЗм-23-1
КЛЮЄВ Ілля Олександрович

Керівник роботи:
МОРОЗОВ Андрій Васильович

Рецензент:
ЧИЖМОТРЯ Олена Геннадіївна

Житомир – 2024

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
«01» жовтня 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **КЛЮЄВ Ілля Олександрович**
Керівник роботи: **МОРОЗОВ Андрій Васильович**
Тема роботи: **«Використання засобів машинного навчання для покращення сервісу для знайомств та пошуку друзів»**,
затверджена Наказом закладу вищої освіти від **«01» жовтня 2024 р. №520/с**
Вихідні дані для роботи: **функціональні, соціальні та технічні аспекти платформ для знайомств, які впливають на користувацький досвід, алгоритми машинного навчання, нейронні мережі та програмні засоби для класифікації, аналізу та генерації контенту.**
Керівники з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Морозов А.В.	01.10.2024р.	01.10.2024р.
2	Морозов А.В.	01.10.2024р.	01.10.2024р.
3	Морозов А.В.	01.10.2024р.	01.10.2024р.

РЕФЕРАТ

Кваліфікаційна робота освітнього ступеня «магістр» присвячена аналізу проблемних аспектів сервісів для знайомств, виявленню можливостей їх оптимізації та розробці програмних засобів для покращення функціональності платформи. У роботі досліджено застосування методів машинного навчання для автоматизації ключових процесів: оцінки якості фотографій, цензурування фотографій, що порушують правила спільноти, верифікації особистості користувачів та генерації першого повідомлення для полегшення комунікації.

Метою роботи є підвищення ефективності сервісів для знайомств через автоматизацію аналізу контенту, верифікації особистості та оптимізації користувацької взаємодії.

Робота складається зі вступу, трьох розділів та висновків. У першому розділі проведено всебічний аналіз сучасних методів машинного навчання, які можуть бути використані для покращення сервісів знайомств та пошуку друзів. У другому розділі зосереджено увагу на практичній реалізації рішень для покращення платформи. Створено та натреновано класифікатор якості фотографій на основі сучасних архітектур Vision Transformer, розроблено багатокроковий інтерактивний механізм верифікації особистості, створено та натреновано модель нейронної мережі для генерації персоналізованих привітальних повідомлень. Третій розділ присвячений аналізу результатів впровадження запропонованих рішень. Детально оцінено ефективність розроблених моделей, їх інтеграцію в платформу та вплив на покращення користувацького досвіду. Окрему увагу приділено перспективам подальшого вдосконалення сервісу.

Наукова новизна роботи полягає у комплексному підході до вдосконалення платформ для знайомств через впровадження інноваційних методів машинного навчання. Запропоновані рішення враховують специфіку платформ та спрямовані на забезпечення якості контенту, автоматизацію верифікації та оптимізацію комунікації між користувачами.

Практичне значення роботи полягає у створенні готових до інтеграції програмних модулів, які підвищують якість взаємодії користувачів з платформою, полегшують адміністрування та підвищують довіру до сервісу.

Обсяг роботи: пояснювальна записка містить 135 сторінок, включає 109 ілюстрацій, 9 таблиці, 57 посилання на літературні джерела та 3 додатки.

Ключові слова: СЕРВІСИ ЗНАЙОМСТВ, МАШИННЕ НАВЧАННЯ, ТРАНСФОРМЕРИ, НЕЙРОННІ МЕРЕЖІ, ВЕРИФІКАЦІЯ, ГЕНЕРАЦІЯ ТЕКСТУ, GPT, FACEMESH, КЛАСИФІКАТОРИ, MTCNN, VISION TRANSFORMER, ЯКІСТЬ ФОТОГРАФІЙ.

ABSTRACT

The master's qualification thesis is dedicated to analyzing problematic aspects of dating services, identifying opportunities for their optimization, and developing software tools to improve platform functionality. The study explores the application of machine learning methods to automate key processes: assessing photo quality, censoring photos that violate community guidelines, verifying user identities, and generating the first message to facilitate communication.

The aim of the thesis is to enhance the efficiency of dating services by automating content analysis, user verification, and optimizing user interactions.

The thesis consists of an introduction, three chapters, and conclusions. The first chapter provides a comprehensive analysis of modern machine learning methods applicable to improving dating and social networking platforms. The second chapter focuses on the practical implementation of solutions to enhance the platform. A photo quality classifier based on modern Vision Transformer architectures was developed and trained, a multi-step interactive user verification mechanism was designed, and a neural network model for generating personalized greeting messages was created and trained. The third chapter is devoted to analyzing the results of implementing the proposed solutions. It includes a detailed evaluation of the developed models' effectiveness, their integration into the platform, and their impact on improving the user experience. Special attention is given to the prospects for further enhancement of the service.

The scientific novelty of the work lies in a comprehensive approach to improving dating platforms through the introduction of innovative machine learning methods. The proposed solutions take into account platform-specific requirements and are aimed at ensuring content quality, automating verification processes, and optimizing user communication.

The practical significance of the work is reflected in the creation of ready-to-integrate software modules that enhance user interaction with the platform, simplify administration, and increase trust in the service.

Volume of work: The explanatory note contains 135 pages, including 109 illustrations, 9 tables, 57 references and 3 annexes.

Keywords: DATING SERVICES, MACHINE LEARNING, TRANSFORMERS, NEURAL NETWORKS, VERIFICATION, TEXT GENERATION, GPT, FACEMESH, CLASSIFIERS, MTCNN, VISION TRANSFORMER, PHOTO QUALITY.

ЗМІСТ

ВСТУП.....	5
РОЗДІЛ 1. АНАЛІЗ ЗАСОБІВ МАШИННОГО НАВЧАННЯ ДЛЯ ПОКРАЩЕННЯ СЕРВІСУ ДЛЯ ЗНАЙОМСТВ ТА ПОШУКУ ДРУЗІВ.....	8
1.1. Аналіз методів оцінки якості фотографій.....	8
1.2. Аналіз методів автоматизованої верифікації особистості.....	16
1.3. Аналіз методів генерацій першого повідомлення.....	19
1.4. Постановка задачі дослідження.....	23
Висновки до першого розділу.....	26
РОЗДІЛ 2. РЕАЛІЗАЦІЯ ПОКРАЩЕННЯ СЕРВІСУ ДЛЯ ЗНАЙОМСТВ ТА ПОШУКУ ДРУЗІВ ЗА ЗАСОБАМИ МАШИННОГО НАВЧАННЯ.....	27
2.1. Класифікатор якості фотографій.....	27
2.2. Процес верифікації особистості.....	44
2.3. Генератор першого повідомлення.....	74
Висновки до другого розділу.....	88
РОЗДІЛ 3. АНАЛІЗ РЕЗУЛЬТАТІВ ТА ЗАСТОСУВАННЯ ПОКРАЩЕННЯ СЕРВІСУ ДЛЯ ЗНАЙОМСТВ ТА ПОШУКУ ДРУЗІВ ЗАСОБАМИ МАШИННОГО НАВЧАННЯ.....	89
3.1. Оцінка отриманого класифікатора якості фотографій.....	89
3.2. Оцінка створеного процесу верифікації особистості.....	94
3.3. Оцінка отриманого генератора перших повідомлень.....	96
3.4. Застосування інших досліджень.....	110
Висновки до третього розділу.....	116
ВИСНОВКИ.....	118
ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ.....	120
ДОДАТКИ.....	127

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

- CPU — (англ. Central Processing Unit) — Центральний процесор;
- GPU — (англ. Graphic Processing Unit) — Графічний процесор;
- TPU — (англ. Tensor Processing Unit) — Тензорний (нейронний) процесор;
- Інференс — (англ. Inference) — Процес виконання нейронної мережі;
- CNN — (англ. Convolutional Neural Network) — Згорткова нейронна мережа;
- ViT — (англ. Vision Transformer) — Архітектура трансформера для роботи з візуальними даними;
- Loss — (англ. Cross Entropy Loss) — Перехресна ентропія (функція втрат);
- ПЗ — (укр. Програмне Забезпечення);
- SaaS — (англ. Software as a Service) - бізнес-модель, при якій постачальник надає споживачам доступ до ПЗ через Інтернет;
- LLM — (англ. Large Language Model) — Велика мовна модель;
- Landmarks — (англ. Face Landmarks) — Ключові точки на обличчі людини;
- TTS — (англ. Text To Speech) — Технологія перетворення тексту в усне мовлення;
- Датасет — (англ. Data Set) — Набір даних для тренування моделей;
- Промпт — (англ. Prompt) — Запит до мовної моделі, вхідні дані;
- PWA — (англ. Progressive Web Application) — Односторінковий веб-додаток із можливістю встановлення;
- Збиральник — (англ. Bundler) — Програма, що автоматизує процес збирання та компіляції програмного продукту.

ВСТУП

Актуальність теми. У сучасному світі сервіси для знайомств стали невід'ємною частиною соціального життя, надаючи користувачам можливість знаходити партнерів та друзів через зручні онлайн-платформи. Наприклад, річний дохід додатку Bumble, що також об'єднує романтичні та дружні знайомства, за останні 4 роки збільшився більше ніж в два рази[1]. Це свідчить про високий попит на послуги онлайн знайомств та актуальність даного напрямку.

Додаткова актуальність теми зумовлена соціальними викликами сучасності. Через активні військові дії на території України багато людей стали вимушеними переселенцями, які опинилися у складній ситуації потреби адаптації до нового середовища. Для таких осіб платформи для знайомств і пошуку друзів можуть стати дієвим інструментом для створення соціальних зв'язків у нових умовах, дозволяючи знаходити людей із схожими цінностями, інтересами у нових місцях проживання.

Проте, ефективність додатків для знайомств та пошуку друзів обмежується кількома проблемними аспектами, такими як низька якість фотографій, несправжні профілі, складнощі в першій комунікації та недостатня персоналізація. Використання засобів машинного навчання дозволить вирішити ці проблеми шляхом автоматизації процесів та впровадження інтелектуальних підходів для покращення користувацького досвіду.

У цьому контексті впровадження технологій машинного навчання стає вирішальним інструментом для подолання зазначених викликів.

Мета роботи. Метою роботи є аналіз проблемних аспектів функціонування платформ для знайомств та пошуку друзів, виявлення можливостей їх оптимізації та розробка програмних засобів, що покращують якість сервісу за допомогою методів машинного навчання.

Завдання дослідження. Для досягнення мети необхідно виконати такі глобальні завдання:

1. Проаналізувати проблемні місця сервісів для знайомств та пошуку друзів, зокрема наповнення профілів, процеси верифікації особистості, складнощі встановлення та продовження комунікації між користувачами, та інші аспекти, що впливають на ефективність платформи.
2. Виявити та описати можливості для покращення функціональності та користувацького досвіду за допомогою використання сучасних засобів машинного навчання та штучного інтелекту.
3. Розробити програмні засоби для імплементації цих покращень.
4. Ввести розроблені рішення в експлуатацію шляхом інтеграції у реальну платформу для знайомств та пошуку друзів, провести тестування та оцінити результати їх впливу на ефективність сервісу.

Об'єкт дослідження — функціональні, соціальні та технічні аспекти платформ для знайомств, які впливають на користувацький досвід.

Предмет дослідження — алгоритми машинного навчання, нейронні мережі та програмні засоби для класифікації та генерації контенту.

Наукова новизна. У роботі запропоновано інтеграцію комплексних рішень на основі машинного навчання, що охоплюють оцінку та аналіз фотографій, автоматизовану інтерактивну верифікацію особистості, а також контекстуальну генерацію повідомлень з метою покращення досвіду користувачів при використанні платформи.

Практичне значення. Розроблені рішення спрямовані на підвищення ефективності сервісів для знайомств та пошуку друзів через поліпшення взаємодії між користувачами, підвищення якості контенту та автоматизацію ключових процесів, що дозволяє збільшити довіру до платформи та покращити користувацький досвід.

Апробація. Окремі дослідження за темою кваліфікаційної магістерської роботи опубліковано у вигляді чотирьох тез на різних науково-технічних конференціях [2], [3], [4], [5], а також протестовані в реальних умовах інтеграції до платформи знайомств та пошуку друзів.

РОЗДІЛ 1. АНАЛІЗ ЗАСОБІВ МАШИННОГО НАВЧАННЯ ДЛЯ ПОКРАЩЕННЯ СЕРВІСУ ДЛЯ ЗНАЙОМСТВ ТА ПОШУКУ ДРУЗІВ

1.1. Аналіз методів оцінки якості фотографій

У сучасному цифровому світі сервіси для знайомств стали невід'ємною частиною життя багатьох людей. Вони надають можливість швидко і легко знаходити нові знайомства і формувати романтичні та дружні стосунки. Однак, незважаючи на високу популярність, такі платформи стикаються з численними викликами, одним із яких є орієнтовність на фотографії.

Хоча сучасні додатки для знайомств і намагаються підключити до процесу підбору партнера інші фактори, такі як психологічні тести, оцінки сумісності, аналіз спільних інтересів та цінностей; все одно одним із найвагоміших факторів, що впливають на рішення відправити симпатію людині, залишаються фотографії анкети.

Фотографії відіграють вирішальну роль у формуванні першого враження про користувача. Дослідження показують, що неякісні зображення можуть суттєво знижувати інтерес до профілю, що, в свою чергу, впливає на загальну ефективність сервісу. Незважаючи на важливість цього аспекту, значна кількість користувачів завантажує фотографії, які не відповідають базовим критеріям якості, що суттєво знижує шанси їх анкет на успіх.

До таких фотографій належать:

1. Розмиті зображення, на яких обличчя або інші важливі деталі не видно через погане фокусування;
2. Фотографії з поганим освітленням, що робить обличчя користувача невидимим або непривабливим;
3. Неправильні ракурси: фотографії, зроблені з невдалих кутів, що спотворюють зовнішність користувача;

4. Зображення низької роздільної здатності, де обличчя складається з окремих пікселів;
5. Недоцільні сторонні фотографії, такі як зображення природи, меми, кадри із фільмів, або ж просто картинки з інтернету;
6. Непрофесійний контекст: фотографії, зроблені в невідповідних обставинах, наприклад, у брудних або захаращених приміщеннях;
7. Фотографії, що не показують користувача належним чином, такі як знімки зі спини, фотографії, де обличчя займає весь кадр, фото у темних окулярах або у дзеркало, де телефон закриває значну частину обличчя, тощо;

Неякісні фотографії призводять до того, що інші користувачі менше зацікавлені в перегляді таких анкет. Це значно знижує кількість взаємних контактів та може створювати негативне враження про сам сервіс.

Також, багато користувачів не усвідомлюють, що їхні проблеми з відсутністю симпатій можуть бути пов'язані саме з якістю фотографій. Вони можуть вважати, що це сервіс не працює належним чином або не підходить для них, що сприяє зниженню довіри до платформи та відтоку користувачів.

Вирішити цю проблему може створення класифікатора, який би автоматично аналізував фотографії, що завантажуються користувачами. Такий класифікатор зможе виявляти неякісні та недоцільні фотографії, і підсвічувати їх для користувачів. Крім того, можна продемонструвати користувачу невеличку інструкцію з прикладами, як правильно робити фотографії для максимальної ефективності анкети, завдяки чому загальна якість фотографій у сервісі зросте, що покращить враження користувачів від платформи.

1.1.1. Критерії якості фотографії

Для початку, сформуємо критерії якості на основі проблем, описаних раніше. Ці критерії будуть використовуватися у майбутньому для оцінки якості навчання моделі та її можливого подальшого ручного до-навчання.

Фотографію можна вважати якісною, якщо на ній зображена лише одна людина (передбачається, що це власник анкети), без сторонніх або групових зображень.

Освітлення повинно бути таким, щоб було добре видно обличчя та фігуру людини, при цьому, перевага надається яскравим фотографіям над темними.

На фото не повинно бути артефактів, таких як стікери, snapchat-маски, шум або помітні фільтри. Додаткові рамки, колажі або інші елементи, що спотворюють вигляд зображення, також мають бути відсутні.

Фотографія повинна бути правильно орієнтованою, тобто не повернутою. Ракурс і масштаб мають бути адекватними, фотографії сильно зверху або знизу вважаються неякісними. Також, це стосується випадків де обличчя або окремі частини тіла займають увесь розмір зображення.

Якщо, фотографія не підпадає під усі вищевказані критерії, її потрібно вважати за неякісну. Теоретично, можуть допускатися невеликі відхилення для стилістичних фотографій, проте, їх кількість повинна бути мінімізована.

1.1.2. Проблематика непристойних фотографій

Окрім проблеми з неякісними фотографіями, деякі користувачі можуть завантажувати фотографії непристойного виду, що здатне викликати дискомфорт та негативний досвід у інших користувачів системи. У зв'язку з цим, на посади модераторів фотографій у популярних додатках для знайомств залучається велика кількість людей, що призводить до значних витрат, адже кількість фотографій може сягати десятків мільйонів. Проте, навіть за цих умов

не завжди вдається забезпечити ефективне виявлення та вилучення непристойного контенту.

У цьому контексті використання засобів машинного навчання стає дуже перспективним напрямком, адже сучасні нейронні мережі можуть автоматизувати процес виявлення непристойних фотографій та спростити його для адміністраторів сервісів для знайомств. Впровадження таких систем не лише допоможе зберегти безпеку та комфорт користувачів, але й призведе до значної економії часу та ресурсів, які витрачаються на ручну модерацію.

У кваліфікаційній роботі освітнього ступеня «бакалавр»[6] в розділі 2.4.3, вже було описано застосування моделі для розпізнавання непристойного контенту у фотографіях. Тоді для цього використовувалася модель *suko/nsfw*[7], що побудована на основі технології Microsoft Lobe. Проте, під час практичного застосування даної моделі було помічено доволі низьку точність, адже модель натренована лише на 1000 фотографіях[7]. Це призвело до великої кількості хибно-позитивних та хибно-негативних результатів.

Однак, власноручне подальше тренування цієї моделі також не є доцільним, адже на сьогоднішній день підтримка Microsoft Lobe завершилася та сам інструмент був визнаний застарілим[8].

Якщо хибно-позитивні ще можна виправляти вручну, то з хибно-негативними результатами виникають проблеми, адже потрібно переглядати усі фотографії, що завантажуються користувачами.

У зв'язку з цим, в рамках даної роботи було прийняте рішення знайти альтернативний спосіб виявлення непристойних фотографій, що міг би забезпечити більшу точність класифікації.

1.1.3. Розгляд існуючих засобів та методів вирішення задачі

Коли річ заходить про класифікацію зображень на основі комплексних та часто неочевидних факторів, відразу розглядається використання штучних

нейронних мереж, адже інші моделі машинного навчання, що доступні на даний момент, використовують більш прості алгоритмічні підходи та не можуть розглядати зображення в цілому.

На сьогоднішній день існує два основних засоби якісної класифікації зображень — це згорткові нейронні мережі (Convolutional Neural Networks або CNN) та трансформери бачення (Vision Transformer або ViT). Для того, щоб порівняти ці види нейронних мереж та обрати найкращий варіант для нашої задачі, розглянемо кілька ключових аспектів, таких як архітектура, ефективність на невеликих датасетах, здатність виявляти складні патерни, та результати, які вони демонструють при використанні попередньо навчених моделей.

CNN використовують згорткові шари для виявлення локальних патернів у зображеннях, поступово зменшуючи розмір вибірки у кожному наступному шарі[9]. Вони добре справляються із завданнями, що потребують розпізнавання текстур, контурів і просторових ієрархій[10]. CNN ефективно використовують параметри, що робить їх відносно компактними і менш вимогливими до обчислювальних ресурсів. Але до основних недоліків згорткових мереж можна віднести погану здатність виявляти глобальні зв'язки в зображенні, оскільки основний акцент робиться на локальні патерни.

Vision Transformers, у свою чергу, використовують під час обробки фотографій механізм уваги (attention mechanism). При цьому зображення розбивається на фрагменти (patches), що обробляються незалежно один від одного, а потім застосовується механізм уваги для виявлення глобальних зв'язків між цими фрагментами. До недоліків можна віднести потребу у значно більшій кількості даних та потужностей, необхідних для навчання “з нуля”, , але ця проблема значною мірою вирішується[11] за рахунок використання попередньо навченої моделі.

Винайдення трансформерів зробило новий крок у створенні інтелектуальних систем аналізу та генерації контенту, такого як текст (нині популярні GPT моделі), зображення та відео. Саме тому, проаналізувавши вищезазначені аспекти, було зроблено висновок, що використання Vision Transformer є кращим вибором для задачі аналізу якості та доцільності фотографій, адже поняття “якості” та “доцільності” є суб’єктивними і для правильної класифікації дуже важливим є розуміння глобального контексту фотографії[12], а не окремих її патернів.

Також, архітектура Vision Transformer показала високу ефективність під час використання у інших модулях платформи для знайомств, таких як виявлення контенту для дорослих та класифікація емоцій на фотографіях, що стало ще одним фактором на користь вибору саме цього типу нейронних мереж для вирішення нашої задачі.

1.1.4. Огляд архітектури Vision Transformer

Візуальні трансформатори (ViT)[13] - це інноваційна технологія для аналізу зображень, яка базується на трансформерних моделях. В архітектурі трансформерів в цілому, увага приділяється взаємодії між різними частинами послідовності, вперше це було продемонстровано на прикладі текстових даних. Застосування трансформерів до зображень дозволяє моделі розуміти контекст та взаємозв'язки між різними частинами зображення.

Vision Transformer

Results

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	88.55 ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4/88.5*
ImageNet ReaL	90.72 ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	99.50 ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06	—
CIFAR-100	94.55 ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08	—
Oxford-IIIT Pets	97.56 ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23	—
Oxford Flowers-102	99.68 ± 0.02	99.74 ± 0.00	99.61 ± 0.02	99.63 ± 0.03	—
VTAB (19 tasks)	77.63 ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70	—
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k

sovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).

Рис. 1.1. Ефективність тренування на різних датасетах та затрачені TPU-дні

Однак, важливо відзначити, що тренування моделей Vision Transformer вимагає значних обчислювальних CPU та TPU (рис. 1.1) ресурсів[13] (наприклад, тренування маленького ViT по типу DeiT-S займає близько 72 годин при використанні чотирьох GPU), що може бути складно забезпечити для більшості користувачів. Тому для проектів з обробки зображень зазвичай використовують уже попередньо навчені моделі, такі як *vit-base-patch16-224-in21k*[14] від Google. Ці пре-треновані моделі, як правило, навчаються на великих наборах даних і показують вражаючі результати у багатьох завданнях розпізнавання зображень. Модель може вивчити різні ознаки та структури на зображеннях, які можуть вказувати на наявність неочевидних факторів, на основі яких може бути створений класифікатор, який може визначати, чи відноситься зображення до певної категорії.

Розглянемо детальніше структуру Visual Transformer, що зображена на рис. 1.2. Слід зазначити, що оригінальні праці по цій темі [11], [13], [15] написані англійською мовою, тому далі у тексті для кращого розуміння будуть

використані україномовні адаптації понять від автора роботи, а в дужках зазначено оригінальні назви, що зображені на рисунку.

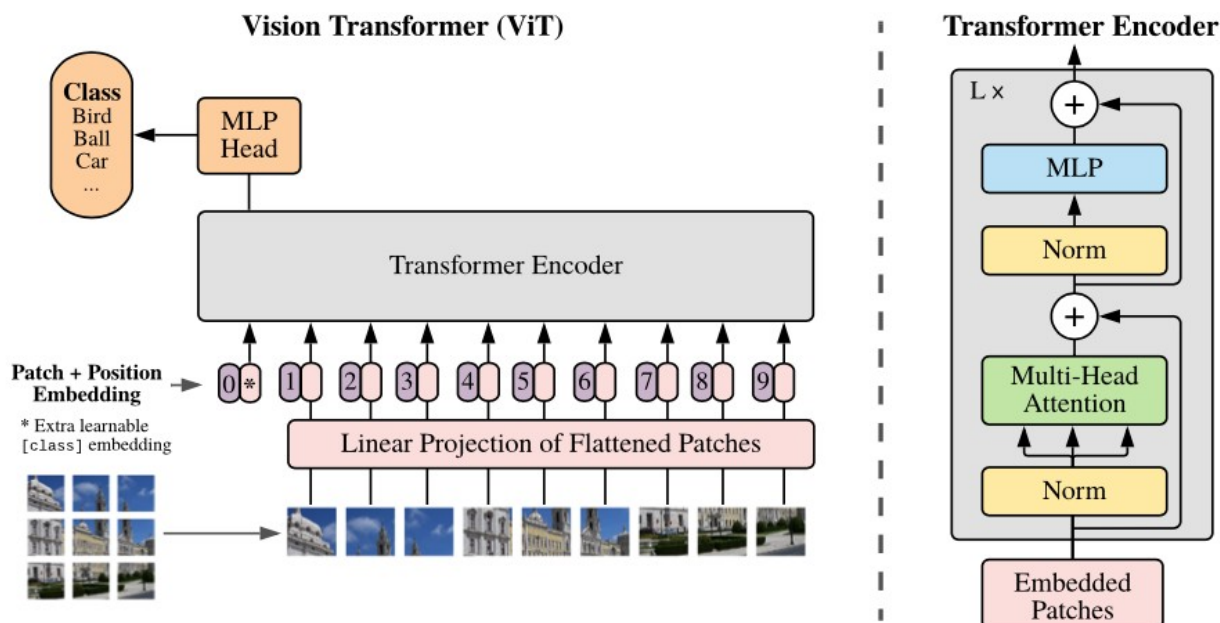


Рис. 1.2. Загальний алгоритм Vision Transformer

Vision Transformer використовує архітектуру трансформера для обробки зображень. Основними компонентами є фрагменти (Patches), на які розділяється фотографія та координати їх позиції (Position Embeddings), що перетворюють зображення на послідовність векторів, які можна обробляти кодувальником трансформера (Transformer Encoder). Кожен фрагмент зображення представляється вектором, а координати позиції додають інформацію про взаєморозташування фрагментів на початковому зображенні.

Transformer Encoder складається з Multi-Head Attention та MLP (Multi-Layer Perceptron). Multi-Head Attention дозволяє моделі фокусуватися на різних частинах вхідного зображення одночасно, а MLP забезпечує нелінійність та обробку ознак на кожному рівні[16]. Після цього виконується нормалізація, що стабілізує процес навчання та полегшує збір градієнтів.

На завершальному етапі виконується лінійна проекція розгорнутих фрагментів (Linear Projection of Flattened Patches), що дозволяє моделі робити

прогнози для класифікаційних завдань. Основні блоки моделі включають MLP Head, який генерує остаточний вихід моделі, та Transformer Encoder, який відповідає за обробку зображення у вигляді послідовності векторів.

1.2. Аналіз методів автоматизованої верифікації особистості

Окрім якості фотографій, однією з величезних проблем у сфері інтернет знайомств є підробні анкети, коли зловмисник намагається видати себе за іншу людину, завантажуючи її фотографії замість своїх. Найбільш популярні сервіси для знайомств вимагають від користувачів проходження верифікації особистості за допомогою завантаження фотографії або відео з деяким жестом. Далі ці матеріали потрапляють до людей-верифікаторів. Вони можуть бути як штатними співробітниками фірми, так і аутсорс-працівниками. Ці люди порівнюють отримані верифікаційні матеріали з фотографіями у профілі користувача, шукають ознаки підробки верифікаційних матеріалів та, якщо проблем не виявлено, помічають цього користувача як верифікованого.

Такий спосіб є доволі надійним, але вимагає значних людських ресурсів у випадку великої кількості користувачів сервісу. Також, варто пам'ятати, що користувач може змінювати фотографії у профілі вже після проходження верифікації. В ідеалі, ці нові фотографії також мають проходити через верифікаторів, але через затратність даного способу, лише невелика частина сервісів можуть собі дозволити такі повторні перевірки. Це дає змогу зловмисникам завантажити свої фотографії, пройти верифікацію, а потім видалити їх та завантажити фотографії особи, за яку вони хочуть себе видати. В результаті, у профілі, що помічений як верифікований, можуть бути фотографії іншої особи.

Детально вивчивши описану проблематику, було прийняте рішення замість людської праці використовувати засоби штучного інтелекту для верифікації особи користувачів. По-перше, це значно знизить витрати, необхідні на оплату роботи працівників-верифікаторів, по-друге, це дасть

можливість перевіряти усі нові фотографії на відповідність верифікованій особі.

1.2.1. Розпізнавання жестів для верифікації

Розглянувши способи верифікації, що використовують інші додатки для знайомств, виявилось, що більшість з них опирається на особливі жести. Користувачеві пропонується зробити селфі з деяким жестом (Рис. 1.3) та відправити цю фотографію на верифікацію. Після перевірки, профіль користувача помічається як верифікований.

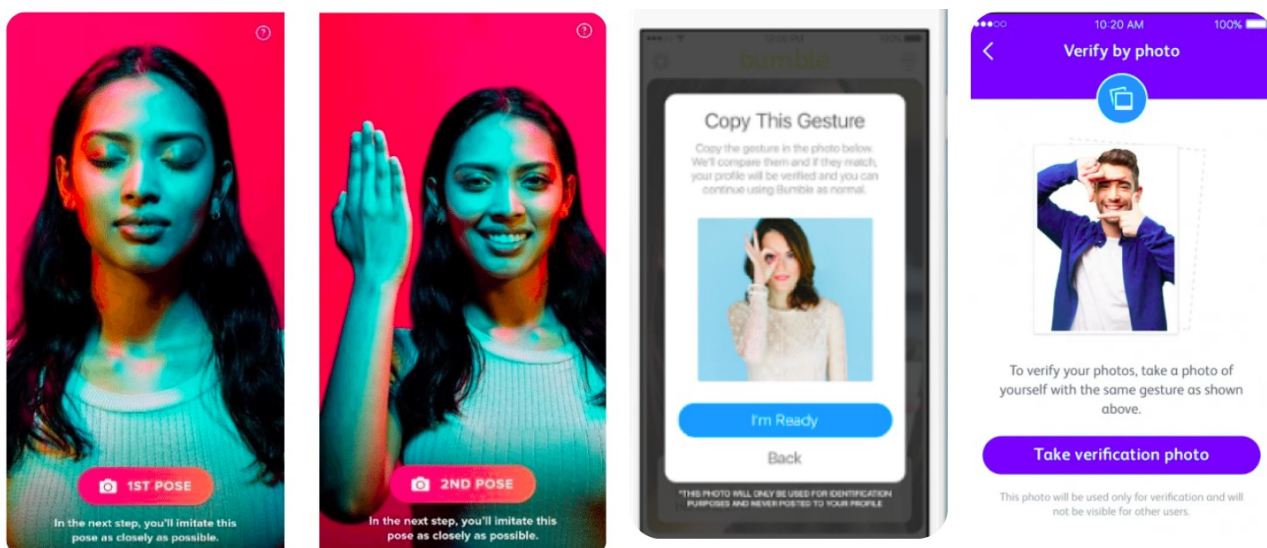


Рис. 1.3. Верифікація за жестом, яку використовують популярні додатки знайомств

На початку дослідження, нами також розглядалося використання схожого методу. Було проведено дослідження методів розпізнавання жестів у фотографіях із використанням сучасних бібліотек комп'ютерного зору. Результати цього дослідження опубліковані у збірнику тез XI Всеукраїнської науково–практичної конференції молодих учених, що відбулася 16 травня 2024 року в Київському столичному університеті імені Б. Грінченка[5]. У тезах описано процес впровадження розпізнавання жестів та запропоновано можливі технологічні рішення. Проте, згодом виявилися і значні недоліки такого метода.

Перш за все, це є ускладнення процесу для користувачів, оскільки виконання жесту вимагає стабільної фіксації пристрою, що може створювати незручності.

Іншою причиною є відносна легкість підробки таких фотографій, адже окрім великої кількості готових фотографій з подібним жестом, на яких можна просто замінити обличчя, самі жести можна накласти на необхідну фотографію людини за допомогою програм фотомонтажу.

Крім цього, в такому положенні камери обличчя займає лише невелику частину кадру, що значно зменшує точність розрахунку дескриптора ключових точок обличчя для подальшого його розпізнавання та порівняння з фотографіями у профілі користувача.

Замість цього було вирішено зосередитися на верифікації на основі розпізнавання обличчя, яке вже забезпечує високий рівень захисту та навіть використовується для біометричної авторизації [17].

1.2.2. Вибір інструментарію

Проектовану систему можна розділити на дві основні частини: клієнтську та серверну. Клієнтська відповідає за отримання даних з камери користувача та покрокове супроводження його через всі необхідні для проходження верифікації завдання. Серверна частина отримує дані від клієнтської, повторно їх перевіряє та виконує процес верифікації.

Так як систему верифікації планується інтегрувати до вже існуючої платформи знайомств, інструментарій розробки базуватиметься на інструментарії, що був використаний для розробки самої платформи.

Серверна частина верифікації буде інтегрована в існуючий код серверної частини основного додатку, тому вона буде виконана у вигляді модулю фреймворку Nest.js, що взаємодіє з існуючими модулями системи. Також, для збереження даних буде використовуватися окрема колекція в базі даних

MongoDB. Для роботи з базою даних використовуються бібліотеки Mongoose v7.1.0 та Kindagoose v1.3.5.

Клієнтська частина буде виконана у вигляді односторінкового веб-додатку (не PWA), що побудований на бібліотеці Vue.js версії 3. Для управління станом додатку використовуватиметься бібліотека Pinia. У якості збиральника, за свою швидкість, було обрано Vite.

Для обробки даних з відеокамери у реальному часі використовуватиметься бібліотека FaceMesh із пакету MediaPipe[18] від Google, що побудована на основі нейронного рушія Tensorflow.js. Також, слід зазначити, що у пакеті MediaPipe, також, є бібліотека для розпізнавання жестів[19], проте, як згадувалося у пункті 1.2.1. , ми не будемо її використовувати.

Клієнтська та серверна частини будуть комунікувати на основі REST API, представляючи інформацію у вигляді JSON об'єктів.

1.3. Аналіз методів генерацій першого повідомлення

На попередніх етапах дослідження ми зосередилися на вирішенні фундаментальних проблем, пов'язаних із якістю та автентичністю візуального контенту на платформах для знайомств та пошуку друзів. Однак, навіть маючи якісні та перевірені профілі, користувачі часто стикаються з іншою перешкодою – нездатністю ефективно розпочати знайомство.

Статистичний аналіз великої кількості чатів демонструє, що більшість користувачів використовують шаблонні фрази для першого повідомлення, такі як “Привіт, як справи?”, “Привіт, давай познайомимось”, або ж просто “Привіт” (Рис. 1.4). Велика кількість із них залишається без відповіді, адже користувача може втомлювати постійно пропонувати тему для розмови. Це призводить до одноманітності комунікації, знижує зацікавленість співрозмовника та, як наслідок, негативно впливає на загальний досвід користування платформою.

Така ситуація не лише знижує шанси на успішне знайомство, але й демотивує користувачів продовжувати використовувати сервіс.

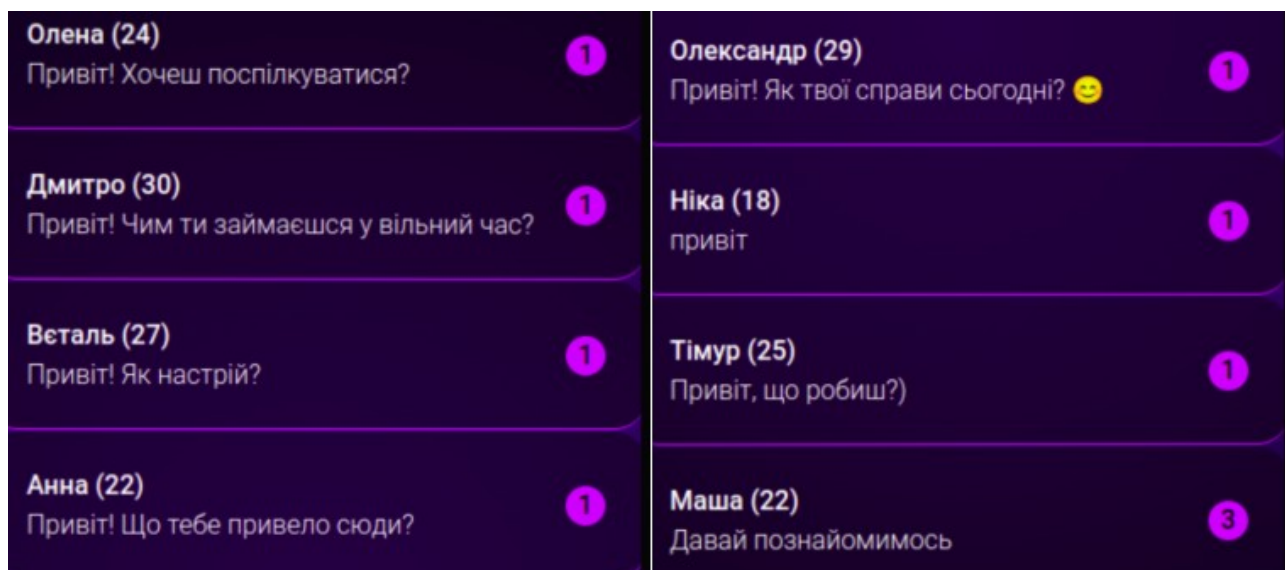


Рис. 1.4. Приклад діалогів популярного користувача

Однією з основних причин використання шаблонних фраз на платформах для знайомств є брак часу. Багато користувачів намагаються зекономити свої зусилля на кожному етапі знайомства, тому обирають готові формулювання для першого повідомлення. Це дає змогу швидко зв'язатися з більшою кількістю потенційних партнерів, не витрачаючи час на написання індивідуальних текстів.

Ще одним фактором може бути відсутність інформації, на основі якої можна розпочати персоналізовану комунікацію. Проте, на платформі для знайомств та пошуку друзів, що розглядається у даній роботі, цю проблему було вирішено введенням мінімального порогу заповненості профілю для відображення у стрічці користувачів.

Але часто користувачі просто стикаються з відсутністю креативності чи впевненості у собі, що також впливає на вибір шаблонних повідомлень. Не всі мають навички, необхідні для створення оригінального та привабливого повідомлення, а страх бути відкинутим часто змушує їх обирати більш безпечні, але передбачувані варіанти.

Цю проблему ускладнює велика кількість користувачів на платформах для знайомств. Прагнучи охопити більше потенційних партнерів, користувачі схильні спрощувати комунікацію і використовувати універсальні фрази, що не мотивують залучатися у спілкування. Негативний досвід, наприклад, відсутність відповідей на попередні повідомлення, також знижує мотивацію вкладати зусилля в унікальні тексти. У результаті перші повідомлення часто сприймаються як формальність, яка не має на меті створити справжній емоційний зв'язок. Це, у свою чергу, зменшує інтерес потенційного партнера до продовження спілкування. А якщо користувачі не знаходять мотивів продовжувати спілкування, це негативно відбивається на репутації платформи та може призвести до переходу на інші сервіси або взагалі до відмови від онлайн-знайомств.

Для вирішення описаної вище проблеми, було прийняте рішення розробити механізм генерації персоналізованих перших повідомлень на основі даних з профілів користувачів. Це можна зробити за допомогою алгоритмічного підходу, або ж, з використанням методів машинного навчання, таких як нейронні мережі. Нейронні мережі, на відміну від використання шаблонів, можуть створювати більш природні повідомлення, об'єднувати декілька пов'язаних інтересів інші параметри профілів для створення найбільш ефективного повідомлення.

1.3.1. Огляд архітектури Generative Pre-trained Transformer (GPT)

Розглянемо одну з найпопулярніших архітектур нейронних мереж, що активно використовується для задач генерації тексту - а саме, трансформери. Трансформери були вперше запропоновані у 2017 році в роботі "Attention Is All You Need"[20] і відтоді стали основою для багатьох потужних моделей, таких як GPT[21] (Generative Pre-trained Transformer), BERT та інші.

Архітектура трансформера була детально описана раніше у пункті 1.1.4. даної роботи, під час розглядання Vision Transformer для оцінки якості фотографій, тому коротко пригадаємо лише основні моменти у контексті аналізу та генерації тексту. Ключовим елементом трансформера є механізм уваги (Attention), що дозволяє моделі враховувати важливість кожного токена у контексті всього вхідного тексту. Трансформери використовують Multi-Head Attention, що обробляє вхідний текст паралельно за допомогою кількох «голів» уваги. Кожна «голова» здатна знаходити різні залежності між токенами, що підвищує здатність моделі до розуміння складних структур мови. Механізм уваги, який визначає, наскільки важливим є кожен токен в контексті інших. Нормалізація та шар повної зв'язності стабілізує процес навчання та забезпечує можливість обробки великих обсягів даних. Мережа зворотного проходу (Feed-Forward Neural Network) додатково обробляє інформацію після механізму уваги.

На відміну від Vision Transformer, де перетворення зображень у вектори відбувалося за допомогою фрагментів, процес роботи текстового трансформера починається з токенизації — поділу тексту на окремі елементи, що називаються токенами. Токен може бути словом, частиною слова або символом, залежно від конкретної моделі. Після токенизації кожен токен перетворюється на числове подання за допомогою механізму embeddings. Embeddings — це вектори фіксованої розмірності, що описують семантичне значення токенів, дозволяючи моделі враховувати контекст.

Трансформерні моделі, такі як GPT, проходять навчання у два етапи. Перший етап називається попереднім навчанням (pre-training), на ньому модель навчається на великому обсязі текстових даних для розуміння загальної структури мови, лексики та пунктуації. Другим етапом є до-навчання (fine-tuning) — адаптація моделі до специфічної задачі, наприклад, у нашому

випадку це генерація привітальних повідомлень для платформи знайомств та пошуку друзів.

У процесі використання модель приймає текстовий запит, токенизує його, пропускає через приховані шари, де відбувається обробка даних, і генерує відповідь, відновлюючи її з числових значень знову у текст.

Така структура дозволяє моделі ефективно обробляти контекст, знаходити смислові зв'язки та генерувати тексти, що максимально відповідають заданому запиту.

1.4. Постановка задачі дослідження

З метою вирішення задач, що впливають із проблем, розглянутих у попередній параграфі, було поставлено наступні задачі для дослідження:

1. Розробити класифікатор якості фотографій;
2. Розробити метод автоматизованої верифікації людини, зображеної на фотографіях у профілі;
3. Розробити спосіб генерації персоналізованих перших повідомлень на основі профілів користувачів;
4. Покращити виявлення непристойних фотографій;

Окремо розглянемо підзадачі для кожної із задач.

1.4.1. Класифікатор якості фотографій

Для вирішення проблеми, описаної в параграфі 1.1. , основним завданням є створення моделі машинного навчання на базі архітектури Vision Transformer, яка здатна автоматично аналізувати завантажені користувачами фотографії та класифікувати їх на два класи: "якісні" та "неякісні".

Для досягнення цієї мети необхідно вирішити наступні підзадачі:

1. Провести масове опитування користувачів сервісу щодо якості та доцільності конкретних фотографій.

2. Створити або зібрати набір даних, що містить фотографії, які будуть використовуватися для навчання та тестування моделі. Даний набір повинен містити якісні та неякісні зображення, а також їх оцінки.
3. Навчити модель на підготовленому наборі даних. Під час навчання модель повинна навчитися відповідно класифікувати фотографії на два класи: "якісні" та "неякісні".
4. Після завершення навчання, оцінити якість роботи моделі на наборі даних для валідації. Це допоможе з'ясувати, наскільки точно модель може класифікувати фотографії та виявляти неякісні зображення.
5. Після успішної оцінки моделі її потрібно підготувати для інтеграції у сервіс для знайомств.

Як було розглянуто у пункті 1.1.2. , окрім проблематики якості фотографій, існує також проблематика непристойних фотографій. Для них також необхідно застосувати або створити модель класифікації.

1.4.2. Верифікація особистості

Після огляду доступних засобів верифікації, які використовуються в інших сервісах для знайомств, що були розглянуті в параграфі 1.2. , було прийняте рішення замість жестів використовувати верифікацію за допомогою обличчя.

Для цього буде виконуватися порівняння завантажених фотографій з реальними зображеннями особи, отриманими за допомогою камери під час процесу верифікації. Верифікація здійснюватиметься через серію завдань, що перевіряють "живість" особи, і буде інтегрована з основним функціоналом платформи для знайомств.

Задачі, що необхідно вирішити:

1. Виявлення ключових точок обличчя.
2. Аналіз та вибір завдань для верифікації.

3. Проектування та реалізація системи для верифікації користувачів.
4. Порівняння обличчя, отриманого під час верифікації користувачів з обличчям на фотографіях у профілі.
5. Тестування та впровадження отриманої системи до існуючої платформи для знайомств та пошуку друзів.

1.4.3. Генерація першого повідомлення

Як розглядалося у пункті 1.3. , для вирішення проблеми встановлення першої комунікації між користувачами було прийняте рішення створити модель нейронної мережі, що буде аналізувати профілі обох користувачів та створювати унікальні персоналізовані привітання з відкритими запитаннями, що будуть слугувати гарним початком бесіди.

Створена та натренована модель повинна відповідати вимогам, зображеним на таблиці 1.

Таблиця 1: Вимоги до моделі генерації привітальних повідомлень

Вимога	Пояснення
Self-hosted	Модель повинна запускатися локально та мати можливість до-навчання у майбутньому.
Швидкість інференсу	Модель повинна працювати максимально швидко (в межах 1 секунди) з використанням CPU, підтримувати можливість працювати з GPU, та мати мінімальний необхідний розмір;
Двомовність	Модель повинна генерувати повідомлення англійською та українською мовами. Мова вказується при кожній генерації
Контекстуальність	Модель повинна генерувати повідомлення із фліртом у романтичному контексті та без нього у дружньому контексті. Контекст вказується при кожній генерації
Широке вікно контексту моделі	Профілі користувачів можуть займати до тисячі токенів, тому модель повинна підтримувати широке вікно контексту щоб правильно обробити отриману інформацію

Висновки до першого розділу

У першому розділі кваліфікаційної роботи було проведено детальний аналіз проблематики і можливостей покращення сервісів знайомств та пошуку друзів за допомогою засобів машинного навчання.

Особлива увага приділена впливу якості фотографій на ефективність профілів. Виявлено, що низька якість зображень значно знижує шанси користувачів на успіх у встановленні контактів, а також може негативно впливати на загальний досвід користування платформою. Для вирішення цієї проблеми запропоновано використання сучасної архітектури Vision Transformer для автоматичної класифікації якості фотографій. Це дозволить суттєво підвищити рівень візуального контенту.

Крім того, проаналізовано питання модерації непристойних фотографій. З'ясовано, що існуючі методи ручного виявлення таких фотографій потребують значних ресурсів і не завжди є ефективними. Використання методів машинного навчання, зокрема сучасних нейронних мереж, дозволить автоматизувати процес модерації, забезпечуючи комфорт та безпеку для користувачів.

Також розглянуто проблеми справжності та автентичності профілів. Недоліки поточних підходів до верифікації, таких як використання жестів, ускладнюють процес для користувачів і не гарантують високої точності. Замість цього запропоновано застосовувати технології розпізнавання обличчя, що дозволить спростити процес перевірки й підвищити його надійність.

Важливим аспектом виявлено складність налагодження першої комунікації між користувачами через використання однотипних і малоефективних привітальних повідомлень. Для вирішення цієї проблеми запропоновано механізм створення персоналізованих повідомлень на основі даних профілів за допомогою алгоритмів нейронних мереж, таких як GPT. Це сприятиме кращій взаємодії між користувачами.

РОЗДІЛ 2. РЕАЛІЗАЦІЯ ПОКРАЩЕННЯ СЕРВІСУ ДЛЯ ЗНАЙОМСТВ ТА ПОШУКУ ДРУЗІВ ЗА ЗАСОБАМИ МАШИННОГО НАВЧАННЯ

2.1. Класифікатор якості фотографій

Після аналізу проблематики якості фотографій та вибору Vision Transformer у якості архітектури моделі нейронної мережі, можна перейти до підготовки даних, що будуть використовуватися під час навчання моделі.

2.1.1. Збір інформації

Для навчання класифікатора необхідна велика кількість даних, у нашому випадку, це фотографії та оцінка їх якості. Для їх отримання можна скористатися одним з публічних датасетів, або ж сформувати свій власний. Так як тестуватися та застосовуватися ця модель буде у вже існуючому сервісі, найбільш логічним варіантом є використання реальних фотографій користувачів даного сервісу для формування власного датасету для навчання нейронної мережі, адже вони вже мають потрібний розмір, а також, їх завантажили реальні користувачі цього ж сервісу. Крім того, на момент написання цього розділу, на платформу було завантажено 1398 фотографій, чого цілком достатньо для навчання трансформеру.

Для збору інформації було прийнято рішення створити окрему сторінку, на якій користувачі можуть оцінити якість фотографій інших користувачів, та заохочувати людей переходити на неї та виставляти свої оцінки (Рис. 2.1).

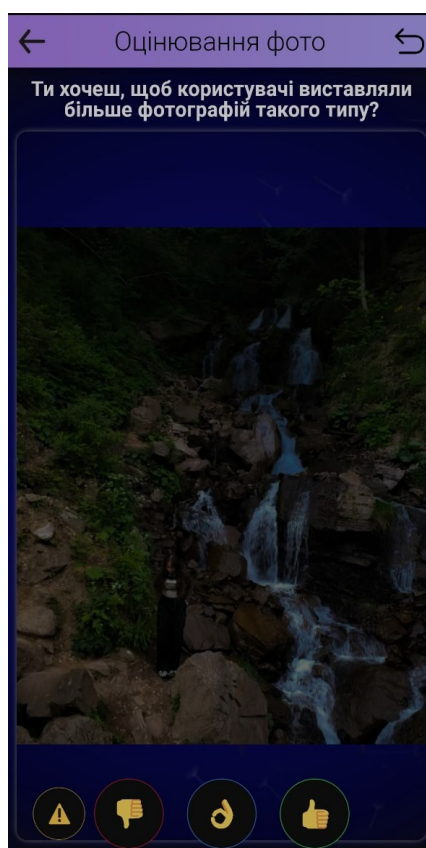


Рис. 2.1. Оцінювання фото

Користувачу пропонується три варіанти оцінки: палець вниз (не подобається), символ “окей” (буде вважатися за пропуск), та палець вверху (подобається). Символ “увага” зліва слугує для відправлення скарги на зображення.

Оцінки користувачів акумулюються та записуються в окрему колекцію бази даних, щоб мати змогу бути експортованими у майбутньому.

Система намагається, в першу чергу, дати користувачам на оцінку ті фотографії, що були оцінені найменше, таким чином, балансуючи кількість оцінок конкретної фотографії в сторону середнього значення.

За період в три місяці, вдалося зібрати оцінки для 1398 фотографій, з них 472 оцінено як якісні, 619 — як неякісні та 307 фотографій було пропущено. Пропущені зображення будуть оцінені вручну за критеріями, описаними у пункті 1.1.1. та використані для перевірки якості класифікації фотографій вже

навченої моделі. Співвідношення оцінок зображено на діаграмі нижче (Рис. 2.2).

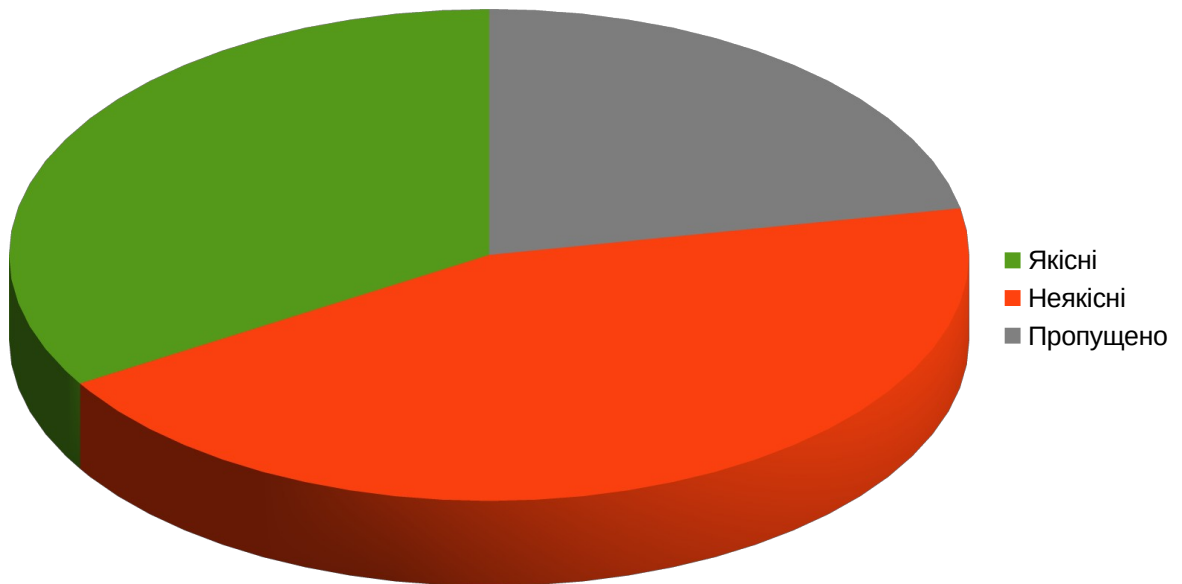


Рис. 2.2. Діаграма розподілення оцінок

Як бачимо з діаграми, більшість завантажених фотографій самими користувачами визнані неякісними, що додатково підтверджує проблему, описану у параграфі 1.1. .

2.1.2. Аналіз та підготовка зібраних даних

Отож, зібрані дані складаються з самих файлів фотографій та JSON масиву метаданих, що містить id усіх фотографій та масив з їх оцінками користувачами. Схема у форматі TypeScript інтерфейсу зображена на Рис. 2.3.

```
interface ImageQualityMetadata {  
  id: string;  
  createdAt: Date;  
  doLike: (-1 | 0 | 1)[];  
}
```

Рис. 2.3. Схема об'єкту метаданих

Цей масив дає можливість завантажити потрібні фотографії з серверу платформи та містить оцінки, на яких ми будемо навчати нашу модель. Оцінка може бути одним із варіантів: -1 (не подобається), 0 (не впевнений), 1 (подобається).

Першим кроком, нам необхідно завантажити масив з метаданими з сервера платформи. Після цього завантажуються файли усіх фотографій, адже навчання на локальних файлах відбувається значно швидше. Для цього проходимося по масиву, завантажуюмо фотографію по масці та зберігаємо у файл з відповідним id (Рис. 2.4).

```
import urllib.request
import os

os.mkdir('images')

def download(url, path):
    if not os.path.isfile(path): urllib.request.urlretrieve(url, path)

for item in dataset_manifest_processed:
    download(f'{URL}/{item["id"]}.jpg', f'{DATASET_FOLDER}/{item["id"]}.jpg')
```

Рис. 2.4: Завантаження фотографій з серверів платформи

Далі потрібно вирішити, яким чином ми будемо класифікувати фотографії для навчання, адже зараз для кожної фотографії ми маємо декілька оцінок, причому, їх кількість значно відрізняється від фотографії до фотографії, а також, існує оцінка 0, що означає невизначеність.

Після роздумів та емпіричних експериментів було розроблено наступну схему обрахунку фінальної оцінки: перед початком обробки, з масиву видаляються усі оцінки 0; у якості фінальної оцінки береться мода елементів масиву; у випадку, коли позитивних та негативних оцінок однакова кількість, міркуємо песимістично та виставляємо оцінку -1; якщо оцінок у фотографії немає (були лише оцінки 0, які ми видалили) — ця фотографія не буде використовуватися для навчання моделі, але на ній ми будемо перевіряти натреновану модель.

Песимістичний варіант міркування було обрано тому, що люди більше схильні ставити хороші оцінки, тому якщо фотографія зроблена погано, але видно, що людина старалася, журі можуть поставити більш високу оцінку. За оцінкою отриманих даних виявлено, що якщо фотографія якісна, то в неї або зовсім немає негативних оцінок, або їх незначна кількість, тому у випадку коли кількість позитивних та негативних оцінок однакова і загальна кількість оцінок більше за дві (ситуація 1:1), можна вважати, що у фотографії є проблеми.

Фінальний алгоритм обробки метаданих зображено на Рис. 2.5. Він перетворює масив оцінок `rates` у одне число `rate`, що може бути лише 1 або -1 (ми будемо використовувати всього два класи). Усі фотографії, що не вдалося розбити на ці категорії відфільтровуються.

```
def moda(array):
    moda, maximo = 0, 0

    for i in array:
        if array.count(i) > maximo:
            moda, maximo = i, array.count(i)

    return moda

dataset_manifest_processed = []

for item in dataset_manifest:
    rates = [rate for rate in item['rates'] if rate != 0]

    if len(rates) == 0:
        continue

    dataset_manifest_processed.append({
        'id': item['id'],
        'rate': moda(rates)
    })
```

Рис. 2.5. Попередня обробка метаданих

Тепер, коли фотографії завантажені та метадані попередньо оброблені, можна приступити до вибору базової моделі та підготовки датасету, на якому ми будемо навчати нейронну мережу.

2.1.3. Вибір базової моделі

Як ми вже згадували під час опису архітектури трансформерів, у цього типу нейронних мереж є одна особливість. В загальному, для їх навчання потрібні значні затрати обчислювальної потужності (зокрема, відеокарт) та величезна кількість даних (значно більша ніж для згорткових нейронних мереж), що починається від декількох мільйонів фотографій. Але коли модель вже навчена та вміє визначати конкретні образи, її можна до-навчити на своїх даних та класах і для цього потрібно значно менша кількість фотографій. Такий процес до-навчання прийнято називати *fine-tuning*. У нашому випадку, ми маємо трохи більше 1000 фотографій, чого не вистачить для навчання трансформера “з нуля”, але цього цілком достатньо для того, щоб навчити попередньо навчену модель класифікувати фотографії потрібним для нас чином.

За основу ми візьмемо модель *vit-base-patch16-224-in21k[14]* від Google, що дуже добре зарекомендувала себе у якості складного класифікатора зображень (вищезгадана платформа для знайомств вже використовує її до-навчені версії для виявлення контенту для дорослих та класифікації емоцій на обличчі). Вона була попередньо натренована на наборі даних ImageNet-21k[22], що містить 14 мільйонів зображень 224x224px та 21843 класи для них. Потім модель була до-навчена на більш специфічному наборі даних ImageNet 2012, що містить 1 мільйон зображень та 1000 класів. Це значно покращило її здатність до класифікації об'єктів у широкому спектрі категорій. Використання такого підходу дозволяє моделі зберігати багаті і узагальнені уявлення про різні типи зображень, що підвищує її продуктивність на нових, раніше не бачених даних.

Також, існує багато проектів, які вже використовують дану модель у якості основи для своїх власних моделей. Наприклад, у сфері харчових технологій вона використовується для класифікації їжі, оцінки її об'єму та

калорійності. Один із проектів минулого року включає використання моделі vit-base-patch16-224-in21k, що була додатково навчена на датасеті Food101, у поєднанні з Apple AR Photogrammetry і Nutritionix API для надання комплексного аналізу харчових продуктів на основі зображень[23].

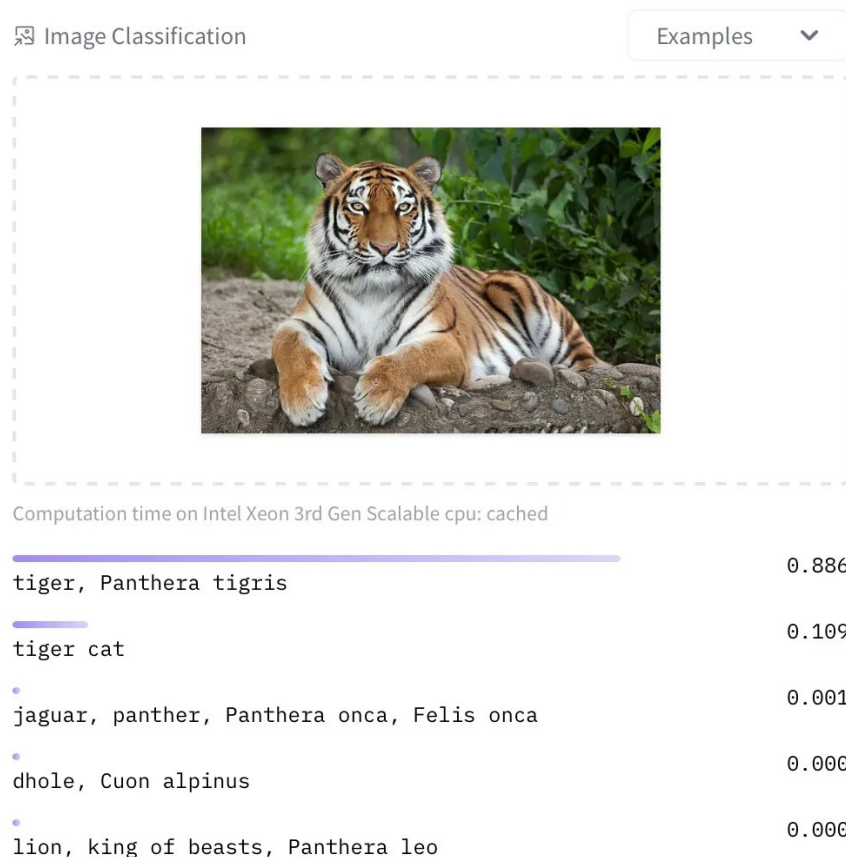


Рис. 2.6. Приклад роботи моделі без до-навчання

На Рис. 2.6 зображено приклад того, як працює класифікатор базової моделі без до-навчання. Як можна побачити, модель натренована розпізнавати, в тому числі, тварин.

2.1.4. Формування та підготовка датасету

Під час перших експериментів, фотографії просто завантажувалися у пам'ять, перетворювалися у тензори та віддавалися мережі на навчання, але на практиці виникла проблема переповнення оперативної пам'яті, не дивлячись на запуск навчання на серверах Google Colab та ручні виклики garbage collector у

Python. Тому було вирішено використовувати стандартний підхід для створення датасетів для подальшої класифікації зображень від бібліотеки PyTorch[24] та сформувати датасет за допомогою класів DataLoader та ImageFolder.

Цей підхід вимагає того, щоб фотографії були фізично розбиті на окремі теки, що будуть називатися так само, як і класи, за якими буде відбуватися класифікація. У нашому випадку, це будуть класи **good** та **bad**. На Рис. 2.7 зображено код, що створює відповідні теки, проходиться по всім фотографіям та переміщує їх у відповідні теки згідно оцінки.

```
import shutil

for item in dataset_manifest_processed:
    imageClass = "good" if item["rate"] > 0 else "bad"

    if not os.path.exists(f'{DATASET_RANKED_FOLDER}/{imageClass}'):
        os.makedirs(f'{DATASET_RANKED_FOLDER}/{imageClass}')

    if not os.path.isfile(f'{DATASET_FOLDER}/{item["id"]}.jpg'):
        print(f"NOT FOUND {item['id']}")
        continue

    shutil.copy(
        f'{DATASET_FOLDER}/{item["id"]}.jpg',
        f'{DATASET_RANKED_FOLDER}/{imageClass}/{item["id"]}.jpg'
    )
```

Рис. 2.7: Розподілення фотографій за класовими теками

У кінцевому варіанті ми отримали 472 зображення у класі “good” та 620 зображень у класі “bad”. На перший погляд видно перевагу у бік неякісних фотографій, але вона невелика і в архітектурі трансформера це не повинно негативно впливати на оцінку хороших фотографій.

Після того, як фотографії розміщені у відповідних теках, потрібно ініціалізувати сам DataLoader (Рис. 2.8). Так як ми використовуємо CUDA, розмір пачки (batch_size) визначається таким чином, щоб повністю вміщуватися у відеопам’ять, не переповняючи її, адже це може призвести до помилки під час навчання. Для більш рівномірного розподілення фотографій (користувачі могли

додавати декілька схожих фотографій підряд), додано рандомізацію за допомогою параметра `shuffle`. Також, для коректного виконання у середовищі Google Colab, потрібно вказати `num_workers=2` та `pin_memory=True`.

```
import torch
import torchvision.datasets as datasets

# Створимо набір даних ImageFolder
dataset = datasets.ImageFolder(root=DATASET_RANKED_FOLDER, transform=transform)
# Вкажемо розмір пачки та пристрій навчання
batch_size = 16
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
data_loader = torch.utils.data.DataLoader(dataset,
    batch_size=batch_size, shuffle=True, num_workers=2, pin_memory=True)
print('Classes:', dataset.classes)
```

Рис. 2.8: Ініціалізація DataLoader

Як можна побачити на Рис. 2.8, окремим параметром при ініціалізації класу `ImageFolder` передається функція трансформації, що перетворює фотографію з масиву байтів у тензор. Розглянемо її вміст на Рис. 2.9.

```
import torchvision.transforms as transforms

transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
])
```

Рис. 2.9: Функція функція трансформації зображення у тензор

Функція трансформації складається з трьох послідовних операцій, об'єднаних за допомогою `transforms.Compose` (Рис. 2.9). Спочатку, ми виконуємо зміну розміру зображення із 640x800 (стандартний розмір зображення на платформі) на 224x224 (розмір, під яку спроектовано базову модель). Далі масив байтів зображення перетворюється на тензор. Третім кроком, згідно документації базової моделі[14], виконується нормалізація RGB каналів.

2.1.5. Тренування моделі на власному датасеті

Наступним кроком потрібно завантажити модель та виконати її донавчання на нашому датасеті. Модель ми будемо завантажувати з відкритого репозиторію на Hugging Face (<https://huggingface.co/google/vit-base-patch16-224-in21k>) за допомогою бібліотеки transformers[25], код зображено нижче на Рис. 2.10.

```
[ ] from transformers import AutoModelForImageClassification
    import torch
    import torch.nn.functional as F

[ ] model = AutoModelForImageClassification.from_pretrained("google/vit-base-patch16-224-in21k")
```

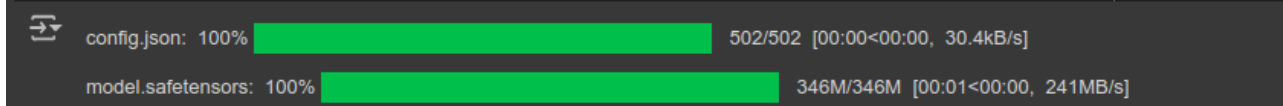


Рис. 2.10. Завантаження та підключення базової моделі

Далі потрібно трохи змінити її архітектуру, адже модель навчена класифікувати зображення на 1000 різних класів, при чому, жоден з них не відповідає нашим потребам. Зміна архітектури зводиться до модифікації кількості нейронів на вихідному шарі з тисячі до двох (Рис. 2.11).

```
# Отримуємо кількість входів моделі
num_features = model.classifier.in_features
# Змінюємо вихідний шар моделі на лінійний з двома виходами
model.classifier = torch.nn.Linear(num_features, 2)
```

Рис. 2.11. Модифікація архітектури нейронної мережі для класифікації за двома класами

Після цього, напишемо код, що буде підтягувати зображення та відповідний клас, відправляти їх на відеокарту[24] (для пришвидшення процесу навчання, використовуємо технологію CUDA[26] від NVidia), отримати передбачення моделі, співвіднести його з дійсним класом та виконати модифікацію вагів за допомогою методу зворотного розповсюдження помилки[27] (back propagation). Нижче, на Рис. 2.12, зображено прокоментований код, що виконує ці операції.

```

model.to(device) # Переносимо модель на відеокарту

for epoch in range(epochs):
    print("### Epoch", epoch)

    for i, (images, labels) in enumerate(data_loader):
        # Переносимо тензори зображення та класів на відеокарту
        images = images.to(device)
        labels = labels.to(device)
        # Обчислюємо передбачення моделі
        outputs = model(images)
        logits = outputs.logits
        # Обчислюємо помилку (метод CrossEntropyLoss)
        loss = criterion(logits, labels)
        print("Loss", loss)
        # Оновлюємо ваги моделі щоб компенсувати помилку (оптимізатор Adam)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

```

Рис. 2.12. Код до-навчання моделі на власних даних

Для навчання використовується функція втрати перехресної ентропії (cross-entropy loss)[27] та класичний оптимізатор Adam[24]. Кількість епох, методом експериментів, виставлена у значення 20 епох. На Рис. 2.13 зображено конфігурування навчання нейронної мережі (цей код виконується перед попереднім).

```

from torch.optim import Adam

criterion = torch.nn.CrossEntropyLoss()
optimizer = Adam(model.classifier.parameters(), lr=0.001)
epochs = 40 # Кількість епох

```

Рис. 2.13. Конфігурування навчання моделі

Значення помилки на кожній епосі зображено на Рис. 2.14 у вигляді графіка. По осі X зображені самі епохи (в даному випадку їх 40), з відсіченою першою цифрою для компактності розташування, по осі Y зображена Cross Entropy Loss. Графік має логарифмічний вигляд і на 40 епосі навчання майже зупиняється.

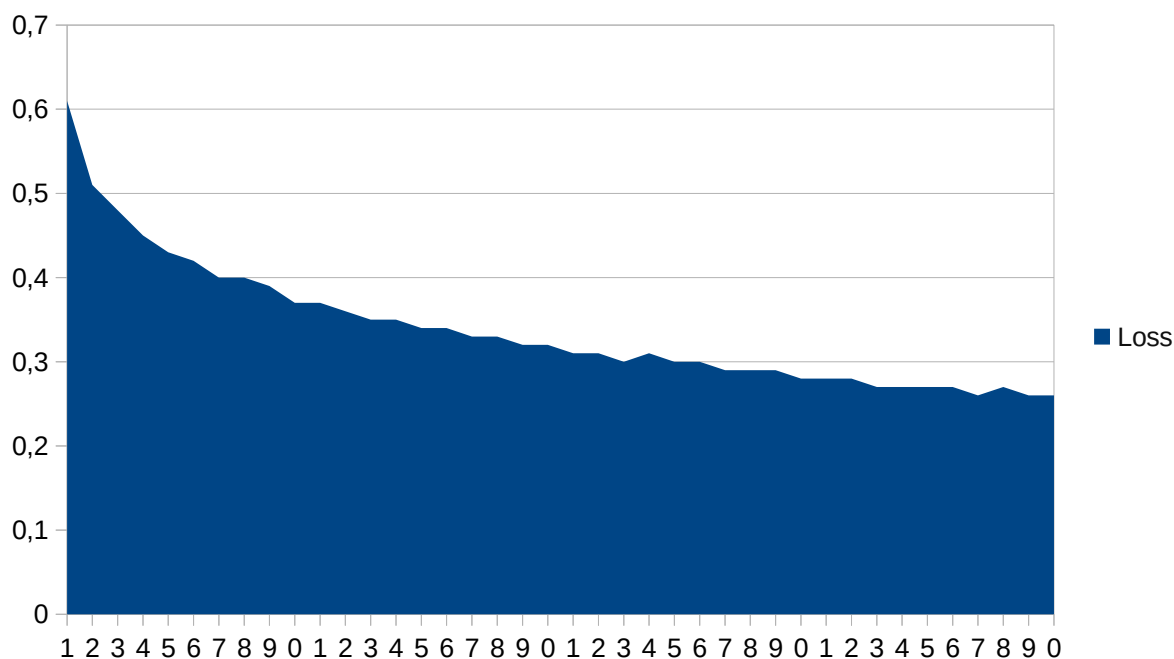


Рис. 2.14. Графік значення помилки відносно епохи

Методом експериментів виявлено, що, не дивлячись на зниження значення помилки ще на 5%, після 20 епохи починається перенавчання моделі, тому було прийнято рішення у фінальному варіанті зменшити кількість епох навчання до 20.

Після завершення навчання, ми отримали модель, готову для збереження та використання. Перевірка на тестовій вибірці показала дуже хороший результат, іноді модель робила більш правильні висновки щодо якості та доцільності фотографій, ніж живі люди, якщо керуватися критеріями, які ми визначили у пункті 1.1.1. першого розділу.

Для збереження моделі у форматі PyTorch використовується метод `torch.save` (Рис. 2.15).

```
torch.save(model, f'{DATASET_FOLDER}/model.pt')
```

Рис. 2.15. Збереження моделі для подальшого використання

2.1.6. До-навчання моделі на окремих фотографіях

Хоча нам і вдалося досягти точності у 87.5%, іноді модель може робити неправильні висновки, тому нам потрібно додати можливість корегувати ваги моделі на окремих фотографіях.

Алгоритм до-навчання на окремих фотографіях майже не відрізняється від навчання на датасеті, але містить вагомі відмінності, тому ми опишемо його у даному пункті.

У кінцевому результаті, ми маємо отримати функцію, що отримує на вхід id зображення та клас, до якого мережа повинна його відносити. Функція повинна завантажити зображення у пам'ять, натренувати модель, вивантажити зображення з пам'яті та зберегти модель.

Першим кроком, нам необхідно завантажити саме зображення. У нашому випадку, нейронна мережа матиме прямий доступ до сховища зображень, тому завантаження відбуватиметься з диску, а не шляхом HTTP запиту. При навчанні на датасеті, весь процес завантаження зображень у пам'ять виконував DataLoader, але зараз нам потрібно виконати усі кроки самостійно. Для цього скористаємося бібліотекою PILLOW (Рис. 2.16). Так як ми завантажуюмо одне зображення, а не пачку, як це робить DataLoader, в кінці потрібно виконати метод `unsqueeze` отриманого тензора, що зробить віртуальну пачку для навчання з одного елемента.

```
import PIL

def loadSingleImage(path):
    image = PIL.Image.open(path).convert("RGB")
    tensor = transform(image).to(device)
    image.close()
    return tensor.unsqueeze(0)
```

Рис. 2.16. Завантаження зображення у пам'ять, перетворення його у тензор та перенос на відеокарту

Далі створюємо функцію для класифікації зображення (Рис. 2.17). Для самого до-навчання вона не потрібна, але ми хочемо побачити, що до-навчання виконалося коректно та після тренування, модель класифікує зображення саме за тим класом, який ми передали. Модель повертає кортеж, де перший елемент це клас, а другий — впевненість моделі (дробове число від 0 до 1).

```
import torch.nn.functional as F

def predict(path, classes=dataset.classes):
    tensor = loadSingleImage(path)
    output = model(tensor)
    logits = output.logits # Діставання тензору логітів
    # Застосування softmax для конвертації логітів у ймовірності
    probs = F.softmax(logits, dim=1)
    predicted_class = probs.argmax(dim=1).item()
    predicted_prob = probs[0][predicted_class].item()
    return classes[predicted_class], predicted_prob
```

Рис. 2.17. Функція для класифікації зображення

Після цього, напишемо саму функцію до-навчання (Рис. 2.18). Важливо на цьому етапі виставити ту саму кількість епох, що використовувалася для навчання моделі на датасеті задля уникнення перенавчання або недостатнього навчання.

```
def finetune(path, realClass, epochs=epochs):
    if not (realClass in ('good', 'bad')):
        raise Exception(f'Unknown class {realClass}')

    classLabel = 1 if realClass == 'good' else 0
    tensor = loadSingleImage(path)
    labels = torch.tensor([classLabel]).long().to(device)

    for epoch in range(epochs):
        output = model(tensor) # Обчислюємо передбачення мережі
        logits = output.logits
        loss = criterion(logits, labels) # Обчислюємо помилку
        optimizer.zero_grad() # Оновлюємо ваги
        loss.backward()
        optimizer.step()
```

Рис. 2.18. Функція для до-навчання моделі на одній фотографії

Далі перевіряємо процес до-навчання шляхом отримання передбачення моделі на фотографії, виконання до-навчання з протилежним класом та перевірки, що модель почала класифікувати це зображення за новим класом.



Рис. 2.19. Спочатку зображення класифікувалося як 71% хороше

Як бачимо на Рис. 2.19 та Рис. 2.20, алгоритм працює коректно та передбачення моделі дійсно змінюється.

```
[ ] finetune('./original.jpg', 'bad')  
↔ tensor([-0.1377,  0.0439]), device='cuda:0', grad_fn=<AddmmBackward0>  
Loss: tensor(0.7881, device='cuda:0', grad_fn=<NllLossBackward0>)  
  
[ ] predict('./original.jpg', ['bad', 'good'])  
↔ ('bad', 0.5161916613578796)
```

Рис. 2.20. Після до-навчання - 51% погане

2.1.7. Розпізнавання непристойних фотографій

Після успішного досвіду з використанням Vision Transformer для класифікації фотографій за якістю, було прийняте рішення використати цей самий підхід і для розпізнавання непристойних фотографій. Окрім високої ефективності Vision Transformer, це дасть змогу використовувати механізми донавчання, описані у пункті 2.1.6. , для корегування відповідей моделі у випадку виявлення некоректної класифікації, що дасть більшу гнучкість у практичному застосуванні в рамках додатку для знайомств та пошуку друзів.

Перед тренуванням власної моделі, було виконано огляд попередньо навчених моделей за цією тематикою. Виявилось, що проблематика розпізнавання непристойних фотографій є актуальною для багатьох інших сфер та за час, що пройшов із написання кваліфікаційної роботи бакалавра, було створено декілька готових моделей, що вирішують нашу задачу.

Знайдені моделі були завантажені для порівняння з метою виявлення найкращого кандидату для застосування. Результат порівняння зображено нижче, на таблиці 2. Порівняння було проведено в ручному режимі на вибірці з 56 фотографій, серед яких були пристойні, непристойні та відомі хибно-позитивні та хибно-негативні результати моделі від suko.

Таблиця 2: Порівняння моделей розпізнавання непристойних фотографій

Модель	Хибно-позитивні	Хибно-негативні
suko/nsfw[7]	24	13
Falconsai/nsfw_image_detection[28]	0	7
LukeJacob2023/nsfw-image-detector (strict)[29]	6	5
LukeJacob2023/nsfw-image-detector (loyal)[29]	0	27

Під «хибно-позитивними» результатами мається на увазі ситуація, коли модель визнала непристойною нормальну фотографію. Під «хибно-негативними» - коли непристойну фотографію визнано нормальною.

На відміну від інших моделей, що видають два класи, модель LukeJacob2023/nsfw-image-detector класифікує фотографії за наступними категоріями: `neutral`, `porn`, `sexy`, `hentai` та `drawings`. Тому її було додано до таблиці у двох «режимах»: `strict` та `loyal`. У `strict` режимі фотографії класу `sexy` рахуються за непристойні, а у `loyal` — за нормальні.

За результатами, зображеними на таблиці 2, можна зробити наступні висновки:

1. У моделі `suko/nsfw` найгірші результати, тому її дійсно варто замінити;
2. Модель `LukeJacob2023/nsfw-image-detector` у `strict` режимі видає менше всього хибно-негативних результатів, проте, помічає як непристойні деякі нормальні фотографії, що може призвести до скарг користувачів;
3. Модель `Falconsai/nsfw_image_detection` показала найкращий збалансований результат;

Таким чином, на основі результатів порівняння, було прийняте рішення використати модель `Falconsai/nsfw_image_detection`.

Застосування моделі відбувається через механізм `pipeline`, що надається бібліотекою `Transformers`.

```
from transformers import pipeline

nsfw_detector = pipeline(
    task = "image-classification",
    model="Falconsai/nsfw_image_detection"
)
```

Рис. 2.21. Застосування моделі розпізнавання непристойних фотографій

На вхід `nsfw_detector` отримує зображення у форматі бібліотеки `Pillow`. На виході віддає список з вірогідностями двох класів `normal` та `nsfw`. Першим у списку розміщено клас із найбільшою вірогідністю, отже, у найпростішому випадку застосування достатньо перевіряти, чи поле першого елементу дорівнює `nsfw`.

2.2. Процес верифікації особистості

Перейдемо до розробки процесу верифікації особистості користувача за фотографіями. Цей процес включатиме як клієнтську, так і серверну частину, які ми поступово розглянемо у даному параграфі. Проте, для початку познайомимося із основним робочим поняттям — ключовими точками обличчя.

2.2.1. Аналіз ключових точок на обличчі FaceMesh

Бібліотека FaceMesh отримує на вхід зображення з камери, а на виході видає масив координат 468 окремих ключових точок (англ. landmarks)[30]. Кожна точка містить у собі X, Y та Z координати. Для отримання конкретної ключової точки з масиву необхідно знати її індекс. Мапу індексів зображено на Рис. 3. Крім того, групування цих точок можна побачити у коді бібліотеки за посиланням

<https://github.com/tensorflow/tfjs-models/blob/838611c02f51159afdd77469ce67f0e26b7bbb23/face-landmarks-detection/src/mediapipe-facemesh/keypoints.ts>.

Повна мапа ключових точок у вигляді фотографії зображена у додатку В. Для цілей верифікації, ми будемо використовувати лише невелику кількість з цих точок, а саме, точки рота, очей та брів. Для визначення положення голови у просторі, також можуть бути використані точки носа та вух (самі вуха не входять до мапи FaceMesh, проте, точками вух вважаються точки на лівій та правій частинах обличчя, де вуха анатомічно з'єднуються з черепом).

Для зручності використання, оголосимо константи для індексів точок, що будуть використовуватися під час верифікації. Їх список зображено на Рис. 2.22.

```

export const LANDMARK = {
  NOSE_BOTTOM: 2, NOSE_TOP: 6, NOSE_RIGHT: 131, NOSE_LEFT: 360, NOSE_TIP: 1,

  EYE_RIGHT_RIGHT: 33, EYE_RIGHT_LEFT: 133, EYE_RIGHT_TOP: 159,
  EYE_RIGHT_BOTTOM: 145,
  EYE_LEFT_RIGHT: 362, EYE_LEFT_LEFT: 263, EYE_LEFT_TOP: 386,
  EYE_LEFT_BOTTOM: 374,

  MOUTH_RIGHT: 61, MOUTH_LEFT: 291, MOUTH_TOP: 13, MOUTH_BOTTOM: 14,

  EYEBROW_RIGHT_RIGHT: 63, EYEBROW_RIGHT_LEFT: 107,
  EYEBROW_RIGHT_CENTER: 66, EYEBROW_LEFT_CENTER: 296,
  EYEBROW_LEFT_RIGHT: 336, EYEBROW_LEFT_LEFT: 293,

  EAR_RIGHT: 234, EAR_LEFT: 454,
};

```

Рис. 2.22. Індеси основних ключових точок FaceMesh

2.2.2. Вибір завдань для верифікації

Проходження верифікації полягатиме у проходженні набору випадково обраних завдань. Метою цих завдань буде виявлення живості особи, що їх проходить для подальшого порівняння «достовірно отриманих» фотографій з фотографіями, які користувач завантажив на платформу знайомств.

До завдань ставитимуться наступні вимоги (стосовно більшості людей): вони повинні бути чіткими та зрозумілими, легкими для виконання за мінімальний час та повинні перевіряти живість піддослідного. Сформований список завдань з описами зображено нижче на таблиці 3.

Завдання обиратимуться на стороні серверу під час першої ініціалізації процесу верифікації та будуть прив'язані до акаунту користувача. Це дасть змогу значно ускладнити використання завчасно підготовлених фото та відео матеріалів, які зловмисники вживають для проходження процесу підтвердження особистості. Під час ініціалізації завдань, обиратимуться не лише самі завдання з переліку можливих, а ще їх порядок та індивідуальні параметри кожного завдання (наприклад, яке саме око потрібно закрити).

Таблиця 3: Список завдань для верифікації

Завдання	Опис
Кліпання очима	Для проходження цього завдання потрібно на секунду закрити очі (одне конкретне або обидва), а потім відкрити.
Відкриття рота	Для проходження цього завдання потрібно на секунду широко відкрити рот, а потім закрити
Посмішка	Для виконання цього завдання потрібно на секунду широко посміхнутися, а потім повернутися в спокійний стан
Рух бровами	Для виконання цього завдання потрібно на секунду підняти брова, далі опустити їх (насупити), а в кінці повернути у стан спокою
Нахил голови	Для проходження цього завдання необхідно нахилити голову у конкретний бік (вліво, вправо, вниз, вверх), а потім повернути у рівне положення
Поворот голови	Для проходження цього завдання необхідно повернути голову у конкретний бік (вліво, вправо), а потім повернути у рівне положення

Проходження цих завдань дасть змогу переконатися у живості особи, що проходить верифікацію. Тепер перейдемо до практичної реалізації поставлених задач та завдань.

2.2.3. Ініціалізація камери та FaceMesh

Перед початком роботи із зображенням, необхідно отримати дозвіл від користувача браузеру на доступ до камери. Першим важливим фактором, який для цього необхідно врахувати є захищене підключення (HTTPS), адже у разі використання HTTP, браузера користувача не надасть доступ до камери.

Нижче, на Рис. 2.23, зображено процес отримання доступу до камери. Варто зазначити, що якщо користувач натисне на кнопку “заборонити”, повторно отримати доступ на мобільних пристроях буде вкрай важко. Тому перед викликом системного запиту на дозвіл до камери слід переконатися, що користувач готовий до цього запиту та відповідь на нього позитивно.

```

1  const media = await navigator.mediaDevices?.getUserMedia({
2    video: {
3      width: {
4        ideal: 640
5      }
6    }
7  });
8
9  if (media && videoEl.value && canvasEl.value) {
10   const settings = media.getVideoTracks()[0].getSettings();
11
12   videoEl.value.width = canvasEl.value.width = settings.width!;
13   videoEl.value.height = canvasEl.value.height = settings.height!;
14
15   const camera = new cam.Camera(videoEl.value, {
16     onFrame: () => faceMesh?.send({ image: videoEl.value }),
17   });
18
19   camera.start();
20 }

```

Рис. 2.23. Отримання доступу до камери користувача

Після того, як доступ до камери отримано, переходимо до підключення та налаштування FaceMesh. Налаштовуємо мінімальну “впевненість” у значення 0.95 (Рис. 2.24, стрічка 6), що вимагатиме високу чіткість камери та гарну освітленість кадру для того, щоб модель почала повертати ключові точки. До цього моменту відображатиметься повідомлення з проханням перевірити якість картинки та освітлення.

Далі підписуємось на отримання ключових точок обличчя за допомогою методу onResults (Рис. 2.24, стрічка 8). У ньому ми відображаємо ці ключові точки на Canvas елементі поверх відеокадру з камери. Детальніше цей код показано нижче на Рис. 2.25. Після цього, у майбутньому, буде виконуватися обробка виконання верифікаційних завдань.

```

1 faceMesh = new facemesh.FaceMesh({
2   locateFile: file =>
3     (`https://cdn.jsdelivr.net/npm/@mediapipe/face_mesh/${file}`);
4 });
5 faceMesh.setOptions({ maxNumFaces: 1, refineLandmarks: true,
6   minDetectionConfidence: 0.95, minTrackingConfidence: 0.95 });
7
8 faceMesh.onResults((results) => {
9   let landmarks = results.multiFaceLandmarks?.[0];
10  if (!landmarks) return (isFaceFound.value = false);
11  repaintCanvas(landmarks);
12  drawChartLandmarks(getNormalizedLandmarks(landmarks));
13  doInstructionCheck(landmarks); // ...Обробка завдань
14 });

```

Рис. 2.24. Ініціалізація бібліотеки FaceMesh

Далі, з метою налагодження, налаштовуємо проекцію ключових точок на відеопотік (Рис. 2.25). Для цього на сторінці є прихований Canvas елемент, що відображається лише у режимі відлагодження. На ньому малюється останній кадр з відеокамери, а потім поверх накладається проекція тривимірної моделі ключових точок FaceMesh, схожа із зображеною у додатку В.

```

const toLandmarksArray = (indices: number[]) => {
  const output: [number, number][] = [];
  for (let i = 1; i < indices.length; i++)
    output.push([indices[i - 1], indices[i]]);
  output.push([indices[indices.length - 1], indices[0]]);
  return output;
};
const repaintCanvas = (landmarks: Landmarks) => {
  canvasCtx.drawImage(videoElement, 0, 0, cameraSize.width, cameraSize.height);
  drawingUtils.drawConnectors(canvasCtx, landmarks,
    facemesh.FACEMESH_TESSELATION, { color: "#C0C0C070", lineWidth: 1 } );
  const draw = (color: string, landmarksArray: number[]) =>
    drawingUtils.drawConnectors(canvasCtx, landmarks,
      toLandmarksArray(landmarksArray), { color, lineWidth: 3 } );
  // Left eye
  draw("#FF0000", [ LANDMARK.EYE_LEFT_LEFT, LANDMARK.EYE_LEFT_TOP,
    LANDMARK.EYE_LEFT_RIGHT, LANDMARK.EYE_LEFT_BOTTOM ]);
  // ...Решта прикладів опущено для компактності
}

```

Рис. 2.25. Відображення ключових точок на кадрі

Нижче зображено відеокадр (Рис. 2.26) та проекція ключових точок на нього (Рис. 2.27). На проекції можна побачити модель обличчя у вигляді білої сітки, накладену на відеокадр. Вона відображає усі точки, що виявляє бібліотека FaceMesh. Іншими кольорами зображено ключові точки, що використовуються нами під час проходження верифікації, а саме: брови (жовтий та блакитний), очі (червоний та зелений), ніс (фіолетовий) та рот (бірюзовий). Біла лінія з лівого краю обличчя до правого символізує з'єднання точок вух, що будуть використовуватися для визначення куту нахилу голови.



Рис. 2.26. Кадр з відеокамери

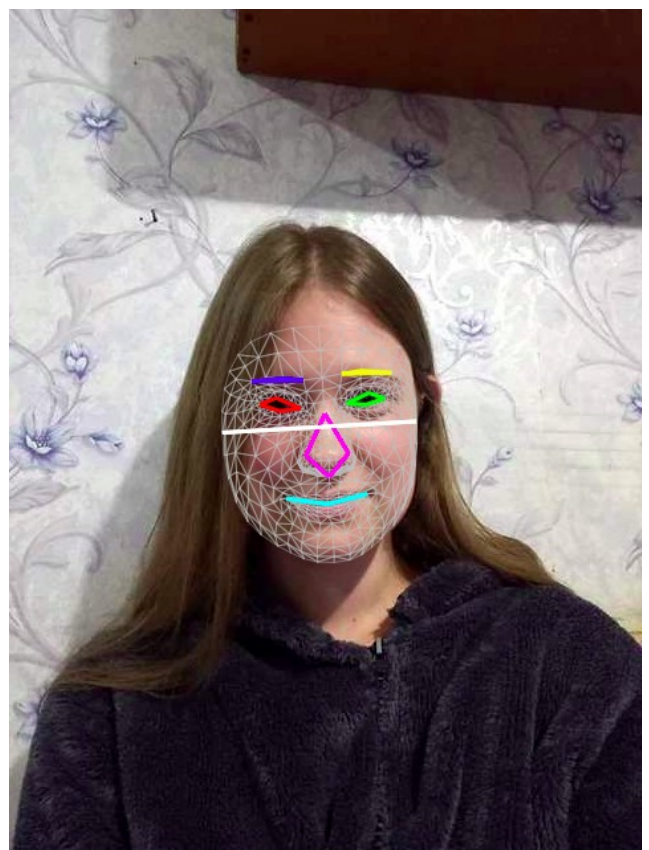


Рис. 2.27. Проекція ключових точок

2.2.4. Нормалізація ключових точок

На даному етапі ми отримали масив координат ключових точок, проте, використовувати такі координати для реальних розрахунків неможливо, адже

вони прямо залежать від положення обличчя людини у кадрі, відстані до камери (розміру обличчя), розширення матриці камери та інших факторів.

Для приведення координат до деякого стандартного вигляду, на основі якого можна робити обрахунки, необхідно виконати процес нормалізації координат. Під час нормалізації ми відсічемо пустий простір та відмасштабуємо ключові точки таким чином, щоб усі координати розташовувалися у межах від 0 до 1.

На Рис. 2.28 зображено код функції нормалізації, розглянемо його детальніше. Спочатку ми розраховуємо мінімальні та максимальні координати X, Y та Z для виявлення прямокутних границь (англ. bounding box) обличчя. Наступним кроком, ми виконуємо лінійне масштабування координат всередині границь обличчя за допомогою функції `mapValue` таким чином, щоб вони розташовувалися у межах від 0 до 1.

```
1 export const mapValue = (x: number, min: number, max: number,
2   l: number, h: number) => l + (h - l) * (x - min) / (max - min);
3 export const getNormalizedLandmarks = (landmarks: Landmarks) => {
4   let xMin = Infinity, yMin = Infinity, zMin = Infinity,
5     xMax = -Infinity, yMax = -Infinity, zMax = -Infinity;
6   const { min, max } = Math;
7   landmarks.forEach((point) => {
8     xMin = min(xMin, point.x); yMin = min(yMin, point.y);
9     zMin = min(zMin, point.z); xMax = max(xMax, point.x);
10    yMax = max(yMax, point.y); zMax = max(zMax, point.z);
11  });
12  const norm = (value: number) => mapValue(value, xMin, xMax, 0, 1);
13  const normalize = (point: Point) => ({
14    x: norm(point.x), y: norm(point.y), z: norm(point.z)
15  });
16  return landmarks.map(normalize);
17 }
```

Рис. 2.28. Код нормалізації ключових точок

Нижче, на Рис. 2.29 зображено візуалізацію цих нормалізованих ключових точок, що доступна у режимі відлагодження системи. Вона має квадратну форму та ній зображено вищеописані ключові точки у нормальній

системі координат від 0 до 1. Краї проекції займають точки вух, що зображені у вигляді чорних кружечків.

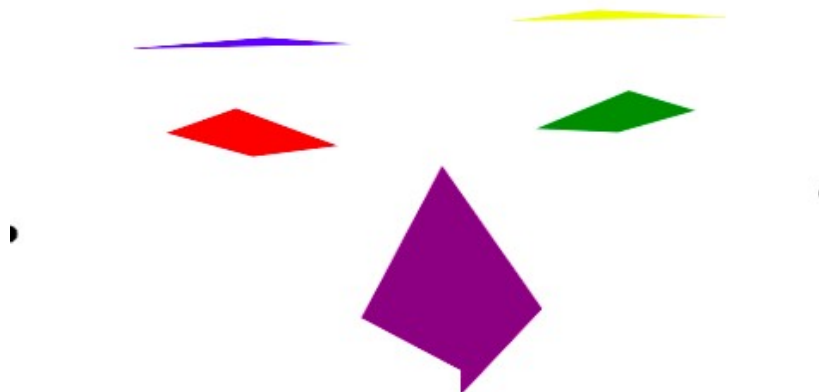


Рис. 2.29. Візуалізація нормалізованих ключових точок

Після виконання цих операцій, ми можемо бути впевнені, що координати точок будуть відносно однаковими (приймаємо до уваги похибку детекції точок) при будь-якому положенні обличчя в кадрі та на будь-якій адекватній відстані. Тепер ключові точки готові для виконання обчислень над ними.

2.2.5. Поворотна нормалізація

Хоча застосування нормалізацій координат надало нам можливість проводити стабільні розрахунки, не зважаючи на положення людини в кадрі, проте, для виконання деяких операцій, таких як порівняння усіх ключових точок, окрім лінійної координатної нормалізації, необхідна ще й нормалізація нахилу голови. Після її виконання, при будь-якому реальному нахилі голови,

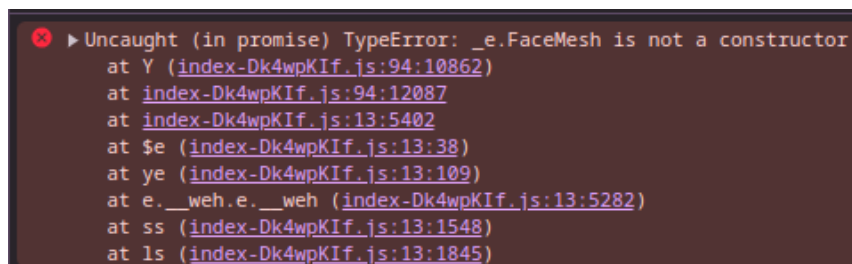
координати ключових точок будуть розташовані так, наче обличчя розташоване чітко навпроти камери.

Нормалізація повороту виконується у два основних етапи. На першому етапі ми розраховуємо нахил голови в усіх трьох осях (`yaw`, `pitch`, `roll`) (функція `getHeadRotation`). Після цього, ми виконуємо обертання всіх точок координат на кут, протилежний нахилу голови.

Повний код поворотної нормалізації зображено у додатку А. Він також застосовує функцію розрахунку положення голови, що описана у пункті 2.2.8.5. даного розділу та застосовується, також, для одного із завдань верифікації.

2.2.6. Виправлення помилки при збиранні FaceMesh

Під час локального запуску проекту за допомогою команди `yarn start`, все працює як очікувалося, проте, після збирання проекту за допомогою команди `yarn build` та подальшому запуску на реальному або тестовому сервері (команда `yarn preview`), в консолі можна побачити помилку про те, що функції з бібліотеки FaceMesh не знайдені. Текст помилки зображено на Рис. 2.30. Після проведення дослідження цієї проблеми, було виявлено, що бібліотека FaceMesh використовує застарілий формат експорту функцій з модуля, тому Vite під час збирання їх просто відкидає як непотрібні.



```
Uncaught (in promise) TypeError: _e.FaceMesh is not a constructor
    at Y (index-Dk4wpKIf.js:94:10862)
    at index-Dk4wpKIf.js:94:12087
    at index-Dk4wpKIf.js:13:5402
    at $e (index-Dk4wpKIf.js:13:38)
    at ye (index-Dk4wpKIf.js:13:109)
    at e.__weh.e.__weh (index-Dk4wpKIf.js:13:5282)
    at ss (index-Dk4wpKIf.js:13:1548)
    at ls (index-Dk4wpKIf.js:13:1845)
```

Рис. 2.30. Текст помилки, що потребує виправлення

Для виправлення цієї проблеми було вирішено створити плагін для збиральника Vite, що виконуватиме виправлення експортів бібліотеки FaceMesh під час збирання проекту. Код цього плагіну зображено на Рис. 2.31.

```

1 function mediaPipeExportsWorkaroundPlugin() {
2   return { name: "mediapipe_workaround",
3     load(id: string) {
4       if (path.basename(id) === "face_mesh.js") {
5         let code = readFileSync(id, "utf-8");
6         code += `exports.FaceMesh = FaceMesh;
7 exports.FACEMESH_TESSELATION = FACEMESH_TESSELATION;
8 exports.FACEMESH_RIGHT_EYE = FACEMESH_RIGHT_EYE;
9 exports.FACEMESH_RIGHT_IRIS = FACEMESH_RIGHT_IRIS;
10 exports.FACEMESH_RIGHT_EYEBROW = FACEMESH_RIGHT_EYEBROW;
11 exports.FACEMESH_LEFT_EYE = FACEMESH_LEFT_EYE;
12 exports.FACEMESH_LEFT_IRIS = FACEMESH_LEFT_IRIS;
13 exports.FACEMESH_LEFT_EYEBROW = FACEMESH_LEFT_EYEBROW;
14 exports.FACEMESH_FACE_OVAL = FACEMESH_FACE_OVAL;
15 exports.FACEMESH_LIPS = FACEMESH_LIPS;`;
16         return { code };
17       } else if (path.basename(id) === "camera_utils.js") {
18         let code = readFileSync(id, "utf-8");
19         return { code: code + "exports.Camera = Camera;" };
20       } else if (path.basename(id) === "drawing_utils.js") {
21         let code = readFileSync(id, "utf-8");
22         return { code: code + "exports.drawConnectors = drawConnectors;" };
23       } else { return null; } } };

```

Рис. 2.31. Vite плагін для виправлення експортів FaceMesh

2.2.7. Розробка архітектури завдань

Для спрощення розробки, тестування та використання завдань, було прийняте рішення побудувати процес з використанням абстрактних класів та патернів проектування Стратегія та Шаблонний метод.

Базовий клас завдання має наступний вигляд:

```

1 export abstract class BaseInstruction {
2   text: string = '';
3   abstract type: string;
4   abstract check(landmarks: NormalizedLandmarks): boolean;
5   screenshot(name: string) { /* ... */ };
6   $t(key: string, options: I$Options = {}) { /* ... */ };
7 }

```

Рис. 2.32. Базовий клас завдання

Окрім базової структури завдання, він також містить деякі методи, спільні для усіх завдань. Розглянемо нижче основні елементи.

Поле `text` містить поточний текст завдання, що буде виведений на екран засобами `Vue.js`. Зазвичай, воно використовується для виведення локалізованих інструкцій для користувача.

Поле `type` містить ідентифікатор типу завдання. Воно використовується для розпізнавання типу завдання після серіалізації у `JSON`.

Метод `check` є головним методом завдання. Він викликається кожен кадр та отримує на вхід поворотно-нормалізовані ключові точки обличчя. Якщо цей метод не повертає `true`, виконання завдання продовжується, якщо ж повертає — система вважає завдання виконаним та переходить до наступного чи завершує процес верифікації.

Методи `screenshot` та `$t` є утилітарними. Перший з них зберігає поточний кадр у пам'ять, а другий повертає локалізовану строку за її ідентифікатором.

Розглянемо приклад дочірнього класу завдання на прикладі завдання «Посміхнись» (Рис. 2.33).

```
export class SmileInstruction extends BaseInstruction {
  public type = "smile";
  private stage = 0;
  check(landmarks: NormalizedLandmarks) { this.applyStageText();
    if (this.stage === 0 && this.isNotSmiling(landmarks)) {
      this.stage++; this.screenshot("0-not-smiling");
    } else if (this.stage === 3) return true; // ...Кроки 1 та 2 скорочено
  }
  private getSmileFactor(lmk: NormalizedLandmarks) {
    const mouthEdge = Math.max(lmk.mouth.left.y, lmk.mouth.right.y);
    return (lmk.mouth.top.y - mouthEdge) * 1000;
  }
  isSmiling = (landmarks) => this.getSmileFactor(landmarks) > 24;
  isNotSmiling = (landmarks) => this.getSmileFactor(landmarks) < 12;
  private applyStageText() {
    if (this.stage === 0) this.text = this.$t("kyc.smile.no"); // ... скорочено
  }
}
```

Рис. 2.33. Клас завдання на прикладі завдання «Посміхнись»

У методі `check` відбувається весь процес перевірки завдання. Для зручності сприйняття коду, усі розрахунки були винесені в окремі методи із

зрозумілими назвами. Спочатку оновлюється поточний текст завдання. Наступним кроком йде сама логіка завдання. Завдання “Посміхнись” має три етапи: звичайне положення, посмішка, та знову звичайне положення. Коли виконуються умови переходу на наступний етап, відбувається інкремент поля `stage`. Коли усі етапи пройдені, метод повертає `true`, що показує менеджеру завдань перейти до наступного завдання.

Метод `getSmileFactor` розраховує деякий “коефіцієнт посмішки” на основі різниці підйому кутових точок рота відносно його верхньої точки. Виконання даного завдання передбачає пряме положення обличчя відносно камери, тому, окрім застосування поворотної нормалізації, задля підвищення точності розрахунків, ми тут використовуємо саме двовимірні розрахунки, замість тривимірних.

Далі окремими методами винесені `isSmiling` та `isNotSmiling`. Вони розраховують поточний коефіцієнт посмішки та порівнюють його з пороговим значенням `THRESHOLD` (у нашому випадку це значення 24).

Таке порогове значення було виявлено шляхом емпіричних досліджень на людях різних вікових категорій. Воно повинно бути достатньо великим для того, щоб не виявляти посмішку там, де її немає, але і достатньо малим для того, щоб людина могла фізично підняти кутки рота до такої міри.

Після ознайомлення з типовим класом завдання, розглянемо деякі основні моменти процесу, що керує цими класами, а саме, менеджменту завдань.

По-перше, розглянемо функцію `doInstructionCheck`, яку раніше згадували у пункті 2.3.3. на Рис. 2.24. Лістинг функції зображено на Рис. 2.34. Вона виконує дві основні задачі: по-перше, вона відображає на екрані текст поточного завдання (стрічки 8-10), а по-друге, вона перевіряє поточне завдання на успішність виконання і, у випадку його успішності, інкрементує показник на поточне завдання та, якщо завдання було останнім, відображає повідомлення

про успішне проходження верифікації, вивантажує модель з пам'яті та завершує процес верифікації.

```
1  const doInstructionCheck = (landmarks: NormalizedLandmarks) => {
2    if (instructions.currentIdx >= instructions.instructions.length) {
3      statusMessage.value = $t("kyc.common.success");
4      materials.takeScreenshot("finish");
5      unloadModel();
6      return emit("finish");
7    }
8    if (statusMessage.value !== instructions.currentInstruction.text) {
9      statusMessage.value = instructions.currentInstruction.text;
10   }
11   if (instructions.currentInstruction.check(landmarks)) {
12     instructions.currentIdx++;
13   }
14 };
```

Рис. 2.34. Відображення та перевірка поточного завдання

Тепер розглянемо змінну `instructions`, що використовувалась на Рис. 2.34. Вона являє собою `Pinia Store`, що відповідає за отримання завдань та ініціалізацію відповідних класів. На Рис. 2.35 зображено лістинг цього модуля.

```
export const useInstructionsStore = defineStore('instructions', () => {
  const instructions = ref<BaseInstruction[]>([]);
  const idx = ref(0);
  const currentInstruction = computed(() => instructions.value[idx.value]);
  const parseInstructions = (data: ISKycInstruction[]) => data.map(i => {
    switch (i.type) {
      case "blinking": return new BlinkingInstruction(i.props);
    } // ... Інші завдання опущені з метою компактності
  });
  const fetchInstructions = async () =>
    (instructions.value = parseInstructions(await KycApi.getInstructions()));
});
```

Рис. 2.35. Отримання та ініціалізація списку завдань

2.2.8. Розробка коду завдань

Код завдання “Посміхнись” ми вже розібрали у попередньому пункті, тепер розглянемо реалізацію інших завдань, опускаючи моменти, що повторюються.

2.2.8.1. Кліпання очима

Тут потрібно розрахувати коефіцієнт відкриття ока (EAR — Eye Aspect Ratio). Це робиться через ділення висоти на ширину. Код розрахункової функції зображено на Рис. 2.36.

```
1 export const getEAR = (lndmrks: NormalizedLandmarks, left = true) => {  
2   const eyeLeft = left ? lndmrks.eyeLeft.left : lndmrks.eyeRight.left;  
3   const eyeRight = left ? lndmrks.eyeLeft.right : lndmrks.eyeRight.right;  
4   const eyeTop = left ? lndmrks.eyeLeft.top : lndmrks.eyeRight.top;  
5   const eyeBottom = left ? lndmrks.eyeLeft.bottom : lndmrks.eyeRight.bottom;  
6  
7   return Math.abs(eyeTop.y - eyeBottom.y) / distance2D(eyeLeft, eyeRight);  
8 };
```

Рис. 2.36. Розрахунок коефіцієнту відкриття ока

Пороговим значенням відкритості ока емпіричним шляхом було обрано 0.02. Завдання має три етапи: відкрити обидва ока, закрити необхідне око, відкрити обидва ока. Сам процес розпізнавання відбувається настільки швидко, що для успішного виконання завдання, людині достатньо просто кліпнути потрібним оком.

Також, через те, що у людини, за замовчуванням, два ока, це завдання має три можливі варіанти кліпання: лівим оком, правим оком та обидвома очима. Варіант завдання зберігається у полі класу під назвою eye та ініціалізується у instructionsStore відповідно до отриманого списку завдань.

2.2.8.2. Відкриття та закриття рота

Це завдання по реалізації схоже на кліпання очима та посмішку. Необхідно розрахувати показник MAR (Mouth Aspect Ratio), що розраховується аналогічним чином (Рис. 2.37), проте, порівняння йде не з шириною рота, а з шириною всього обличчя, адже ширина рота у різних людей може доволі сильно варіюватися.

```

1 export const getMAR = (landmarks: NormalizedLandmarks) => {
2   const upperLip = landmarks.mouth.top;
3   const lowerLip = landmarks.mouth.bottom;
4
5   const earRight = landmarks.ear.right;
6   const earLeft = landmarks.ear.left;
7
8   return distance2D(upperLip, lowerLip) / distance2D(earRight, earLeft);
9 }

```

Рис. 2.37. Розрахунок коефіцієнта відкритості рота

Пороговим значенням було обрано 0.3.

2.2.8.3. Підняття та опускання брів

У цьому випадку розрахунки відрізняються, адже нам потрібно вирахувати звичайне положення брів для даного обличчя, а також їх підняте та насуплене положення. Спочатку планувалося створення декількох варіантів цього завдання, аналогічно до кліпання очима, проте, у ході дослідження виявилось, що лише невелика кількість людей може рухати бровами окремо одна від одної. Тому було прийняте рішення обмежитися завданням для обох брів відразу.

Розглянемо розрахунок коефіцієнту підняття брів на Рис. 2.38. Для цього розраховується відстань від брови до нижньої частини відповідного ока та ділиться на ширину цього самого ока.

```

1 private getBrowDistance(lmks: NormalizedLandmarks, left = true) {
2   const brow = left ? lmks.eyebrowLeft.center : lmks.eyebrowRight.center;
3   const eyeBottom = left ? lmks.eyeLeft.bottom : lmks.eyeRight.bottom;
4
5   const dist = distance2D(brow, eyeBottom);
6
7   const eyeLeft = left ? lmks.eyeLeft.left : lmks.eyeRight.left;
8   const eyeRight = left ? lmks.eyeLeft.right : lmks.eyeRight.right;
9
10  const eyeWidth = distance2D(eyeLeft, eyeRight);
11
12  return dist / eyeWidth;
13 }

```

Рис. 2.38. Розрахунок коефіцієнта підняття брів

На практиці виявилось, що це завдання є самими складним для виконання завданням, зокрема, насуплення брів. Адже, через використання косметичних засобів або генетичну схильність, деяким людям дуже важко сильно насуплювати брови. Емпіричним шляхом було встановлено порогове значення піднятих брів у значення 1.0, а насуплених — у 0.85.

2.2.8.4. Нахил голови

На відміну від минулих завдань, які вимагали розрахунку двовимірної та тривимірної відстані між ключовими точками, це завдання потребує розрахунку кутів.

У пункті 2.3.3. даної роботи згадувалося використання точок вух для виявлення нахилу голови. Тепер розглянемо це детальніше на Рис. 2.39. Спочатку знаходяться координати точок вух, потім розраховується арктангенс відношення координат цих точок, що дає кут в радіанах, а далі радіани конвертуються у градуси за відомою формулою перетворення.

```
1 export const getHeadRoll = (landmarks: NormalizedLandmarks) => {  
2   const left = landmarks.ear.left;  
3   const right = landmarks.ear.right;  
4   const angle = Math.atan2(left.y - right.y, left.x - right.x);  
5   return angle * (180 / Math.PI);  
6 }
```

Рис. 2.39. Розрахунок кута нахилу голови

Пороговими значеннями було обрано -15 градусів для нахилу вправо, 15 градусів для нахилу вліво та <10 градусів за модулем для прямого положення.

2.2.8.5. Поворот голови

Поворот голови є найскладнішим завданням з точки зору реалізації, адже він вимагає розрахунку кутів між точками у тривимірному просторі. Для початку розглянемо трохи теорії.

У тривимірному просторі орієнтацію твердого тіла можна однозначно описати за допомогою трьох кутів, відомих як кути Ейлера: pitch (тангаж), yaw

(рискання) та roll (крен). Вони широко використовуються в авіації, робототехніці, комп'ютерній графіці та інших галузях для опису орієнтації та руху об'єктів. Розглянемо їх детальніше.

Pitch - Кут повороту об'єкта навколо горизонтальної осі, перпендикулярної напрямку руху. Позитивний pitch відповідає опусканню "носа" об'єкта.

Yaw - Кут повороту об'єкта навколо вертикальної осі. Позитивний yaw відповідає повороту "носа" об'єкта вправо.

Roll - Кут повороту об'єкта навколо поздовжньої осі, що проходить вздовж напрямку руху. Позитивний roll відповідає нахилу "правого крила" об'єкта вниз.

Після ознайомлення з теорією, можна перейти до розглядання коду для розрахунку цих кутів, що зображений нижче, на Рис. 2.40. Спочатку отримуються координати необхідних ключових точок, далі розраховуються точки між очами та між вухами, що знадобляться для подальших розрахунків кутів, що застосовують Z-координату через порівняння з координатами кінчика носа. Опис розрахунків кожного з кутів зображено на таблиці 4.

Таблиця 4: Опис розрахунків кутів повороту голови

Кут	Опис розрахунку
Yaw	Обчислюється кут між лінією, що з'єднує центри вух та площиною XZ.
Pitch	Обчислюється кут між лінією, що з'єднує центр очей та кінчик носа, та площиною YZ. До результату додається 60° , для цілей калібрування. Таким чином нульовим кутом буде вважатися положення голови «чітко в камеру». Завдяки цьому, дуже легко виявити підняття/опускання голови — для цього достатньо подивитися на знак.
Roll	Обчислюється кут між лінією, що з'єднує центри вух та площиною XY.

```

1 private getHeadRotation(landmarks: NormalizedLandmarks) {
2     const leftEyeOuter = landmarks.eyeLeft.left;
3     const rightEyeOuter = landmarks.eyeRight.right;
4     const noseTip = landmarks.nose.tip;
5     const earLeft = landmarks.ear.left;
6     const earRight = landmarks.ear.right;
7
8     const eyeCenter = {
9         x: (leftEyeOuter.x + rightEyeOuter.x) / 2,
10        y: (leftEyeOuter.y + rightEyeOuter.y) / 2,
11        z: (leftEyeOuter.z + rightEyeOuter.z) / 2
12    };
13
14    const yaw = -angle3D(earRight, earLeft).xz;
15    const pitch = -(angle3D(eyeCenter, noseTip).yz + 60);
16    const roll = -angle3D(earRight, earLeft).xy;
17    return { yaw, pitch, roll };
18 }

```

Рис. 2.40. Розрахунок кутів нахилу голови

Обрані порогові значення зображені на таблиці 5. Можна помітити, що для опускання вниз обрано менший кут, ніж для підняття вверх. Це пов'язано з анатомічними особливостями людини, через які деяким користувачам значно складніше опустити голову, ніж її підняти. Для розрахунку звичайного положення береться значення кутів по модулю.

Таблиця 5: Порогові значення для положень повороту голови

Положення голови	Значення
Звичайне положення	$ yaw < 10 \ \& \ pitch < 10$
Поворот вправо	$yaw > 30$
Поворот вліво	$yaw < -30$
Підняття вверх	$pitch > 25$
Опускання вниз	$pitch < -20$

Як можна побачити, показник roll можна було використати у завданні “Нахил голови”, проте воно було додано задовго до повороту, тому має свою власну реалізацію. У майбутньому допускається об’єднання цих двох завдань, в такому випадку нахил голови стане одним із варіантів повороту.

2.2.9. Розпізнавання заміни людини

Ще однією проблемою під час процесу автоматизованої верифікації є відсутність розпізнавання обличчя. На будь-якому етапі телефон може взяти інша людина і система ніяк цього не виявить.

2.2.9.1. Розпізнавання обличчя

Зазвичай, завдання розпізнавання обличчя виконується потужними моделями комп'ютерного зору. Нижче, у пункті 2.2.11. описано використання моделі FaceNet, у результаті роботи якої отримується дескриптор[3] із 512 елементів, що дозволяє доволі точно (Рис. 2.51) розпізнавати та порівнювати обличчя людей. Ця модель виконується на сервері та виконує порівняння фотографій у профілі користувача з фотографіями, отриманими під час процесу верифікації.

Проте, для виявлення заміни людини в клієнтському додатку, було вирішено додати простий алгоритм розпізнавання та порівняння облич, який використовує базові можливості FaceMesh. Цей алгоритм працює в реальному часі та підходить навіть для мобільних пристроїв.

Алгоритм складається з двох основних етапів. На першому етапі розраховується дескриптор обличчя перед виконанням завдань. Далі, розраховується дескриптор для кожного кадру та порівнюється з референсним за допомогою розрахунку евклідової відстані між векторами. Якщо відстань менша за порогову величину, — вважається, що на кадрі та сама людина, якщо ж більша — виконання завдання припиняється.

На відміну від вищезгаданої тези[3], де використовувалася модель, розроблена спеціально для розрахунку дескриптору обличчя, в нашому випадку використовується модель, націлена на трекінг точок обличчя, тому і дескриптор нам прийдеться формувати самостійно. Для цього візьмемо основні величини, за допомогою яких можна розрізняти людей між собою: ширина очей, відстань

між очима, ширина голови, ширина носа, відстань між носом та очима, відстань між носом та ротом, та відстань від вух до очей. Емпіричні дослідження показали високу ефективність обраних факторів, тому сукупність цих величин ми будемо вважати дескриптором обличчя. Для розрахунку відстані ми будемо використовувати суму квадратів різниці величин. Шляхом емпіричних досліджень, пороговим значенням було обрано число 0.001. Код розрахунку дескриптора обличчя зображено на Рис. 2.41.

```
export const calculateFaceDescriptor = (landmarks: Landmarks): number[] => {
  const norm = getNormalizedLandmarks(getRotationNormalizedLandmarks(landmarks));
  const eyeLeftWidth = distance3D(norm.eyeLeft.left, norm.eyeLeft.right);
  const eyeRightWidth = distance3D(norm.eyeRight.left, norm.eyeRight.right);
  const noseWidth = distance3D(norm.nose.left, norm.nose.right);
  const headWidth = distance3D(norm.ear.left, norm.ear.right);
  const browLeftWidth = distance3D(norm.eyebrowLeft.left, norm.eyebrowLeft.right);
  const browRightWidth = distance3D(norm.eyebrowRight.left, norm.eyebrowRight.right);
  const distanceBetweenEyes = distance3D(norm.eyeLeft.right, norm.eyeRight.left);
  const distanceFromNoseToLeftEye = distance3D(norm.eyeLeft.right, norm.nose.tip);
  const distanceFromNoseToRightEye = distance3D(norm.eyeRight.left, norm.nose.tip);
  const distanceFromEarToLeftEye = distance3D(norm.eyeLeft.left, norm.ear.left);
  const distanceFromEarToRightEye = distance3D(norm.eyeRight.right, norm.ear.right);
  const distanceFromNoseToMouth = distance3D(norm.mouth.top, norm.nose.tip);

  return [
    eyeLeftWidth, eyeRightWidth, noseWidth, headWidth, browLeftWidth,
    browRightWidth, distanceBetweenEyes, distanceFromNoseToLeftEye,
    distanceFromNoseToRightEye, distanceFromEarToLeftEye,
    distanceFromEarToRightEye, distanceFromNoseToMouth
  ];
}

export const compareFaceDescriptorsSquared = (descriptor1, descriptor2) => descriptor1
  .map((value, i) => Math.pow(value - descriptor2[i], 2))
  .reduce((acc, value) => acc + value, 0);
```

Рис. 2.41. Розрахунок та порівняння дескрипторів обличчя

Також, слід зазначити, що розрахунок дескрипторів виконується на основі ключових точок, що пройшли через поворотну нормалізацію, детально описану у пункті 2.2.5. даної роботи.

2.2.9.2. Компенсація неточностей ключових точок

На жаль, точність ключових точок, що надає бібліотека FaceMesh не дозволяє проводити реальне розпізнавання людини за обличчям, адже навіть при невеликому повороті голови, накопичені помилки призводять до того, що розрахований дескриптор настільки відрізняється від базового, ніби це зовсім інша людина.

Для компенсації цієї проблеми, було вирішено циклічно збирати дескриптори за деякий проміжок часу, порівнювати їх усіх з базовим дескриптором та обирати варіант з найкращим порівнянням (найменше середньо-квадратичне відхилення). Для цього ми зберігаємо у пам'яті деяку статичну кількість дескрипторів у вигляді масиву та маємо лічильник, що циклічно зростає та скидається до нуля коли доходить до кінця масиву. У кожен момент часу розраховується новий дескриптор та зберігається у поточну комірку буферу, на яку вказує лічильник. Таким чином, вдається досягти поступового оновлення даних та уникнути проблем, які виникають через неточності обчислення ключових точок.

Шляхом емпіричних дослідів, було виявлено, що розміру буферу у 50 дескрипторів достатньо для того, щоб успішно пройти інструкції, які потребують значної зміни положення голови (поворот та нахил), але недостатньо для того, щоб підмінити людину під час проходження перевірки.

Якщо у момент часу мінімальне відхилення перевищує встановлене значення (на основі експериментів, таким значенням встановлено число 0.001), виконання завдання припиняється та користувачу показується попередження про те, що його неможливо впізнати. Як тільки проблема виправляється, виконання завдання продовжується з того ж місця, на якому було припинено. Експерименти довели успішність такого способу інформування.

2.2.10. Виявлення облич на фотографії (MTCNN)

Після того, як ми завершили клієнтську частину процесу верифікації та маємо фотографії користувача, отримані під час виконання завдань верифікації, необхідно перевірити, чи дійсно вони відповідають фотографіям у профілі користувача.

Першим кроком на шляху порівняння обличчя є його виявлення. Одним з перших та найбільш відомих методів для виявлення облич у цифрових зображеннях є алгоритм Віюлі-Джонса, описаний у їх роботі 2001 року “Rapid object detection using a boosted cascade of simple features”[31]. Він базується на використанні ансамблю ознаки Хаара та методу каскадних класифікаторів для швидкого та ефективного виявлення облич[32].

Ознаки Хаара використовуються для опису різних частин обличчя шляхом визначення яскравості зображення в різних областях, вони описують такі особливості, як краї, границі та зміни інтенсивності в різних частинах зображення. Метод каскадних класифікаторів використовує набір класифікаторів, які навчаються визначати, чи є певна область зображення обличчям чи ні (рис. 2.42).

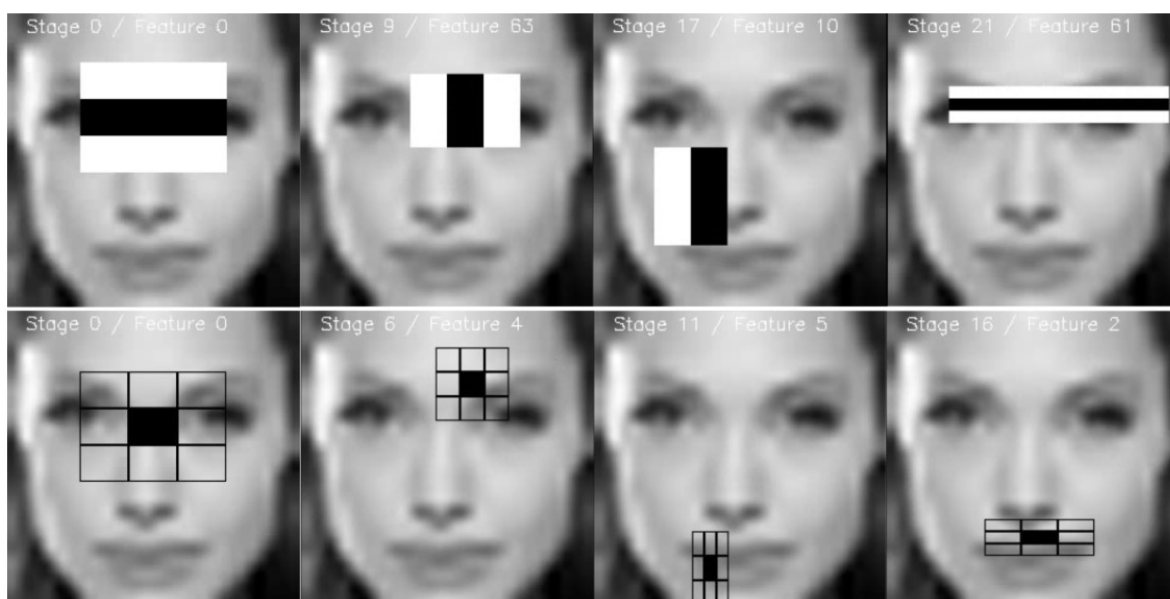


Рис. 2.42. Демонстрація роботи класифікатора

Каскадний підхід полягає в тому, що на кожному етапі зображення проходить через послідовні класифікатори, і якщо зображення не відповідає критеріям класифікації на будь-якому етапі, воно відкидається, що дозволяє швидко визначати зображення, що не містять обличчя. Алгоритм Віоли-Джонса виявляє обличчя на зображенні шляхом порівняння різних частин зображення з попередньо навченими шаблонами облич.

Цей метод відомий своєю швидкістю та простотою реалізації, а також здатністю працювати у реальному часі. Однак, в порівнянні з більш сучасними методами, такими як HOG та MTCNN, алгоритм Віоли-Джонса може видавати більше хибно-позитивних результатів та менш точно виявляти обличчя під незвичайними кутами та в ускладнених умовах, демонструючи максимальну ефективність при роботі з фронтальними зображеннями обличчя при гарному освітленні.

Більш сучасним способом визначення обличчя на фотографії є метод гістограми направлених градієнтів, або HOG (Histogram of Oriented Gradients). Він є потужним знаряддям для опису особливостей об'єктів у зображеннях. Своєю історією він зобов'язаний роботам В. Triggs та N. Dalal в 2005 році[33]. Вони розробили цей метод для вирішення завдання виявлення пішоходів на зображеннях з використанням класифікатора на основі метода опорних векторів - SVM (Support Vector Machine).

Основна перевага методу HOG полягає в його стійкості до змін у розміщенні об'єктів, оскільки форма об'єкта характеризується розподілом локальних градієнтів і напрямком країв. На першому кроці зображення розділяється на невеликі комірки, щоб отримати локальні області для аналізу. Далі виконується обчислення гістограм орієнтованих градієнтів (HOG), що відображає розподіл градієнтів в даній області. На останньому кроці усі гістограми комірок об'єднуються в один вектор ознак, який стає унікальним для кожного обличчя. Це дозволяє не тільки визначити обличчя на фотографії, а ще

й отримати вектор ознак, що можна використати у майбутньому для порівняння облич на різних фотографіях.

Однак метод HOG також має деякі обмеження. Він не працює з обличчями під незвичайними кутами. Цей метод, також, як і метод Віолі-Джонса, використовується, зазвичай, для виявлення облич у сканованих документах, таких як водійські посвідчення та паспорти, але не дуже підходить для сервісу знайомств, адже фотографії користувачів часто більш складні.

Тому для виявлення обличчя на фотографіях ми будемо використовувати сучасний метод MTCNN (Multi-Task Cascaded Convolutional Networks), що являє собою комплексну систему для виявлення облич у реальному часі, яка використовує згорткові нейронні мережі[10]. Цей метод відомий своєю високою точністю та ефективністю навіть у складних умовах. У порівнянні з HOG, метод MTCNN забезпечує низький рівень хибно-позитивних результатів та високу точність виявлення облич, зокрема, під різними кутами та у різних умовах освітлення.

Цей метод складається з трьох послідовних моделей: моделі пропозицій [англ. Proposal] (P-Net, рис. 2.43), моделі уточнення [англ. Refine] (R-Net, рис. 2.44) та вихідної моделі [англ. Output] (O-Net, рис. 2.45)[34].

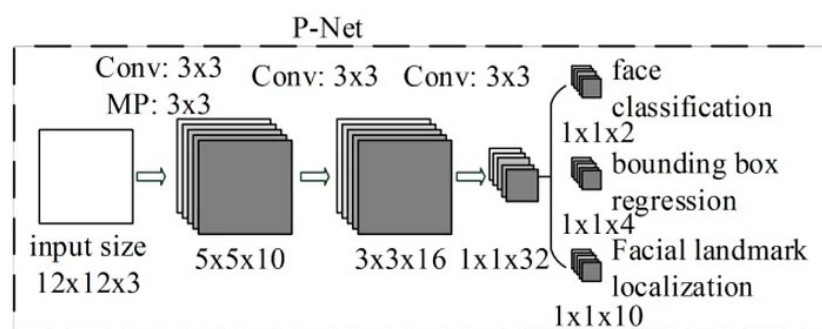


Рис. 2.43. Модель пропозицій (P-Net)

На першому етапі зображення проходить через повно-згорткову нейронну мережу (FCN). Вона використовується для отримання координат прямокутників, у яких знаходиться потенційне обличчя, а також, їх вектори

регресії. Після цього проводиться корекція для об'єднання областей, що перекриваються, в одну більшу область.

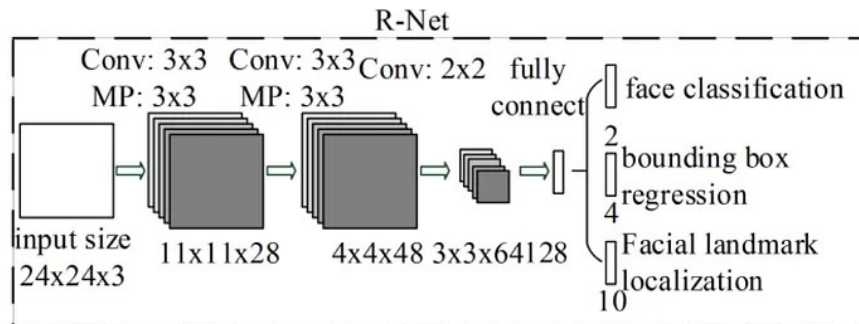


Рис. 2.44. Модель уточнення (R-Net)

Далі усі кандидати з мережі P-Net подаються на вхід мережі R-Net. Вона використовується для подальшого зменшення кількості кандидатів, виконання калібрування з використанням регресії отриманих областей та застосування методу неперевикнення максимуму (NMS)[35] для об'єднання кандидатів, що перекриваються.

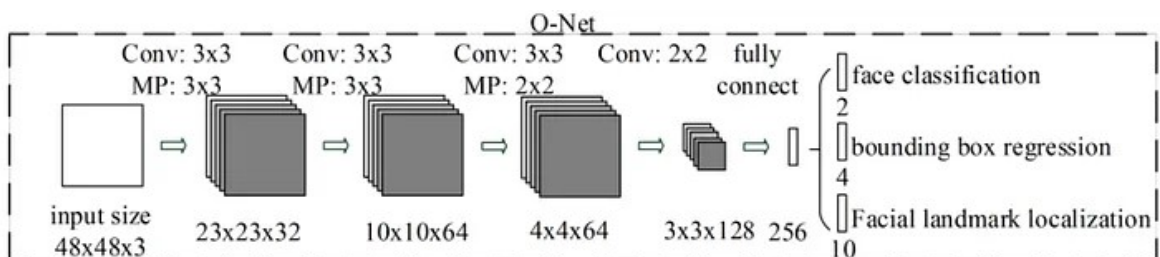


Рис. 2.45. Вихідна модель (O-Net)

Далі йде модель O-Net. Цей етап схожий на R-Net, але вихідна мережа ставить за мету детальніше описати обличчя та вивести позиції п'яти ключових точок обличчя (facial landmarks): очі, ніс та рот. Це ті ж самі ключові точки можуть, що були використані для розрахунку положення голови, кліпання очима та виявлення інших на етапі верифікації.

Усі згадані методи видають на виході координати обличчя на фотографії у вигляді прямокутника. Це можна використовувати для відлагодження у вигляді малювання прямокутників навколо облич для виявлення хибних спрацювань, а також для автоматичного кадрування для наступної його

передачі до інших алгоритмів, що натреновані працювати з зображеннями обличчя. У випадку виявлення декількох облич на зображенні, ми будемо вважати основним найбільше з них.

В нашому випадку, кадровані фотографії використовуватимуться для розпізнавання обличчя та емоцій, що будуть описані у наступних пунктах даної роботи. Крім того, використання кадрованих фотографій для подальшої обробки вигідно заощадженням місця на диску та пропускну здатності мережі, адже обличчя, зазвичай, займає лише невелику частину зображення.

Перейдемо до практичної реалізації. Для початку необхідно встановити бібліотеки `torch`, `facenet-pytorch`, та `pillow`. На Рис. 2.46 зображено ті версії, що використовувалися у даній роботі.

```
!pip install torch facenet-pytorch==2.5.3 pillow==10.3.0
```

Рис. 2.46. Встановлення необхідних бібліотек

Після цього, імпортуємо ці бібліотеки та проведемо ініціалізацію моделі MTCNN, як показано на Рис. 2.47. Параметр `min_face_size` вказує на мінімальний допустимий розмір обличчя у пікселях. В нашому випадку, робота із обличчями менше 80x80px є недоцільною, адже точність ключових точок у них буде замалою для розрахунків.

```
from facenet_pytorch import MTCNN
import torch
import torchvision.transforms as transforms
from PIL import Image

mtcnn = MTCNN(min_face_size=80)
```

Рис. 2.47. Ініціалізація MTCNN

Наступним кроком оголошуємо функції (Рис. 2.48) для виявлення головного обличчя на фотографії та її подальшого кадрювання. Функції приймають на вхід зображення у форматі бібліотеки `Pillow` та коефіцієнт мінімально необхідної впевненості мережі. Далі у функції `getFaces` отримуємо список межових координат (`bounding box`) облич та впевненості мережі у тому,

що це саме обличчя. Після цього фільтруємо цей список, щоб залишити тільки координати з впевненістю вище 90% (можна налаштувати за допомогою аргументу `min_probability`) та повертаємо перше (саме велике). Схожий механізм було використано у бібліотеці FaceMesh у пункті 2.2.3 даної роботи на Рис. 2.24.

Після цього виконуємо кадрування за допомогою бібліотеки Pillow у функції `cropFace`.

```
def getFaceBox(img, min_probability=0.9):
    boxes, probs = mtcnn.detect(img)
    if boxes is None: return None

    boxes = boxes[probs >= min_probability]

    return boxes[0]

def cropFace(img, min_probability=0.9):
    box = getFaceBox(img, min_probability)
    if box is None: return None

    return img.crop(box)
```

Рис. 2.48. Виявлення головного обличчя та кадрування фотографії

Окрім застосування з метою кадрування обличчя для подальшої обробки та класифікації, ми будемо використовувати для виявлення кількості людей на фотографії за кількістю знайдених облич. Ця інформація буде використовуватися під час аналізу ефективності профілю, що буде описано в пункті 3.4.2. даного розділу.

Напишемо функцію для виявлення кількості облич на фото (Рис. 2.49). Вона повністю аналогічна функції `getFaceBox`, але замість повернення координат найбільшого обличчя, вона просто рахує кількість знайдених облич за обраною вірогідністю.

```
def countFaces(img, min_probability=0.9):
    boxes, probs = mtcnn.detect(img)
    if boxes is None: return 0
    boxes = boxes[probs >= min_probability]
    return len(boxes)
```

Рис. 2.49. Виявлення кількості облич на фото

2.2.11. Методи порівняння обличчя на фотографіях (FaceNet)

Після того, як, в наслідок використання MTCNN, ми отримали кадрове зображення обличчя, можна перейти до його розпізнавання. Цей процес схожий з тим, що описаний у пункті 2.2.9.1. , проте використовує значно більш точну модель.

Для порівняння обличчя на різних фотографіях необхідно виконати 4 основні етапи:

1. Кадрувати усі використовувані фотографії за обличчям.
2. Дістати вектор ознак (рис. 2.50) обличчя з референсної фотографії[36].
3. Дістати вектор ознак обличчя з порівнюваної фотографії[37].
4. Розрахувати відсоток співпадіння цих параметрів (евклідова відстань, косинусна відстань або відстань Махаланобіса між векторами ознак)[38].

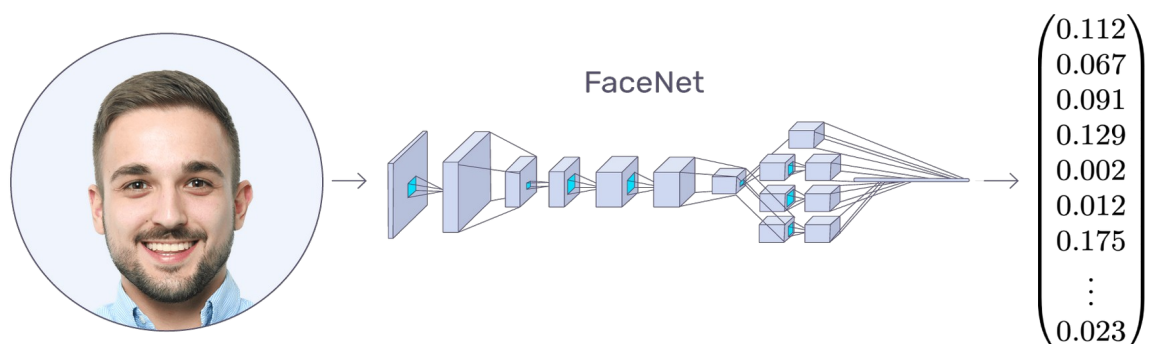


Рис. 2.50. Представлення перетворення зображення на вектор ознак

Для діставання параметрів обличчя використовуються згорткові нейронні мережі. У статті “Methods of face comparison in different photos using computer vision” (2024) описано розв’язання цієї задачі за допомогою нейронної мережі Resnet[39] та бібліотеки dlib[40] (2009), що дають точність близько 99.38% та мають проблеми з розпізнаванням облич під незвичайним кутом або при поганому освітлені. Проте, існують більш сучасні моделі, такі як FaceNet (2015 рік, 99.63% точності) та ArcFace + MS1MV2 + R100 (2018 рік, 99.83% точності) [41] що виявляють дескриптори обличчя більш точно та ефективно (рис. 2.51).

Rank	Model	Accuracy↑	Extra Training Data	Paper	Code	Result	Year	T
1	ArcFace + MS1MV2 + R100,	99.83%	✓	ArcFace: Additive Angular Margin Loss for Deep Face Recognition	Code	Result	2018	
2	FaceNet	99.63%	✓	FaceNet: A Unified Embedding for Face Recognition and Clustering	Code	Result	2015	
3	Dlib	99.38%	✓	Dlib-ml: A Machine Learning Toolkit	Code	Result	2009	
4	VGG-Face	98.78%	✓	Deep Face Recognition	Code	Result	2015	
5	DeepFace	98.37%	×	DeepFace: Closing the Gap to Human-Level Performance in Face Verification	Code	Result	2014	
6	DeepID	97.05%	×	Deep Learning Face Representation from Predicting 10,000 Classes	Code	Result	2014	
7	OpenFace	92.92%	×	OpenFace: A general-purpose face recognition library with mobile applications	Code	Result	2016	

Рис. 2.51. Порівняння моделей розпізнавання обличчя

В рамках даної кваліфікаційної роботи ми будемо використовувати мережу FaceNet, навчену на датасеті VGGFace2 (2018 рік, 99.65% точність)[42], адже вона максимально поєднує у собі швидкодію, розмір моделі та точність розпізнавання. Розглянемо її більш детально.

FaceNet - це нейронна мережа, призначена для вирішення завдання розпізнавання облич у зображеннях та отримання унікального векторного дескриптора обличчя. Основна ідея проекту FaceNet полягає у тому, щоб отримати таких векторів, що будуть мати малу відстань[38] якщо на фотографіях зображено обличчя однієї і тої самої людини, та велику відстань, якщо обличчя належать різним людям (рис. 2.52).

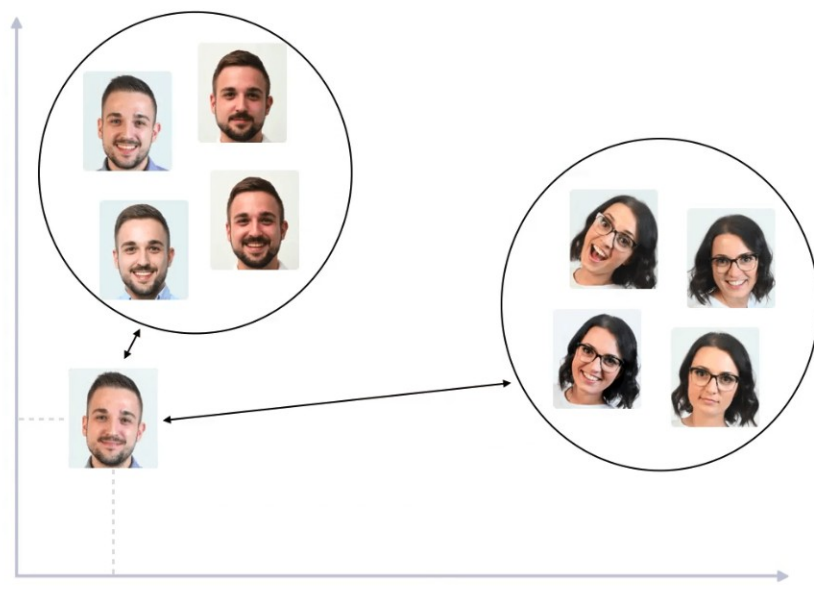


Рис. 2.52. Графічне зображення відстані між векторами дескрипторів

FaceNet використовує глибоку згорткову нейронну мережу (CNN), яка приймає на вхід зображення обличчя та видає вектор ознак (дескриптор), що представляє це обличчя у векторному просторі[43].

Окрім застосування після проходження верифікації, усі описані тут етапи розпізнавання зображення виконуватимуться у рамках одного конвеєру після того, як користувач завантажить нову фотографію в систему, тому деякі етапи, такі як виявлення та кадрування обличчя, можна перевикористовувати на наступних етапах конвеєру.

З точки зору коду, більшість операцій приховано всередині бібліотеки `facenet_pytorch`, тому нам доведеться лише робити високорівневі виклики:

```
from facenet_pytorch import InceptionResnetV1
import torch
import torchvision.transforms as transforms
from PIL import Image

resnet = InceptionResnetV1(pretrained='vggface2').eval()
```

Рис. 2.53. Ініціалізація моделі

Для отримання дескриптора обличчя, виконаємо кадрування за допомогою мережі MTCNN, та передамо результат у мережу FaceNet.

```
def getDescriptor(img):  
    descriptor = resnet(mtcnn(img).unsqueeze(0))  
    return descriptor.detach().numpy().flatten().tolist()
```

Рис. 2.54. Отримання дескриптора обличчя

Для порівняння дескрипторів використовується функція, зображена на Рис. 2.55. Якщо результат менше 0.5, можна вважати, що на фотографіях зображено одну і ту саму людину.

```
function compareDescriptors(desc1: number[], desc2: number[]) {  
    const sum = desc1.reduce((acc, cur, i) => acc + (cur - desc2[i]) ** 2, 0);  
    return Math.sqrt(sum);  
}
```

Рис. 2.55. Порівняння дескрипторів

2.3. Генератор першого повідомлення

Для реалізації поставленого завдання було обрано набір інструментів, що забезпечують ефективне навчання, оптимізацію та використання нейронних мереж. Основним фреймворком для роботи з нейронними мережами обрано PyTorch, який є популярним і потужним інструментом для створення та навчання моделей машинного навчання.

Для доступу до попередньо навчених моделей та їх використання у дослідженні застосовано платформу Hugging Face. Це репозиторій з великою кількістю готових моделей, які можна адаптувати під конкретні задачі. У поєднанні з бібліотекою Transformers, яка також є частиною екосистеми Hugging Face, цей інструмент дозволяє легко завантажувати, тренувати та використовувати різні типи моделей, зокрема трансформери.

Для збереження тренованих моделей було обрано формат Safetensors[44], який є оптимізованим і безпечним способом зберігання даних. Його використання забезпечує більш ефективну та безпечну передачу готових моделей у різних середовищах.

2.3.1. Вибір базової моделі нейронної мережі

Вибір базової моделі нейронної мережі є важливим етапом, що визначає якість, швидкість та ефективність кінцевого рішення. Для створення моделі, здатної генерувати тексти на платформі знайомств та пошуку друзів, було розглянуто кілька кандидатів. Ключовими факторами вибору стали вимоги до продуктивності, можливості запуску на обмежених ресурсах (зокрема, на процесорі) та здатність до до-навчання (fine-tuning).

Спочатку були розглянуті сучасні трансформерні моделі, такі як LLaMA, Qwen та Mistral, які демонструють відмінні результати у задачах генерації тексту. Проте, навіть зменшені та квантизовані версії цих моделей виявилися занадто ресурсозатратними для використання на процесорах. Тому було вирішено розглянути більш легких варіантів моделей, таких як BERT (DistilBERT)[45], GPT-2 (DistilGPT-2) та GPT-3 (GPT-Neo), відомих своєю ефективністю при роботі на CPU.

Після невеликого порівняльного дослідження, у якості основи було обрано модель DistilGPT-2, яка є скороченою версією моделі GPT-2, утвореною методом knowledge distillation[46]. DistilGPT-2 підтримує більшість функцій повнорозмірної GPT-2, але має менше параметрів, що знижує використання ресурсів і підвищує швидкість генерації тексту. Це дозволило досягти прийнятної якості генерації при обмежених обчислювальних потужностях.

Однак, у процесі початкових тестів було виявлено, що кількість токенів у запиті після токенізації перевищує 1024, що є лімітом вікна контексту DistilGPT-2. Це стало проблемою під час обробки великих запитів, таких як профілі користувачів із великою кількістю інтересів. Тому було вирішено використати модель GPT-Neo[47] 125M, що є Open Source альтернативою архітектурі GPT-3[48] від OpenAI та має ширше вікно контексту (2048 токенів) при порівняній швидкості роботи.

2.3.2. Формування датасету

Для успішного навчання моделі, яка зможе генерувати релевантні та привабливі повідомлення для платформи знайомств і пошуку друзів, критично важливим є створення якісних датасетів. Датасети повинні містити приклади текстів, що відображають бажану поведінку моделі різними мовами та в різних контекстах, зокрема романтичному та дружньому. Для формування датасету було визначено кілька ключових вимог, зображених на таблиці 6.

Таблиця 6: Вимоги до датасету

Вимога	Пояснення
Двомовність	Модель повинна генерувати повідомлення як англійською, так і українською мовами, залежно від запиту.
Контекстуальність	Повідомлення повинні бути адаптовані під два основні контексти — романтичний та дружній.
Природність	Повідомлення повинні бути креативними та природними, а також стимулювати до продовження розмови, уникаючи банальних фраз.

Формування датасетів планувалося із залученням потужних великих мовних моделей (LLM), таких як ChatGPT, Gemini та Microsoft Copilot, для генерації зразків ідеальних перших повідомлень. Це дало змогу значно прискорити процес підготовки даних, оскільки не було потреби збирати приклади вручну.

Шляхом емпіричних досліджень було обрано моделі Gemini та Gemini Advanced, адже вони генерували найбільш цікаві та креативні повідомлення, при цьому маючи найбільш привабливі денні ліміти на запити.

Запити для генерації повідомлень формувалися за чіткою структурою, що включала: мову (англійську або українську); контекст (романтичний або дружній); системний запит, що давав моделі інструкції щодо генерації повідомлення; та інформацію про профілі користувачів у вигляді JSON-

формату, що дозволяло моделі створювати персоналізовані повідомлення на основі конкретних характеристик профілю.

У якості запиту до моделі було використано наступну XML структуру:

```
<lang>Мова (en/uk)</lang>  
<context>Контекст (romantic/friendship)</context>  
<system>Текст запиту</system>  
<user>Профіль людини, до якої пишеться привітання у JSON вигляді</user>  
<me>Профіль людини, від імені якої пишеться привітання у JSON вигляді</me>
```

Рис. 2.56: Структура тексту запиту до моделі

При цьому, запит всередині тегів system використовується лише для вказання референтним моделям, яким чином формувати відповідь. Під час навчання моделі, це поле не використовується.

У додатку Б зображено тексти системних запитів, що використовувалися для генерації референтних повідомлень.

У результаті було створено чотири окремі датасети для кожної комбінації мови та контексту:

Мова	Контекст	Кількість повідомлень
Українська	Романтичний	9702
Українська	Дружній	3257
Англійська	Романтичний	2530
Англійська	Дружній	1624
Всього		17113

Рис. 2.57. Склад датасету

Загалом було сформовано датасет із 17113 повідомлень. Такий обсяг даних забезпечує різноманіття прикладів, необхідне для якісного навчання моделі та її адаптації до різних типів користувачів та ситуацій.

Їх розподілення зображено на Рис. 2.58. Як бачимо, більше всього прикладів відведено українськомовним повідомленням, адже моделі попередньо навчені на англійських текстах, що робить для них складнішим розуміння української мови. На романтичний контекст було також відведено набагато

більше прикладів, адже флірт вимагає більш тонкого підходу до формулювання фраз. Це особливо важливо для уникнення шаблонних або незграбних формулювань, які можуть відлякати потенційного співрозмовника. Романтичний контекст потребує більшої кількості варіантів, щоб охопити різні стилі комунікації, емоційні відтінки та підходи до вираження інтересу, тоді як дружній контекст є більш універсальним і не вимагає такої високої точності та делікатності у виборі слів.

Крім того, в публікації «The Role of Language Imbalance in Cross-lingual Generalisation: Insights from Cloned Language Experiments» (2024p)[49] розповідається про дослідження, що задля досягнення багатомовності, більшість даних у датасеті повинна бути «домінуючою» мовою. В нашому випадку це саме українська мова.

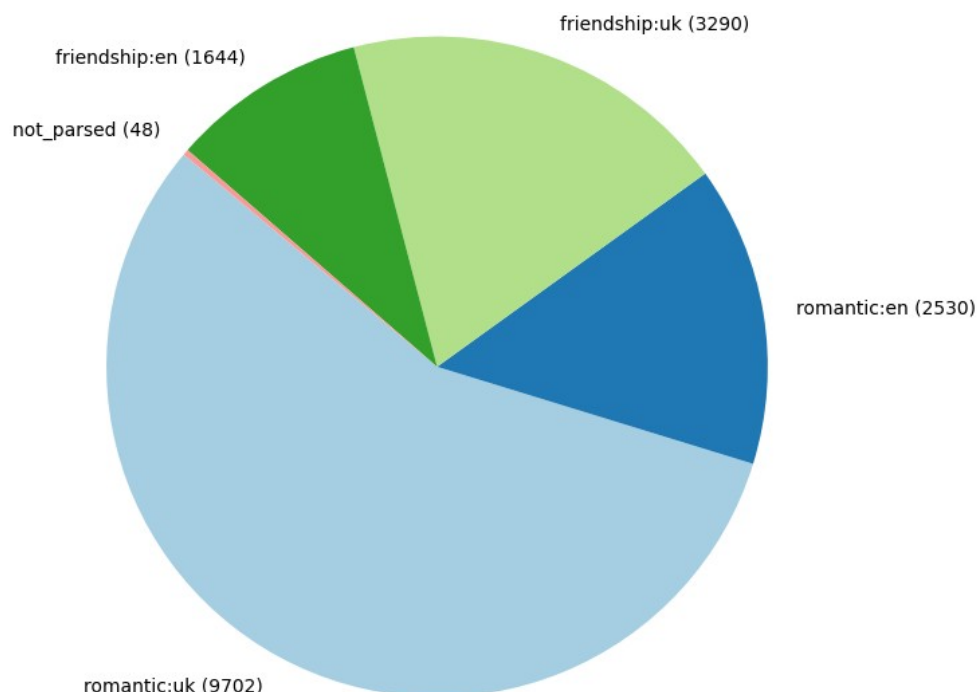


Рис. 2.58. Кількість повідомлень кожного типу в датасеті

2.3.3. Визначення зон відповідальності моделей

Під час розробки моделі для генерації перших повідомлень на платформі для знайомств та пошуку друзів важливо було визначити оптимальну стратегію розподілу задач між моделями, щоб забезпечити точність і ефективність

генерації повідомлень. Враховуючи двомовність та різні контексти — романтичний і дружній — розглядалося кілька підходів до організації тренувальних даних та моделей.

Один із підходів полягав у використанні однієї універсальної моделі для обробки всіх запитів незалежно від мови чи контексту. Така модель мала б компактний розмір і не займала б багато оперативної пам'яті під час використання. Вона могла би впоратися з різними запитами завдяки вказівкам щодо мови та контексту у тексті запиту. Однак, універсальна модель може мати меншу точність, оскільки у ній змішуються мовні особливості, що потенційно може призвести до помилок, таких як випадкове використання україно-англійського суржику.

Іншим підходом було створення окремих моделей для кожної мови. Такий варіант давав би змогу повністю уникнути мовних змішувань і забезпечував би чіткішу роботу з текстами англійською та українською мовами. Проте, це рішення значно збільшувало б кількість необхідних ресурсів, адже на сервері довелося б одночасно зберігати та виконувати дві моделі. Додатково з'являлися ризики плутання контекстів, якщо модель отримувала запити і для романтичного, і дружнього контексту.

Третій варіант полягав у поділі моделей не тільки за мовою, але й за контекстом — романтичним і дружнім. Це дозволило б досягти максимальної точності, оскільки кожна модель була спеціально адаптована до своєї задачі. Наприклад, романтичні моделі могли більш глибоко зосередитися на флірті та емоційному забарвленні повідомлень, тоді як дружні — на дружньому тоні, дослідженні поєднання інтересів та природності спілкування. Однак, цей підхід потребував би чотирьох окремих моделей, що суттєво збільшувало б витрати пам'яті та ускладнювало розгортання системи.

Після серії емпіричних досліджень було вирішено зупинитися на першому варіанті — використанні однієї універсальної моделі. Попередні

експерименти показали, що модель успішно справляється із завданням розпізнавання мови та контексту за допомогою спеціально вказаних тегів. Це дозволило зменшити використання пам'яті і значно спростити процес тренування та інтеграції системи. Завдяки правильному формулюванню запитів модель не допускала помилок зі змішуванням мов, що довело її ефективність в умовах багатомовного середовища.

Також, підтвердилися припущення щодо гіршого розуміння моделлю української мови відносно англійської. Тому варто приділити більше часу саме тренуванню на україномовних прикладах.

2.3.4. Тренування моделі

Для тренування та тестування моделей, будемо використовувати безкоштовні можливості платформи Colab від Google, що доступна за адресою colab.research.google.com. Тренування відбуватиметься на GPU T4. Платформа надає ~3 години безкоштовного використання графічного процесора на день.

Розглянемо детальніше алгоритм тренування моделі на нашому датасеті.

Для початку завантажуюмо попередньо навчений токенизатор та модель GPT-Neo-125M із бібліотеки Hugging Face Transformers.

```
model_name = "EleutherAI/gpt-neo-125M"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name,
                                              pad_token_id=tokenizer.eos_token_id)
```

Рис. 2.59. Завантаження та ініціалізація моделі і токенизатора

У випадку продовження навчання, змінну `model_name` слід замінити на локальний шлях до попередньої збереженої версії моделі.

Далі переносимо модель на доступний пристрій (GPU або CPU). У випадку запуску в оточенні з GPU у Google Colab, `device` матиме значення `cuda`.

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
```

Рис. 2.60. Перенос моделі на GPU, якщо це можливо

Далі переходимо до підготовки датасету. Завантажуємо датасети із вказаного файлу та розміщуємо їх в одному списку.

```
import os, json, random

with open(f'{FOLDER}/dataset_combined.json', 'r') as _file:
    combined_dataset = json.load(_file)

data = [*combined_dataset['romantic:uk'], *combined_dataset['friendship:uk'],
*combined_dataset['romantic:en'], *combined_dataset['friendship:en']]
print(len(data), 'examples loaded')
```

Рис. 2.61. Підготовка датасету

Після цього потрібно розбити наш загальний датасет на два окремі: для тренування та для перевірки. Це необхідно для того, щоб перевірити: показники моделі покращуються тому що вона вчиться знаходити причинно-наслідкові зв'язки, чи почала просто запам'ятовувати датасет. Для цього ми перемішуємо наш датасет, а потім розділяємо його на два у деякому співвідношенні (Рис. 2.62). Зазвичай використовується співвідношення 80/20, проте, в нас великий датасет, тому достатньо буде 90/10.

```
random.shuffle(data)

split_ratio = 0.9
split_idx = int(len(data) * split_ratio)
train_data = data[:split_idx]
valid_data = data[split_idx:]
```

Рис. 2.62. Розділення на датасет для тренування та для перевірки

Наступним кроком, виконуємо очищення даних. Під час створення датасету, у промпті містився статичний текст, що вказував моделі Gemini Advanced, який результат від неї очікується. В нашому випадку, модель буде отримувати на вхід профілі користувачів, а на виході лише генерувати привітальне повідомлення, отже залишити лише ці дані, убравши системний запит (міститься в середині тегу <system>). Це значно зменшить розмір промпту та заощадить токени, що, у свою чергу, пришвидшить генерацію. Очищені промпти зберігаються у нове поле `prompt_cleaned`.

```
import re

def cleanupPrompt(prompt):
    match = re.match(r'<system>(.)</system>\s*<user>(.)</user>\s*<you>(.)</you>', prompt)
    if not match:
        return prompt
    return f'<user>{match[2]}</user><you>{match[3]}</you>'

for user in data:
    user['prompt_cleaned'] = cleanupPrompt(user['prompt'])
```

Рис. 2.63. Очищення промптів

Після цього, переходимо до налаштування токенизатора. Налаштовуємо токен вирівнювання та розділення, адже вони відсутні у даному токенизаторі. Токен вирівнювання (padding) використовується для доповнення коротких запитів, щоб вони мали таку саму довжину, як і довгі. Токен розділення (separator) використовується для чіткого розділення запиту та відповіді моделі.

```
tokenizer.add_special_tokens({
    'pad_token': '[PAD]',
    'sep_token': '[SEP]',
})
```

Рис. 2.64. Додавання власних спеціальних токенів у токенизатор

Далі узгоджуємо розмір словника моделі та словника токенизатора.

```
model.resize_token_embeddings(len(tokenizer))
```

Рис. 2.65. Узгодження розміру словника моделі та токенизатора

Після цього, виконуємо токенизацію даних нашого датасету. Для початку формуємо текстові приклади у форматі {промпт} {роздільник} {відповідь} {кінцевий токен}. Також, до початку промпту додаємо мову та контекст даного прикладу у відповідних тегах. Це робиться для того, щоб модель чітко розрізняла їх. Далі виконуємо токенизацію, додаючи заповнювачі та обрізаючи довгі послідовності. Так як нам це доведеться робити для двох датасетів, обгортаємо дані дії у функцію tokenize, лістинг якої зображено на Рис. 2.66.

```
def tokenize(data, tokenizer):
```

```

    input_texts = [
        f"<lang>{item['lang']}</lang><context>{item['context']}</context>{item['prompt_
cleaned']}] {tokenizer.sep_token} {item['response']}] {tokenizer.eos_token}"
        for item in data
    ]

    encodings = tokenizer(
        input_texts,
        padding="max_length",
        truncation=True,
        return_tensors="pt",
        max_length=1024,
    )
    encodings["labels"] = encodings["input_ids"].clone()

    return encodings

train_encodings = tokenize(train_data, tokenizer)
valid_encodings = tokenize(valid_data, tokenizer)

```

Рис. 2.66. Функція для попередньої обробки та токенизації датасету

Після цього, створюємо клас, у який будемо обгорнути датасет для сумісності з бібліотеками PyTorch та Transformers.

```

class CustomDataset(torch.utils.data.Dataset):
    def __init__(self, encodings):
        self.encodings = encodings

    def __getitem__(self, idx):
        return {key: val[idx].clone().detach() for key, val in
self.encodings.items()}

    def __len__(self):
        return len(self.encodings["input_ids"])

train_dataset = CustomDataset(train_encodings)
valid_dataset = CustomDataset(valid_encodings)

```

Рис. 2.67. Представлення датасетів у вигляді PyTorch Dataset

Після успішної ініціалізації датасетів, задаємо параметри для майбутнього тренування (Рис. 2.68).

```

from transformers import TrainingArguments

training_args = TrainingArguments(
    num_train_epochs=10,

```

```
output_dir=f"./model-tmp",
report_to="wandb",
logging_steps=50,
per_device_train_batch_size=4,
logging_dir=f"./model-tmp-logs",
save_steps=5000,
save_total_limit=2,
eval_strategy="epoch",
do_train=True,
do_eval=True,
)
```

Рис. 2.68. Налаштування параметрів тренування моделі

Розглянемо основні параметри, що використовуються для налаштування процесу тренування моделей, включаючи їхню роль та значення.

Параметр `num_train_epochs` визначає кількість епох, тобто повних проходів моделі через весь навчальний датасет. Цей параметр важливий для контролю тривалості навчання моделі. Занадто мала кількість епох може призвести до недонавчання, коли модель не встигає засвоїти необхідні залежності, тоді як надто велика — до перенавчання, коли модель починає запам'ятовувати навчальні дані, втрачаючи здатність до генералізації.

Параметр `report_to` вказує, куди будуть надсилатися логи, що відображають прогрес тренування. У нашому випадку використовується сервіс `Weights and Biases`[50], який є потужним інструментом для моніторингу та аналізу експериментів машинного навчання. Він дозволяє зручно візуалізувати метрики, такі як похибка, швидкість сходження, зміни вагів тощо, що значно полегшує аналіз процесу навчання.

Параметр `per_device_train_batch_size` задає кількість прикладів, які обробляються моделлю за один цикл навчання перед оновленням вагів. Цей параметр напряму впливає на швидкість і стабільність тренування. Малі значення дозволяють економити оперативну пам'ять, але можуть призводити до нестабільності градієнтів. Великі значення потребують більше пам'яті, але забезпечують більш згладжений процес навчання.

Параметр `eval_strategy` визначає, коли саме буде виконуватись перевірка похибки моделі на валідаційному датасеті. У нашому випадку обрано стратегію перевірки після кожної епохи навчання. Це дозволяє відстежувати, як змінюється якість моделі в процесі тренування, і вчасно виявляти потенційні проблеми, наприклад, зупинити тренування, якщо якість почала погіршуватись через перенавчання.

Після цього ініціалізуємо тренера, передаючи йому модель, налаштування тренування та обидва датасети, та починаємо тренування (Рис. 2.69).

```
from transformers import Trainer

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=valid_dataset,
)

trainer.train()
```

Рис. 2.69. Ініціалізація тренера та початок тренування

Після завершення тренування, зберігаємо отриману модель разом із токенизатором для простого завантаження способом, зображеним раніше на Рис. 2.59. До імені теки додаємо поточну дату та час, щоб запобігти випадковим перезаписам. При цьому, модель зберігається у форматі Safetensors[44].

```
from datetime import datetime

LAST_SAVED_MODEL = f"{FOLDER}/model_gptneo-  
Combined_{datetime.now().isoformat()}"
model.save_pretrained(LAST_SAVED_MODEL)
tokenizer.save_pretrained(LAST_SAVED_MODEL)
```

Рис. 2.70. Збереження натренованої моделі

2.3.5. Застосування моделі

Для завантаження моделі з метою використання слід провести схожі кроки, як і з навчанням. Потрібно завантажити модель у пам'ять, перемикнути у режим виконання (`eval()`), перемістити на пристрій виконання, сформувати текстовий запит із профілів користувачів, додати бажані мови та контекст; токенізувати цей запит, передати його у модель, отримати результат, декодувати його та відрізати непотрібну частину. Розглянемо це у вигляді коду на Рис. 2.71.

1. Підключаємо бібліотеки та завантажуюмо модель у пам'ять

```
from transformers import AutoTokenizer, AutoModelForCausalLM
import torch

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
tokenizer = AutoTokenizer.from_pretrained(LAST_SAVED_MODEL)
model = AutoModelForCausalLM.from_pretrained(LAST_SAVED_MODEL).to(device).eval()
```

Рис. 2.71. Завантаження натренованої моделі та токенізатора

Якщо використання моделі відбувається відразу після тренування, цей крок можна опустити, адже ми вже маємо натреновану модель у змінній `model`.

2. Оголошуємо функцію, що виконує запит до моделі

```
def generate_message(input_text, max_length=1000):
    inputs = tokenizer(input_text, return_tensors="pt").to(device)

    outputs = model.generate(
        inputs["input_ids"],
        max_length=max_length,
        temperature=0.1,
        top_p=0.95,
        do_sample=True,
        pad_token_id=tokenizer.pad_token_id
    )

    return tokenizer.decode(outputs[0], skip_special_tokens=True)

def ask(prompt):
    output = generate_message(prompt)
    return output.split(f' {tokenizer.sep_token} ')[1]
```

Рис. 2.72. Виконання запиту до моделі

Показник температури (temperature) відповідає за баланс між точністю (менші значення) та випадковістю (більші значення). В нашому випадку, важливо якомога точніше відповісти на запит. Параметр `top_p` відсіює токени, що мають меншу вірогідність, таким чином, навіть при використанні високої температури, значення будуть актуальними.

Так як GPT моделі є лише більш просуненими моделями авто-доповнення тексту, модель повертає весь текст запиту, доповнюючи його в кінці власною відповіддю.

Функція `ask` розділяє оригінальний запит та відповідь за допомогою спеціального токена-роздільника (`[SEP]`), повертаючи лише відповідь моделі.

Функцію `ask` ми і будемо використовувати у сервісі під назвою `generateGreeting`. Для цього створимо відповідний Flask маршрут (Рис. 2.73).

```
@app.post("/greeting")
def generate_greeting():
    input_text = getText()

    try:
        return generateGreeting(input_text)
    except:
        return 'Generation failed...'
```

Рис. 2.73. Flask route для генерації привітання

Та налаштуємо бібліотеку Gradio (Рис. 2.73) для тестування моделі з метою налагодження.

```
from gradio.components import Textbox

interface = gradio.TabbedInterface(interface_list=[
    # ... інші моделі, зображені на Рис. 3.4
    gradio.Interface(fn=generateGreeting, inputs=Textbox(), outputs=Textbox()),
])
```

Рис. 2.74. Налаштування Gradio

Висновки до другого розділу

У другому розділі кваліфікаційної роботи були реалізовані поставлені завдання стосовно покращення сервісу для знайомств та пошуку друзів з використанням засобів машинного навчання.

Зокрема, створено класифікатор якості фотографій, побудований на базі Vision Transformer, що дозволяє автоматично оцінювати якість зображень, завантажених користувачами. Для цього проведено збір, аналіз і підготовку даних, формування та обробку датасету, а також тренування моделі та впровадження механізму до-навчання для підвищення її точності. Додатково впроваджено механізм розпізнавання непристойного контенту, що сприяє дотриманню стандартів безпеки сервісу.

Наступним кроком було розроблено систему верифікації особистості, яка базується на аналізі ключових точок обличчя з використанням технології FaceMesh. Ця система включає алгоритми нормалізації координат, завдання для перевірки живості користувачів, а також механізми виявлення заміни особи на фотографіях. Використання серверних технологій, таких як MTCNN та FaceNet, дозволило виконати порівняння фотографій, отриманих під час проходження верифікації з фотографіями у профілях користувачів, забезпечивши надійність верифікації.

Окрему увагу приділено створенню генератора перших повідомлень. На основі адаптованої мовної моделі створено систему, що генерує персоналізовані повідомлення для кожного користувача, враховуючи його профіль та інтереси. У рамках цієї задачі було проведено підбір базової архітектури, формування спеціалізованого датасету, визначення відповідних зон застосування та тренування моделі для забезпечення її ефективності.

РОЗДІЛ 3. АНАЛІЗ РЕЗУЛЬТАТІВ ТА ЗАСТОСУВАННЯ ПОКРАЩЕННЯ СЕРВІСУ ДЛЯ ЗНАЙОМСТВ ТА ПОШУКУ ДРУЗІВ ЗАСОБАМИ МАШИННОГО НАВЧАННЯ

3.1. Оцінка отриманого класифікатора якості фотографій

Оскільки, як було згадано раніше, поняття якості та доцільності фотографії, здебільш, суб'єктивне, то і оцінювати отриманий класифікатор за конкретними метриками складно.

Після перевірки моделі на тестовій вибірці (фотографій, що були завантажені користувачами на платформу та оцінені вже після тренування моделі), отримано показник точності 87.5%.

Під час суб'єктивного перегляду результатів класифікації моделі, на фотографіях з вибірки, що не використовувалась для навчання (фотографії з оцінкою 0), лише на 13 фотографіях зі 104 думка моделі не сходилися із критеріями, описаними у пункті 1.1.1. (Рис. 3.1), хоча навіть там “позиція” моделі була зрозуміла, адже фотографії дійсно балансували між поняттями “якісна” та “неякісна”.

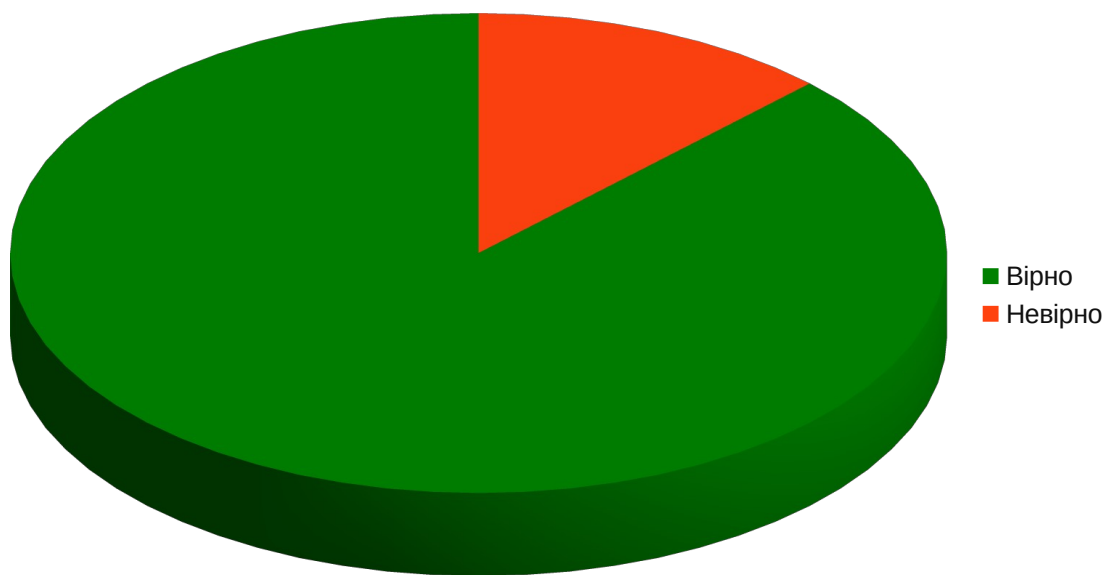


Рис. 3.1. Результати тестування на невизначеній вибірці

На основі цих даних було прийняте рішення признати експеримент успішним та використати отриману модель для оцінки якості фотографій користувачів на платформі для знайомств та пошуку друзів.

3.1.1. Підготовка до практичного використання

Підготовка та тренування моделі відбувалося у середовищі Jupyter Notebook на серверах Google Colab. Це дуже зручно під час експериментів, проектування та навчання нейронних мереж, але неприйнятно для використання на платформі. Тому отриману модель потрібно привести до вигляду, який підтримує мікросервіс платформи, що відповідає за запуск нейронних моделей.

Мікросервіс працює на Python версії 3.11.2, завантажує моделі за допомогою бібліотек PyTorch та Transformers, та надає до них доступ через протокол REST API, використовуючи бібліотеку Flask та Waitress у якості серверу. Під час налагодження та тестування моделей, використовується бібліотека Gradio[51], що дозволяє зручно запускати моделі, додаючи параметри та файли прямо у вікні браузера.

Для початку завантажимо модель під час запуску сервісу (Рис. 3.2).

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
image_quality = torch.load("models/image_quality.pt",
map_location=device).to(device)
```

Рис. 3.2. Завантаження та ініціалізація моделі

Далі перепишемо код таким чином, щоб він відповідав новим вимогам та оптимізуємо його для виконання двома способами: через тестовий інтерфейс Gradio та через endpoint у Flask.

Код класифікації зображено нижче на Рис. 3.3. Деякі статичні дані оголошуються всередині функції, хоча могли бути винесені за її межі (такі як імпорти, оголошення класів та оголошення функції трансформації). Це було

зроблено згідно поточної архітектури AI мікросервісу платформи для кращої підтримки.

```
def qualityDetect(img):
    import torch.nn.functional as F
    CLASSES = ['bad', 'good']
    transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
    ])
    img = transform(img).unsqueeze(0).to(device)
    output = image_quality(img)
    logits = output.logits
    probs = F.softmax(logits, dim=1)
    predicted_class = probs.argmax(dim=1).item()
    class_name = CLASSES[predicted_class]
    probability = probs[0][predicted_class].item()
    return { 'label': class_name, 'score': probability }
```

Рис. 3.3. Класифікація зображення та повернення результату

На початку ми оголошуємо класи, на яких навчалася модель. Далі розташована та сама функція трансформації, що перетворює зображення з масиву байтів на нормалізований тензор розміром 224x224. Наступним кроком трансформація застосовується для самого зображення, додається пустий вимір (unsqueeze) та відправляється у пам'ять відеокарти. Далі цей тензор передається у модель для класифікації, за результатом моделі відбирається клас із найвищим показником впевненості, що конвертується у текстовий вигляд, зрозумілий людині та повертається у вигляді звичайного JSON об'єкта.

Конфігурування Gradio відбувається наступним чином (Рис. 3.4).

```
from gradio import TabbedInterface, Interface
from gradio.components import Textbox, Image, JSON, Number
gr = TabbedInterface(interface_list=[
    Interface(fn=qualityDetect, inputs=Image(type="pil"), outputs=JSON()),
],
    tab_names=["Quality detect",]
)
gr.launch(server_name=HOST, server_port=PORT)
```

Рис. 3.4: Налаштування бібліотеки Gradio

Нижче, на Рис. 3.5, зображено Flask endpoint[52]. У ньому ми отримуємо зображення з тіла API запиту (у вигляді base64 стрічки), виконуємо запит до моделі за допомогою виклику функції на Рис. 3.3 та повертаємо результат.

```
@app.post("/quality/detect")
def quality_detect():
    return qualityDetect(getImage())
```

Рис. 3.5. Flask endpoint

На Рис. 3.6 представлено код конвертації зображення із формату base64 у формат бібліотеки PILLOW. Передбачається використання зображень формату JPEG.

```
def getImage():
    return Image.open(BytesIO(base64.b64decode(request.get_json()['image'])))
```

Рис. 3.6. Конвертація зображення із Base64 у формат PILLOW

Після завантаження моделі на сервер та тестування її у інтерфейсі Gradio (Рис. 3.7), можна вважати, що модель повністю готова для реального використання на платформі.

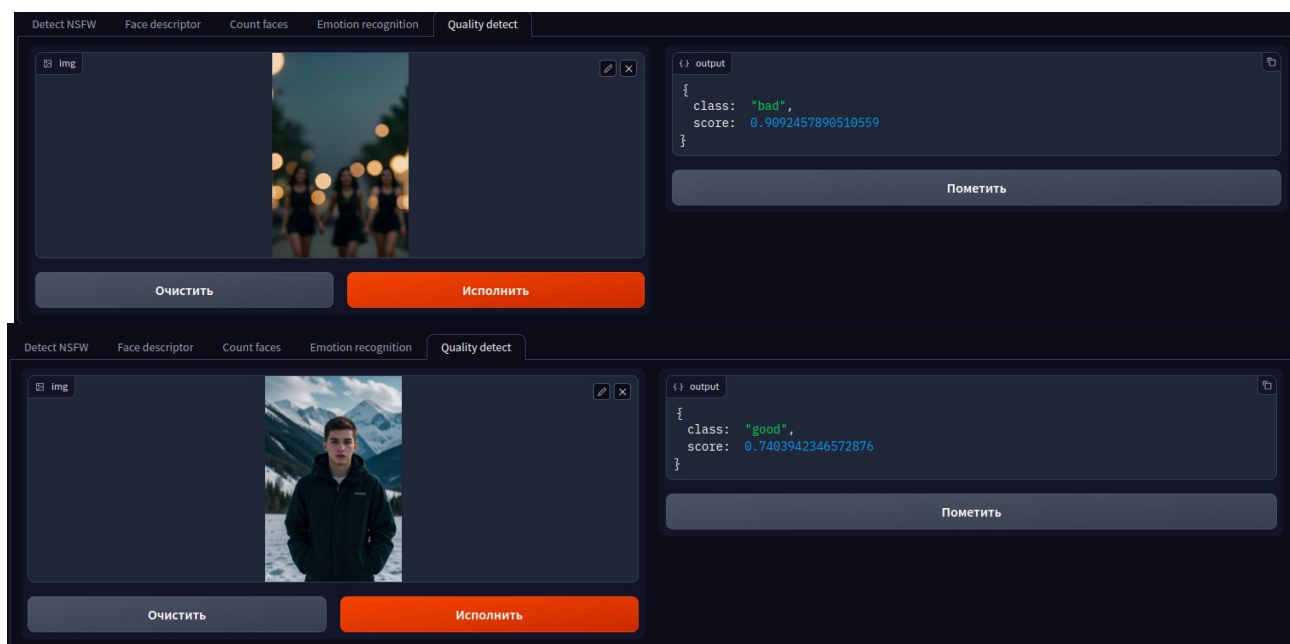


Рис. 3.7: Тестування фінального варіанту нейронної мережі

3.1.2. Можливості для подальшого покращення

Отримана модель у більшості випадків успішно класифікує фотографії за якістю, що вже сприяло підвищенню середнього показника якості фотографій платформи на 15%. Проте, існують додаткові можливості, для подальшого покращення. Розглянемо основні із них:

Розширення і оновлення датасету

Одним із найважливіших напрямків є продовження збору фотографій і соціологічних даних від користувачів. Це дозволить створити більш об'ємний та різноманітний набір даних, який забезпечить краще узагальнення та точність для моделі. З часом можна повторно провести навчання моделі, включивши нові дані для врахування тенденцій та змін у стилях фотографій.

Розширення спектру оцінки

Замість бінарної класифікації фотографій як "якісні" або "неякісні" можна перейти до багатокласового або неперервного підходу. Наприклад, створити шкалу оцінювання якості фотографій від 0 до 10, де користувачі зможуть більш детально оцінити фотографію. Таким чином, модель буде тренуватися на більш зважених даних і це дасть користувачам більш деталізовану інформацію про стан їх зображень.

Деталізація проблем якості

Для покращення користувацького досвіду, доцільно впровадити механізм класифікації проблем, які є у фотографії. Наприклад, модель може визначати тип дефекту, такий як низьке освітлення, розмитість, відсутність людини в кадрі, недоцільність фотографії або неправильний ракурс.

Надання рекомендацій щодо покращення

Інтеграція мультимодальних нейронних мереж для генерації текстових рекомендацій дозволить запропонувати користувачам конкретні дії для покращення своїх фотографій. Наприклад, можна автоматично радити змінити освітлення, позу чи фон.

3.2. Оцінка створеного процесу верифікації особистості

У ході даної роботи було створено повнофункціональний механізм верифікації. Перевірка на реальних користувачах додатку для знайомств та пошуку друзів показала, що процес верифікації, у загальному, є простим та зрозумілим для виконання. Отриманий клієнтський додаток успішно проводить користувача через процес верифікації, а серверна частина розраховує дескриптор обличчя для фотографій, отриманих під час верифікації та успішно порівнює його із фотографіями, завантаженими користувачем.

Процес, отриманий в ході даної роботи можна вважати таким, що відповідає поставленим вимогам, проте, існує декілька напрямків для можливого подальшого його покращення. Розглянемо декілька основних:

Спрощення складних для виконання інструкцій

Для покращення досвіду користувачів, корисним буде запровадження детальної статистики для аналізу виконання завдань, що дасть змогу відслідковувати загальну успішність і виявляти найпоширеніші проблеми.

Таким чином, можна буде виявити складні або незрозумілі інструкції та спростити їх сприйняття та виконання, адже головною метою процесу верифікації є перевірка справжності власника профілю. Чим менше зусиль прикладе користувач для проходження цього процесу, тим кращий досвід в нього залишиться.

Зворотній зв'язок під час виконання завдання

Ще одна ідея, що призведе до покращення користувацького досвіду, полягає у додаванні візуального зворотного зв'язку під час виконання завдання. Таким чином, наприклад, можна буде показати, що рот відкритий, але недостатньо широко для зарахування завдання.

Це можуть бути як звичайні прогрес-бари, так і щось більш цікаве, наприклад, гра «намалюй трикутник кінчиком носа».

Додати візуалізацію завдань

Іншим важливим аспектом є візуалізація. Розширення функціоналу за рахунок використання 3D-моделей або анімованих ілюстрацій допоможе чіткіше пояснювати, як виконати завдання. Часто користувачі плутають поворот та нахил голови, тому візуальне представлення може значно спростити виконання подібних завдань.

Додаткові перевірки

Додатково, для покращення безпеки, можна впровадити більш детальну перевірку на живість через аналіз мікрорухів обличчя, наприклад, неприродно стабільне положення обличчя чи незвичне для людини миготіння, яке часто зустрічається в анімованих зображеннях або підробках на основі штучного інтелекту.

Також, підвищенням безпеки буде запис відео з камери та повторна перевірка його на стороні сервера.

Підтримка людей з обмеженими можливостями

Ще одним важливим напрямком розвитку системи є забезпечення інклюзивності для людей із вадами здоров'я. Наприклад, для осіб, які не можуть виконати завдання через фізичні обмеження (такі як відсутність одного ока, проблеми з моторикою чи інші особливості), можна передбачити альтернативні способи виконання інструкцій. Система могла б автоматично адаптувати набір завдань на основі виявлених особливостей, наприклад, замінювати кліпання очима на нахил голови чи інший жест, який користувач здатен виконати.

Збільшений розмір шрифту та звуковий супровід інструкцій (з використанням технології TTS) зробить процес доступнішим для людей із обмеженнями зору.

Також важливо забезпечити можливість повідомлення користувачами про свої потреби на етапі налаштування, що дозволить налаштувати процес максимально під їхні можливості.

3.3. Оцінка отриманого генератора перших повідомлень

Розглянемо отриману модель GPT-Neo 125M, що пройшла finetuning процес на нашому датасеті. Сумарно модель пройшла 16 епох тренування, набувши Cross Entropy Loss у значенні 0,0248. Графік зміни цього значення зображено на Рис. 3.8.

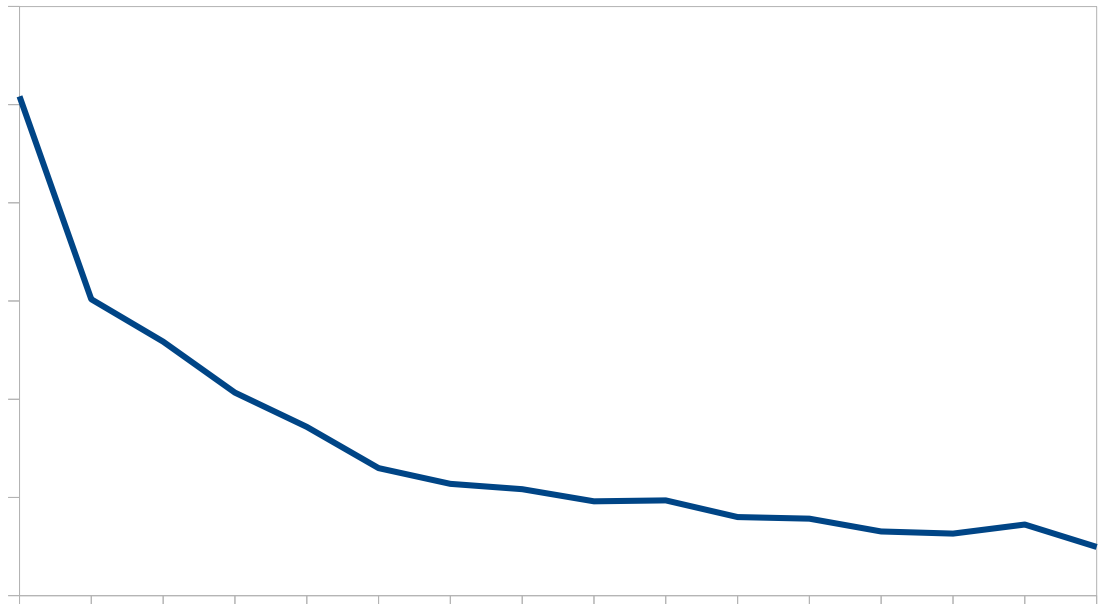


Рис. 3.8. Cross Entropy Loss протягом навчання

Натренована модель показує доволі непогані результати. Як можна побачити на Рис. 3.9, вона успішно навчилася генерувати осмислений текст англійською та українською мовою з використанням інформації про інтереси користувачів.

Мова	Контекст	Приклад відповіді
uk	romantic	Привіт! Я тут шукаю людину, яка зможе скласти мені компанію для перегляду серіалів під ковдрою. Ти підходиш за описом?
uk	friendship	Привіт! Я помітив, що ти теж любиш читати. Які жанри тобі найбільше подобаються? Може, ти порадиш мені щось цікаве?
en	romantic	Your green eyes are simply captivating! And you also mentioned sports. What's your favorite sport? Maybe we can find a common activity together?
en	friendship	Hey there! I noticed we both have an interest in Motion Design, which is pretty cool! What kind of projects do you enjoy working on?

Рис. 3.9. Приклади відповіді моделі у різних контекстах

Проте, швидкодія отриманої моделі не відповідає вимогам, адже один запит англійською на GPU T4 генерується близько секунди, на CPU близько 3 секунд, а українською мовою — близько 15 секунд, що в 5 разів довше. Необхідно розглянути можливості оптимізації для того, щоб зменшити час виконання моделі до 1 секунди на CPU.

3.3.1. Оптимізація токенизатора

Першим фактором, який слід оптимізувати є токенизатор. Переглянувши результат токенизації типового запиту, ми помітили, що стандартний токенизатор демонструє низьку продуктивність у роботі з українською мовою. Наприклад, текст українською мовою розбивається на рівні окремих літер, через що на кожне слово чи навіть літеру витрачається надмірна кількість токенів. Така ситуація є критичною з точки зору швидкодії та економії ресурсів, адже кількість токенів безпосередньо впливає на продуктивність моделі. Чим більше токенів надходить на вхід, тим довше триває обробка тексту, і тим більша ймовірність перевищення розміру контекстного вікна моделі. Також, це значним чином впливає на швидкість генерації відповіді, адже за одну ітерацію додається одна літера, а не частина цілого слова.

Ще однією проблемою є неефективна токенизація вхідного тексту, що складається із суміші XML та JSON. Елементи, що повинні сприйматися як одне ціле (наприклад, `<lang>uk</lang>`), поділяються на декілька окремих токенів. Це не лише погіршує точність сприйняття інформації моделлю, а ще й збільшує кількість токенів, що поступають на вхід моделі, призводячи до зменшення швидкості обробки та генерації тексту.

Для вирішення проблеми із ефективною токенизацією вхідного пром프트, можна вручну додати основні необхідні токени.

Наприклад, конструкція `<lang>uk</lang>` повинна сприйматися як один токен замість восьми (`<`, `lang`, `>`, `uk`, `<`, `/`, `lang`, `>`). Це значно зменшує

кількість токенів, забезпечує коректну обробку складних структур і, відповідно, підвищує ефективність роботи моделі.

Для цього ми аналізуємо формат вхідних даних і визначаємо ключові шаблони, які повинні залишатися неподільними. Заносимо їх в окрему змінну `PREDEFINED_TOKENS` (Рис. 3.10), адже вони знадобляться у майбутньому під час тренування токенизатора.

```
PREDEFINED_TOKENS = [ "[SEP]",
    '<lang>uk</lang>', '<lang>en</lang>', '<context>romantic</context>',
    '<context>friendship</context>',
    '<user>', '</user>', '<you>', '</you>', '[]',
    # keys
    '"mode":', '"gender":', '"age":', '"details":', '"childrenPlans":',
    '"eyeColor":', '"height":', '"hairColor":'
    # values
    '"blond"', '"brown"', '"black"', '"red"', '"gray"', '"painted"', '"bald"',
    # ... решту значень опущено
]
```

Рис. 3.10. Формуємо список оптимізованих токенів

Далі, сформуємо датасет, на якому будемо тренувати токенизатор. Для цього візьмемо описаний раніше датасет для тренування моделі та об'єднаємо усі мови та контексти в один список. Відкриємо `.txt` файл для запису та для кожного елементу датасету створимо стрічку цього файлу, де об'єднаємо як запит до моделі, так і її відповідь, щоб тренер оптимізував токенизатор для всіх варіантів використання. Лістинг коду зображено на Рис. 3.11.

```
import json

with open(f'{FOLDER}/dataset_combined.json', 'r') as _file:
    combined_dataset = json.load(_file)

data = [*combined_dataset['romantic:uk'], *combined_dataset['friendship:uk'],
        *combined_dataset['romantic:en'], *combined_dataset['friendship:en']]

with open(f'{FOLDER}/dataset_tokenizer.txt', 'w') as _file:
    for item in data:
        _file.write(item['prompt_cleaned'] + ' [SEP] ' +
            item['response'].replace('\n', ' ') + '\n')
```

Рис. 3.11. Формування датасету для тренування токенизатора

Наступним кроком, експортуємо словник (`vocab.json`) та правила поєднання (`merges.txt`) оригінального токенизатора моделі та створюємо на їх основі новий Byte Pair Encoding[53] токенизатор, який будемо тренувати.

```
from tokenizers import Tokenizer, models, trainers, pre_tokenizers, decoders, processors
from transformers import PreTrainedTokenizerFast, AutoTokenizer
import os

original_tokenizer.save_vocabulary("./")
tokenizer_trained = Tokenizer(models.BPE.from_file("vocab.json", "merges.txt"))
tokenizer_trained.pre_tokenizer =
pre_tokenizers.ByteLevel(add_prefix_space=False)
tokenizer_trained.decoder = decoders.ByteLevel()
```

Рис. 3.12. Створення нового токенизатора на основі словника існуючого

Розглянемо детальніше деякі методи, що були використані на Рис. 3.12.

BPE (Byte Pair Encoding) — це метод компресії даних, який використовується для сегментації тексту на під-словники та широко використовується у GPT моделях. BPE забезпечує ефективне кодування тексту, що дозволяє зменшити кількість унікальних tokenів і підвищити якість генерації моделей на основі трансформерів.

ByteLevel — попереднє токенизування на рівні байтів, що дозволяє уникнути проблем з обробкою символів, які не входять до словника токенизатора, даючи можливість не створювати окремі токени для рідко вживаних символів, при цьому, зберігаючи можливість їх використання у випадку необхідності.

Перед початком тренування, додаємо власний набір tokenів, описаний вище. Завдяки цьому, ці токени не будуть створені під час тренування, а будуть використовуватися в описаному нами оптимальному вигляді.

```
tokenizer_trained.add_tokens(PREDEFINED_TOKENS)
```

Рис. 3.13. Реєстрація власних tokenів

Після цього налаштовуємо BPE-тренер (Рис. 3.13). `vocab_size` — це максимальна кількість tokenів у словнику, `min_frequency` — мінімальна кількість повторів елемента для того, щоб він міг стати токеном.

```
trainer = trainers.BpeTrainer(vocab_size=66666, min_frequency=2,
                             special_tokens=['[PAD]'])
```

Рис. 3.14. Налаштування тренера

Далі розпочинаємо тренування, передавши раніше згенерований файл `dataset_tokenizer.txt`. Після завершення тренування зберігаємо отриманий токенизатор для подальшого використання.

```
tokenizer_trained.train(
    files=[ f'{FOLDER}/dataset_tokenizer.txt' ],
    trainer=trainer
)

tokenizer_trained.save(f'{FOLDER}/tokenizer_new.json')
```

Рис. 3.15. Тренування та збереження нового токенизатора

Як видно із даних таблиці 7, що зображена нижче, токенизатор, натренований на нашому датасеті, навчився оптимально розподіляти токени як англійською, так і українською мовами. Це значно підвищить продуктивність моделі при роботі з українською мовою, виводячи її на рівень англійської мови, що вирішить проблему, описану у параграфі 3.3.

Таблиця 7. Результат оптимізації токенизатора

Фраза	Кількість токенів	
	Старий токенизатор	Новий токенизатор
Hello, world!	4	5
Привіт, світ!	16	7

Після створення нового токенизатора, необхідно перенавчити модель з ним працювати, оскільки зміна словника токенизації змінює і представлення даних («сенси» токенів для моделі).

3.3.2. Використання DistilGPT-2

Оскільки після оптимізації токенизатора значно зменшилася кількість токенів, що необхідні для кодування інструкцій та українського тексту, та зміна токенизатора вимагає перенавчання моделі для роботи з ним, було прийняте рішення повернутися до використання базової моделі DistilGPT-2 через її швидкість і ефективність. адже тепер вікна контексту у 1024 токени буде цілком достатньо. Модель GPT-Neo, хоча і є більш потужною, краще «розуміє» контекст та демонструє вищу точність, однак через це генерує текст значно повільніше. Оскільки у нашому випадку швидкість інференсу є ключовим показником, було вирішено повернутися до DistilGPT-2 та тренувати для роботи з новим токенизатором саме її.

Сам процес тренування моделі вже був детально описаний у пункті 2.3.4. другого розділу, тому його повторний опис є недоцільним. Єдині відмінності — потрібно використовувати новий токенизатор та не потрібно додавати спеціальні токени, адже вони вже були додані під час тренування токенизатора. Проте не слід забувати про узгодження розміру словників моделі та токенизатора перед початком навчання. Як це зробити зображено на Рис. 2.65.

Розглянемо результати тренування моделі DistilGPT-2 з новим токенизатором. Нижче, на Рис. 3.16, зображено зміну значень похибки тренування (Training Loss) та похибки валідації (Validation Loss). Чим менше це значення — тим краще модель навчилася передбачати наступний токен у послідовності. Також, якщо показник Training Loss стає нижчим за Validation Loss та останній перестає змінюватись — це може свідчити про перенавчання моделі, коли модель починає «запам'ятовувати» правильні відповіді замість вивчення основних принципів. В такому випадку рекомендовано зупинити навчання.

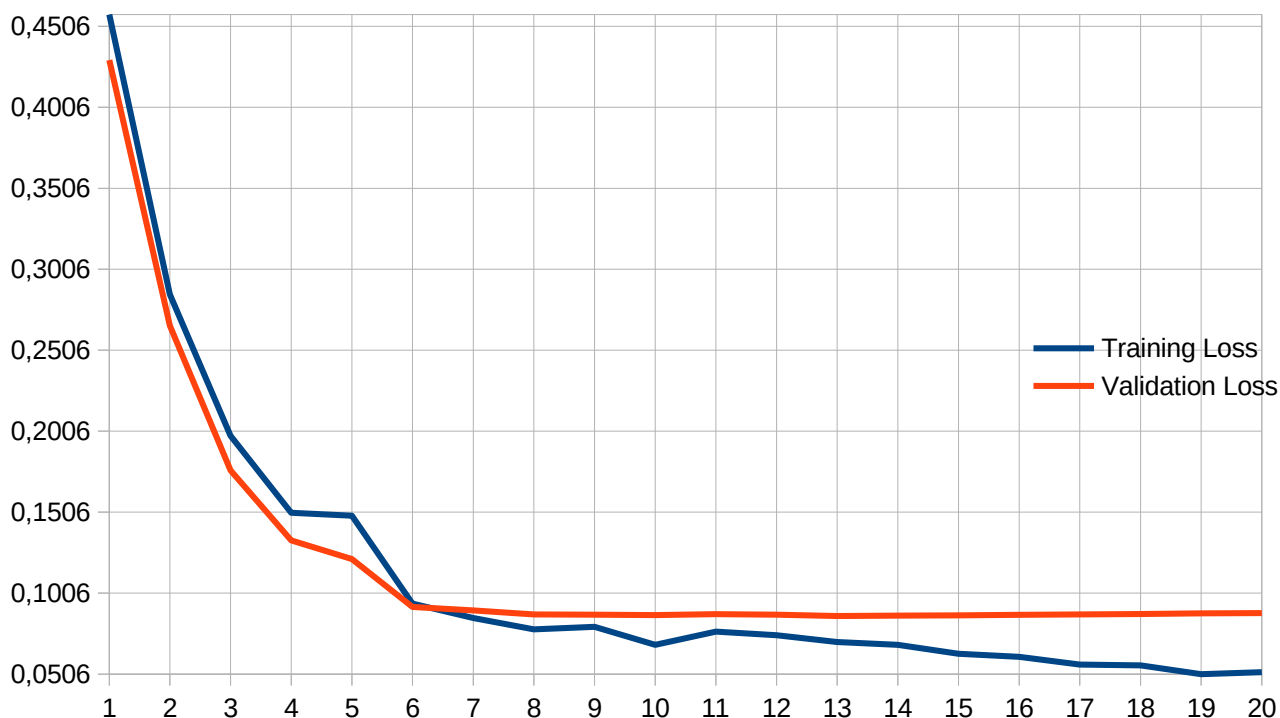


Рис. 3.16. Значення похибок протягом тренування моделі

Як видно з діаграми на Рис. 3.16, після сьомої епохи показник Validation Loss перестав значним чином змінюватись, а показник Training Loss продовжив зменшуватись. Це свідчить про початок перенавчання моделі. При більш детальному вивченні чисел, найкращий показник Validation Loss отримано на 14 епосі (0,086681), тому було вирішено використовувати саме цю версію моделі.

3.3.3. Використання `stopping_criteria` для оптимізації інференсу

Після перевірки виводу моделі за допомогою тимчасового відключення «`skip_special_tokens`» у налаштуваннях декодера токенизатора, було виявлено, що після eos-токену (`<|endoftext|>`) модель продовжує генерувати текст, який потім відсікається.

```
Answer: Привіт! Бачу, ти теж любиш поринути у світ комп'ютерних ігор. Яка твоя улюблена карта в CS:GO? Може, разом знайдемо
щось новеньке для себе?<|endoftext|>g<|endoftext|>.<|endoftext|>, ти знаєш якийсь цікавий, який жанр?<|endoftext|><|endoftext|>
<|endoftext|><|endoftext|><|endoftext|><|endoftext|><|endoftext|><|endoftext|><|endoftext|><|endoftext|><|endoftext|><|endoftext|>
<|endoftext|><|endoftext|><|endoftext|><|endoftext|><|endoftext|><|endoftext|><|endoftext|><|endoftext|> <|endoftext|> <|endoftext|>
"sometimes")),"communicationStyle": "meetings", "friendshipGoal": "chatting", "maritalStatus": "single", "education": "higher", "eyeColor": "blue",
```

Рис. 3.17. Після EOS-токену, модель продовжує генерацію

Така поведінка не є оптимальною, адже модель витрачає час та ресурси на генерацію цих непотрібних токенів.

Для виправлення даної проблеми було вирішено використати параметр `stopping_criteria`, який дозволяє зупиняти генерацію на основі заданих умов. Це допомагає уникнути зайвого навантаження на CPU та покращити ефективність процесу. Нижче наведено приклад реалізації параметра `stopping_criteria` з використанням спеціального класу `StopOnEOSToken` (Рис. 3.18).

```
from transformers import StoppingCriteria, StoppingCriteriaList

class StopOnEOSToken(StoppingCriteria):
    def __init__(self, eos_token_id):
        self.eos_token_id = eos_token_id

    def __call__(self, input_ids, scores, **kwargs):
        return input_ids[0, -1] == self.eos_token_id
```

Рис. 3.18. Stopping criteria для зупинки генерації після EOS токена

Після цього, цей клас можна застосувати під час генерації. (код на Рис. 3.19 це частина коду з Рис. 2.72 із показаними змінами).

```
def generate_message(input_text, max_length=1000):
    inputs = tokenizer(input_text, return_tensors="pt").to(device)

    outputs = model.generate(
        inputs["input_ids"],
        max_length=max_length,
        temperature=0.1,
        top_p=0.95,
        do_sample=True,
        pad_token_id=tokenizer.pad_token_id,
        + stopping_criteria=stopping_criteria
    )

    return tokenizer.decode(outputs[0], skip_special_tokens=True)
```

Рис. 3.19. Застосування `stopping_criteria`

Даний підхід дозволяє припиняти генерацію тексту одразу після появи eos-токену, як зображено на Рис. 3.20. Використання цього методу значно

скоротило час виконання моделі. Зокрема, час генерації тексту на CPU зменшився з 5 секунд до 2 секунд.

Answer: Привіт! Бачу, ти теж любиш поринути у світ комп'ютерних ігор. Яка твоя улюблена карта в CS:GO? Може, разом знайдемо щось новеньке для себе?</endoftext>

Рис. 3.20. Генерація після введення `stopping_criteria`

3.3.4. Оптимізація для виконання на CPU

Після успішної оптимізації токенизатора та повторного тренування моделі, важливим кроком стала її оптимізація. Основною метою оптимізації є зменшення обчислювальних вимог та підвищення продуктивності при роботі на CPU. Для цього розглянемо конвертацію моделі у формат ONNX та її квантизацію.

Формат ONNX (Open Neural Network Exchange) — це відкритий стандарт для обміну моделями машинного навчання, який забезпечує сумісність між різними фреймворками та бібліотеками. Конвертація моделі у формат ONNX дозволяє суттєво оптимізувати її виконання за рахунок спеціальних інструментів, таких як ONNX Runtime[54], які покращують продуктивність при роботі з нейронними мережами на CPU.

Квантизація — це процес зменшення розрядності чисел, які використовуються для представлення параметрів (вагів) моделі. Зазвичай, моделі навчаються з використанням 32-бітових чисел з плаваючою комою (FP32), що забезпечує високу точність, але потребує значних обчислювальних ресурсів. Під час квантизації моделі параметри можуть бути зменшені до 16-бітових чисел (FP16) або навіть до 8-бітових (INT8), що значно знижує споживання пам'яті та прискорює обчислення. Такий підхід дозволяє використовувати потужні моделі навіть на менш продуктивних процесорах (наприклад, на смартфонах), зберігаючи при цьому прийнятний рівень точності.

Існують два основні підходи до квантизації: статична та динамічна.

Статична квантизація передбачає попереднє обчислення мінімальних і максимальних значень активацій та ваг під час спеціальної фази калібрування перед використанням моделі. Це дозволяє зафіксувати відповідні діапазони чисел, що покращує продуктивність, але вимагає додаткового етапу підготовки.

Динамічна квантизація виконується "на льоту", тобто діапазони чисел обчислюються безпосередньо під час виконання моделі. Цей підхід є більш гнучким і простим у застосуванні, особливо для сценаріїв, де необхідна швидка адаптація моделі до різних даних. Саме динамічна квантизація була обрана для оптимізації нашого проєкту, оскільки вона забезпечує відчутне зменшення використання ресурсів без значного впливу на якість результатів.

Optimum[55] — це бібліотека та набір інструментів від Hugging Face, що дозволяє оптимізувати моделі для різних платформ і прискорити їх виконання.

Для початку встановлюємо необхідні модулі:

```
!pip install optimum onnx onnxruntime
```

Рис. 3.21. Встановлення бібліотек

Далі конвертуємо модель у формат ONNX:

```
from optimum.onnxruntime import ORTModelForCausalLM

ort_model = ORTModelForCausalLM.from_pretrained(model_name, export=True)

save_dir = "./model_onnx"
ort_model.save_pretrained(save_dir)
tokenizer.save_pretrained(save_dir)
```

Рис. 3.22. Конвертація у формат ONNX, збереження моделі та токенизатора

Інші API викликів не змінюються, отже можна завантажити модель у змінну `model` та весь код інференсу, описаний у пункті 2.3.5. , продовжить

працювати. Єдина відмінність — ніде не потрібно використовувати метод `.to(device)`, адже ONNXRuntime працює на CPU.

Наступним кроком виконуємо динамічну квантизацію за допомогою ONNXRuntime:

```
from optimum.onnxruntime import ORTQuantizer
from optimum.onnxruntime.configuration import AutoQuantizationConfig

quantizer = ORTQuantizer.from_pretrained(ort_model)

dqconfig = AutoQuantizationConfig.avx512_vnni(
    is_static=False,
    per_channel=False
)

quantizer.quantize(
    save_dir="./model_onnx_quantized",
    quantization_config=dqconfig
)
```

Рис. 3.23. Застосування динамічної квантизації та збереження нової моделі

При використанні динамічно-квантизованої моделі, наступні виклики інференсу будуть виконуватися швидше. Після цього можна додатково оптимізувати граф моделі:

```
from optimum.onnxruntime import ORTOptimizer
from optimum.onnxruntime import AutoOptimizationConfig

optimizer = ORTOptimizer.from_pretrained(ort_model_quantized_dyn)

optimization_config = AutoOptimizationConfig.O2()
```

Рис. 3.24. Налаштування оптимізатора

Існують кілька рівнів оптимізації моделей в `AutoOptimizationConfig`, включаючи O1, O2 та інші. Кожен рівень має свої характеристики та призначення, описані на таблиці 8.

Таблиця 8: Опис рівнів оптимізації графу моделі

Рівень оптимізації	Опис
O1	Базові оптимізації, які не змінюють архітектуру моделі.
O2	Базові та розширені оптимізації, включаючи спеціальні перетворення для Transformers, такі як fusions (техніка, що об'єднує декілька окремих операцій у обчисленнях моделі в одну оптимізовану операцію).
O3	Включає всі оптимізації рівня O2, а також апроксимацію функції GELU для зменшення обчислювальної складності.
O4	Включає всі оптимізації рівня O3 та додає змішану точність (fp16), доступну лише для GPU. Цей рівень забезпечує найвищу швидкість виконання, але потребує підтримки відповідного обладнання.

Також, є можливість детально налаштувати параметри оптимізації вручну, але у нашому випадку було обрано рівень O2, оскільки він забезпечує оптимальний баланс між швидкістю виконання та точністю.

```
optimizer.optimize(save_dir="./model_onnx_quantized_optimized",
                    optimization_config=optimization_config)
```

Рис. 3.25. Виконуємо оптимізацію та зберігаємо нову модель

Оптимізована модель дозволяє швидше обробляти запити та генерувати тексти на CPU. Квантизація допомагає досягти балансу між точністю та продуктивністю, а також, знижує затрати дискової та оперативної пам'яті.

Таблиця 9. Порівняння часу виконання різних варіантів моделі

Варіант	Час виконання (с)
GPU (PyTorch)	0.6627
CPU (PyTorch)	2.5397
CPU (ONNX)	2.3892
ONNX Quantized	1.2580
+ Graph Optimized	1.0852

На таблиці 9 зображено час виконання різних варіантів моделі. Як можна побачити, час виконання на CPU значно зменшувався з кожною оптимізацією та майже досяг значення початкового виконання на GPU. Це є дуже хорошим результатом, що відповідає вимогам, заданих у пунктах 1.4.3. та 3.3.

3.3.5. Можливості для подальшого покращення

Хоча ми досягли усіх поставлених вимог до моделі, існують декілька напрямків для подальшого покращення якості моделі. Розглянемо основні із них:

Ревізія датасету

Один із найбільш перспективних способів підвищення якості моделі — ревізія та удосконалення датасету.

По-перше, необхідно перевірити, чи роди повідомлень відповідають статі задіяних користувачів, щоб уникнути можливих помилок у звертаннях або формулюваннях.

По-друге, слід оцінити коректність використання інтересів у повідомленнях — чи справді модель генерує текст, що відповідає спільним або унікальним характеристикам профілів.

По-третє, важливо гарантувати, що повідомлення створюються потрібною мовою, відповідно до налаштувань користувача. Було виявлено декілька випадків, коли при явному вказанні української мови, у датасет потрапили повідомлення англійською, та, навіть, російською мовами. Під час підготовки датасету було проведено швидке алгоритмічне виявлення таких повідомлень, але, через те, що повідомлення могли містити окремі іншомовні слова,

Найкращим варіантом буде передивитися та по виправляти датасет у ручному режимі, адже це дасть найбільшу впевненість в якості результату. Проте, це займе великий обсяг часу та зусиль через об'єм даних у датасеті.

Використання більших моделей

Для розширення можливостей моделі можна розглянути використання більших архітектур, таких як GPT-3 (GPT-Neo), LLaMA, Qwen або Mistral. Ці моделі можуть забезпечити доступ до ширших контекстів і даних про зовнішній світ, що сприятиме генерації більш детальних, осмислених та креативних повідомлень. Проте їх застосування потребує значних обчислювальних ресурсів і оптимізації для забезпечення швидкодії.

Можна роздивитися такий варіант у випадку розгортання на більш потужний сервер із доступними GPU.

Використання опису "про мене"

Додавання тексту із секції "про мене" профілів обох користувачів до контексту генерації може значно підвищити релевантність повідомлень. Особливо, це зіграє роль у випадку використання більш великих моделей.

Однак, необхідно враховувати ризики, пов'язані із захистом від prompt injection — маніпуляцій користувачів із вмістом цього поля, що можуть вплинути на генерацію.

Використання фотографій у якості контексту

Інтеграція фотографій як джерела додаткової інформації може допомогти моделі генерувати ще більш персоналізовані повідомлення. Наприклад, аналіз фотографій для визначення ключових деталей (хобі, оточення, тощо) може додати нові можливості для адаптації контексту. Такий підхід потребує впровадження методів обробки зображень, таких як мультимодальні нейронні мережі, що значно підвищить складність системи, але потенційно відкриє нові горизонти для її застосування.

3.4. Застосування інших досліджень

3.4.1. Аналіз емоції на фотографії

Розпізнавання емоції на фотографії є важливим інструментом, що може бути використаний для збагачення досвіду користувачів у додатках для знайомств. Хоча цей компонент наразі не знайшов прямого практичного застосування в основній функціональності платформи, він використовується для статистичного аналізу та дослідження потенційних шляхів оптимізації взаємодії користувачів і покращення якості фотографій, які вони завантажують.

Для класифікації фотографій за емоціями використовується модель `dimas806/face_emotions_image_detection`[56] з бібліотеки HuggingFace. Вона також використовує архітектуру Vision Transformer та розподіляє емоції на зображеннях за наступними категоріями[57]:

Назва класу	Опис
ahegao	Вираз обличчя із сексуальним підтекстом
angry	Злість, лють
happy	Щастя, веселість, посмішка
neutral	Спокійний вираз обличчя
sad	Сум, смуток, печаль
surprise	Здивованість, зацікавленість

До перспектив застосування моделі належить покращення контенту профілю, де, аналізуючи емоції на фотографіях користувачів, можна надати рекомендації щодо того, які зображення краще використовувати для приваблення більшої уваги. Також модель може бути корисною для статистичного аналізу, адже дані про емоції можуть використовуватися для збору статистики щодо загального емоційного клімату на платформі, що сприятиме прийняттю стратегічних рішень на рівні платформи. Окрім цього,

вона може допомогти у виявленні непристойних фотографій, зокрема через оцінку вірогідності класу `ahgao`, що стане додатковим покращенням механізму, описаного у пункті 2.1.7. .

Завантаження моделі виконується за допомогою механізму `pipeline` бібліотеки `Transformers`:

```
emotion_recognizer = pipeline(  
    task = "image-classification",  
    model = "dima806/face_emotions_image_detection"  
)
```

Рис. 3.26. Завантаження моделі для розпізнавання емоцій

Зображення для аналізу спочатку обробляються за допомогою алгоритму `MTCNN`, який виконує кадрування облич на фотографіях (Рис. 3.27), оптимізуючи їх для подальшої обробки. Це дозволяє моделі більш ефективно аналізувати вирази облич, концентруючись на релевантних областях зображення.

```
def getEmotion(img, min_probability=0.9):  
    cropped = cropFace(img, min_probability)  
  
    if cropped is None:  
        return []  
  
    return emotion_recognizer(cropped)
```

Рис. 3.27. Використання моделі для розпізнавання емоцій

Для практичного використання, усі вищезгадані моделі було додано до `AI-сервісу`, описаного у пункті 3.1.1. даної роботи. Кожна модель має свій окремий шлях доступу. Крім того, моделі визначення якості фото, визначення емоцій, підрахунок кількості людей на фото та розрахування дескриптора обличчя об'єднані в окремий конвеєр для оптимізації швидкості виконання.

Кожна завантажена фотографія проходить через цей конвеєр у асинхронному режимі. За результатами тестування, класифікація однієї фотографії всіма моделями займає близько 1 секунди на `CPU`, що є хорошим результатом та відповідає нашим вимогам.

Також, розглядалася можливість передбачення віку та статі людини на фото, проте усі моделі, що розглядалися мали дуже високу похибку. А у випадку автоматизованого виявлення маніпуляції віком це є недопустимим. Розглядається теоретична можливість розробки власної моделі у майбутньому.

3.4.2. Аналіз ефективності профілю

Ще одним значним покращенням, реалізація якого стала можливою завдяки імплементації засобів машинного навчання, стала сторінка з рекомендаціями стосовно покращення ефективності профілю. Часто користувачі не розуміють, чому їх профіль отримує замало симпатій, що може призводити до розчарування та зниження довіри до платформи. Багато з них вважають, що проблема криється у системних обмеженнях чи недоліках алгоритмів, хоча насправді ключову роль грають якість фотографій, заповненість профілю та активність користувача.

Для вирішення цієї проблеми платформа пропонує персоналізований звіт про ефективність профілю.

Під час формування звіту аналізуються такі фактори, як наявність проблем зі сповіщеннями на стороні клієнта, загальна кількість фотографій, кількість фотографій, що визнані непристойними, кількість зображень із більше ніж однією особою, а також фотографій без обличчя або поганої якості. Розглядається кількість застосованих фільтрів, відсоток анкет, що відсіюються поточними фільтрами, кількість нерозпочатих діалогів та не переглянутих симпатій. Також враховуються відсоток заповненості профілю, активність користувачів, статус проходження верифікації, кількість зазначених інтересів і якість результатів проходження психологічного тесту.

На основі цих даних формується перелік конкретних рекомендацій (Рис. 3.28) для користувача щодо покращення якості та ефективності його профілю. Це дозволяє користувачеві не лише покращити сприйняття свого профілю

іншими учасниками платформи, але й підвищує загальну ефективність його взаємодії на сервісі.

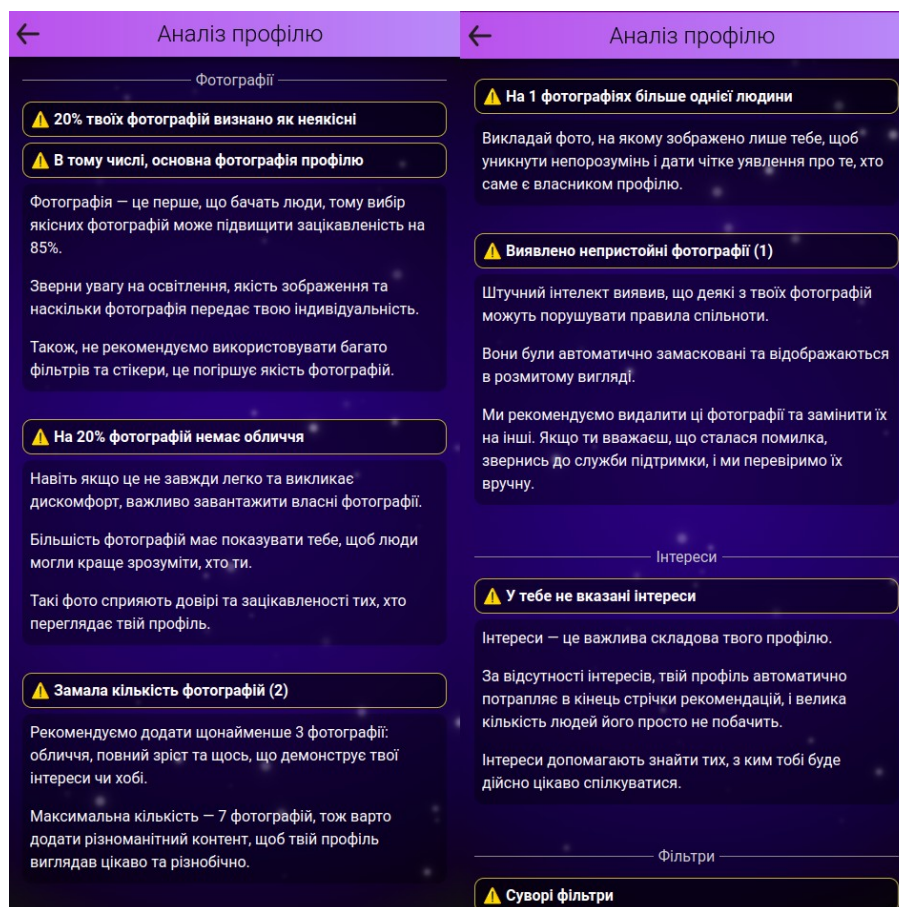


Рис. 3.28. Деякі проблеми, які виявляє аналізатор

Дані для звіту формуються на сервері, базуючись на результатах, що повертають моделі розпізнавання непристойних фотографій, MTCNN (кількість облич на фото), трансформер класифікації якості фотографії (розроблений у межах даної роботи) та алгоритмічним розрахункам (ефективність фільтрів, якість результатів проходження психологічного тесту).

Також, розглядалася можливість додавання рекомендацій щодо створення позитивного емоційного нахилу фотографій, проте даний підхід має контраргументи щодо свободи вираження особистості, тому на момент написання даної роботи, такі рекомендації не були включені до аналізатора.

3.4.3. Рекомендаційні системи

Окрім представлених вище у даній роботі, було проведено ще декілька досліджень з метою виявлення способів покращення додатку для знайомств та пошуку друзів за допомогою використання засобів машинного навчання. На відміну від вищезгаданих, результати аналізу показали недоцільність їх впровадження через такі завади як низька ефективність, складність розробки або висока обчислювальна складність, що призводить до небажаного збільшення часу відгуку ресурсу. Розглянемо ці дослідження у даному пункті.

Підвищення якості рекомендацій людей у стрічці додатку для знайомств та пошуку друзів є одним із перших шляхів покращення, що спали на думку. На цю тему було проведено дослідження можливих реалізацій, аналіз існуючих алгоритмів та підходів. Результати дослідження були опубліковані у вигляді тези на XVII Всеукраїнській науково практичній WEB конференції аспірантів, студентів та молодих вчених у 2024 році[4].

У тексті розглядалося використання рекомендаційних систем у сфері онлайн-знайомств, які допомагають користувачам ефективно знаходити потенційних партнерів серед великої кількості профілів. Основну увагу приділено трьом підходам: алгоритмічним системам, колаборативній фільтрації та нейронним мережам.

Алгоритмічні системи сортують користувачів за схожістю на основі певних критеріїв, таких як інтереси чи рівень освіти. Вони прості у використанні й не потребують великої кількості даних на початку, проте мають обмеження через свою статичність і нездатність адаптуватися до змін уподобань користувачів та їх попереднього досвіду.

Системи колаборативної фільтрації групують користувачів за схожістю і формують рекомендації, базуючись на вподобаннях цих груп. Вони забезпечують адаптивність і вдосконалюються завдяки взаємодії з користувачами. Але ці системи стикаються з проблемами, такими як "холодний

старт" (відсутність даних для нових користувачів) і можливі помилки в групуванні.

Нейронні мережі використовуються для глибшого аналізу індивідуальних уподобань і прогнозування відповідних рекомендацій. Вони можуть враховувати додаткові аспекти, як-от аналіз фотографій і текстів, і з часом підвищують якість рекомендацій. Проте вони є ресурсозатратними й вимагають значних обсягів даних для навчання.

Оптимальна рекомендаційна система пропонувалася як гібридний підхід: для нових користувачів використовуються прості алгоритмічні рекомендації, далі активується колаборативна фільтрація, а для досвідчених користувачів із багатою історією взаємодій застосовуються нейронні мережі. Результати мають подаватися з урахуванням рівня «впевненості», залежно від методу.

На даний момент на платформі використовується та успішно працює алгоритмічна рекомендаційна система. Вона використовує інформацію з профілів користувачів, таку як: стать, вік, орієнтацію, результати проходження психологічного тесту, обрані гілки дерева інтересів, цінності та вподобання. На основі цих факторів, з використанням емпірично підібраних вагових коефіцієнтів, розраховуються показники сумісності: особистісна сумісність та сумісність за інтересами. Далі, на основі цих показників, розраховується загальна сумісність на основі обраного режиму додатку (романтичний або дружній) та вагових коефіцієнтів сумісності. Після цього, відфільтровані профілі сортуються за показником загальної сумісності таким чином, що першими відображаються люди із найвищим значенням показника.

На даному етапі розвитку платформи, використання інших методів рекомендаційних систем було визнано недоцільним через низку обґрунтованих причин. Насамперед, існує проблема недостатності даних для навчання більш складних моделей, таких як нейронні мережі чи колаборативна фільтрація. Такі підходи потребують великого обсягу інформації про поведінку користувачів

для досягнення адекватних результатів. Наразі зібраних даних недостатньо для забезпечення надійної роботи цих систем, що може призвести до зниження якості рекомендацій.

Крім того, алгоритмічна система, яка використовується на платформі, ще не досягла своєї максимальної ефективності та знаходиться на стадії активного вдосконалення. Її поступове покращення дозволяє виявляти і вирішувати поточні проблеми, що сприяє створенню більш стійкої основи для подальшого переходу до складніших підходів. Поспішний перехід до нових технологій може не лише перервати цей процес, а й вплинути на стабільність платформи.

Ще одним важливим аспектом є вплив на швидкодію системи. Складні рекомендаційні алгоритми, такі як нейронні мережі, мають високі обчислювальні витрати, що може призвести до затримок у формуванні рекомендацій. Оскільки користувачі очікують отримувати результати у межах пів секунди, будь-яке уповільнення негативно позначиться на їхньому досвіді.

У майбутньому розглядається можливість використання ще однієї Vision Transformer моделі для кластеризації користувачів за типами фотографій, що може покращити рекомендації. Проте, на даний момент це є недоцільним.

У зв'язку з цим, поточний алгоритмічний підхід залишається найбільш доцільним варіантом для використання у даних умовах.

Висновки до третього розділу

У третьому розділі кваліфікаційної роботи було здійснено аналіз результатів розробки і застосування засобів машинного навчання для покращення сервісу для знайомств та пошуку друзів. Основна увага приділялась оцінці ефективності реалізованих рішень, їх інтеграції у платформу та визначенню потенційних напрямків для подальшого вдосконалення.

Дослідження класифікатора якості фотографій показало, що отримана модель має високий рівень точності, забезпечуючи якісний автоматичний відбір фотографій, що підвищує загальний рівень контенту платформи. Особливістю

реалізації стало врахування потреб інтеграції моделі у мікросервісну архітектуру, що дозволило забезпечити її ефективну роботу в реальному середовищі.

Результати оцінки процесу верифікації особистості підтвердили відповідність рішення поставленим вимогам, водночас було запропоновано кілька перспективних напрямків для покращення.

Генератор перших повідомлень було оптимізовано для підвищення швидкодії на CPU та адаптовано до багатомовного контексту. Застосування сучасних підходів, таких як оптимізація токенизатора, квантизація моделі та вдосконалення алгоритмів генерації, забезпечило ефективність роботи моделі на CPU.

Додатково досліджено можливості аналізу емоцій на фотографіях, оцінки ефективності профілів користувачів та покращення рекомендаційних систем. Зокрема, було розглянуто інструменти для формування персоналізованих рекомендацій, що допомагають користувачам підвищувати ефективність взаємодії на платформі. Незважаючи на перспективність складніших підходів, було визначено доцільність лише поступового переходу до їх реалізації у майбутньому через обмеження ресурсів та необхідність збереження швидкодії.

Результати третього розділу підтверджують значний потенціал використання машинного навчання для вдосконалення платформ для знайомств та пошуку друзів.

ВИСНОВКИ

У кваліфікаційній роботі освітнього ступеня «магістр» було проведено комплексне дослідження проблемних аспектів сервісів для знайомств, визначено можливості для їх покращення та розроблено програмні рішення для підвищення ефективності та зручності користування платформою.

Було проведено детальний аналіз основних викликів платформ для знайомств, таких як низька якість завантажених фотографій, наявність непристойних фотографій, ненадійність процесів верифікації користувачів та складнощі першої комунікації між користувачами. Виявлено, що ці фактори суттєво впливають на загальну ефективність сервісів та задоволеність користувачів.

Створено та натреновано на власному датасеті модель для автоматичного оцінювання якості фотографій з використанням архітектури Vision Transformer. Ця модель дозволила виявляти неякісні та недоцільні фотографії та повідомляти про це користувачів.

Застосовано модель на базі Vision Transformer для розпізнавання непристойного контенту, а також, виявлення емоцій на фотографіях;

Розроблено механізм автоматизованої верифікації особистості, що базується на аналізі ключових точок обличчя за допомогою бібліотеки FaceMesh. Алгоритм дозволив забезпечувати інтерактивне проходження процесу верифікації в режимі реального часу навіть на мобільних пристроях.

Застосовано мережу MTCNN для виявлення кількості людей (облич) на фотографії та кадрування фотографії по найбільшому обличчю з метою подальшої класифікації іншими моделями.

Застосовано мережу FaceNet (VggFace2) для розрахунку дескриптору обличчя з метою порівняння верифікаційних фотографій з фотографіями, які завантажив користувач.

Створено та натреновано на власному датасеті модель для генерації першого повідомлення з метою полегшення встановлення комунікації між користувачами. Модель базується на сучасній архітектурі GPT і враховує інформацію з профілю співрозмовників для створення персоналізованого повідомлення.

Розглянуто можливості створення рекомендаційних систем для підвищення ефективності та персоналізації платформи.

Розроблені рішення були успішно інтегровані у реальну платформу для знайомств та пошуку друзів, де були протестовані у реальних умовах. Результати показали суттєве покращення середньої якості фотографій профілів, підвищення довіри користувачів до сервісу та зростання кількості активних діалогів після впровадження даних засобів.

Наукова новизна: У роботі запропоновано комплексний підхід до оптимізації сервісів для знайомств із застосуванням сучасних методів машинного навчання, що забезпечує автоматизацію ключових процесів платформи та можуть бути застосовані у різних сервісах схожої направленості.

Практичне значення: Розроблені програмні модулі є готовими до впровадження на реальних платформах для знайомств, сприяючи покращенню якості контенту, автоматизації процесів та підвищенню задоволеності користувачів.

Таким чином, результати роботи дозволяють покращити ефективність сервісів для знайомств, створюючи більш зручний, безпечний та привабливий інструмент для соціальної взаємодії. Отримані розробки можуть бути використані для вдосконалення існуючих платформ, а також для створення нових сервісів із високим рівнем автоматизації та персоналізації.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. «Dating App Revenue and Usage Statistics (2024)», Business of Apps. Дата звернення: 23, Грудень 2024. [Online]. Доступний у: <https://www.businessofapps.com/data/dating-app-market/>
2. Kliuiev I. O., Vakaliuk T. A, «Using Vision Transformer For Automated Image Quality Assessment In Dating Services», представлена на XVIII Міжнародна науково-практична конференція магістрантів та аспірантів, м. Харків, Національний технічний університет „Харківський Політехнічний Інститут”, Лис 2024, с. 66. [Online]. Доступний у: <https://web.kpi.kharkov.ua/masters/wp-content/uploads/sites/135/2024/11/Zbirnyk-tez-TPRYS2024.pdf>
3. Kliuiev I. O. i Vakaliuk T. A, «Methods of face comparison in different photos using computer vision», представлена на Тези XIV Міжнародної науково-технічної конференції «Інформаційно-комп'ютерні технології – 2024», м. Житомир: Житомирська політехніка, 29 2024, с. 17–18. Дата звернення: 26, Грудень 2024. [Online]. Доступний у: <https://conf.ztu.edu.ua/wp-content/uploads/2024/05/17.pdf>
4. Kliuiev I. O. i Vakaliuk T. A, «The use of recommender systems in online dating», представлена на Комп'ютерні інтелектуальні системи та мережі. Матеріали XVII Всеукраїнської науково-практичної WEB конференції аспірантів, студентів та молодих вчених, м. Кривий Ріг: Криворізький національний університет, 28 2024, с. 80–82. Дата звернення: 26, Грудень 2024. [Online]. Доступний у: <https://drive.google.com/file/d/1f9ofpBVYgdeKTFsu79ZMAqNindxpr2Fe/view>
5. Kliuiev I. O. i Vakaliuk T. A, «Gesture recognition methods in photos using computer vision libraries», представлена на Інформаційні технології – 2024: зб. тез XI Всеукраїнської науково-практичної конференції молодих учених,

- м. Київ: Київський столичний університет ім. Б. Грінченка, Трав 2024, с. 122–123. Дата звернення: 26, Грудень 2024. [Online]. Доступний у: <https://zcit.kubg.edu.ua/index.php/journal/issue/view/12/22>
6. Ключев І.О., «Мобільний додаток для знайомств та пошуку друзів», Кваліфікаційна робота бакалавра за спец. 121, Житомирська політехніка, м. Житомир, 2023, с. 123.
 7. «suko/nsfw · Hugging Face». Дата звернення: 10, Грудень 2024. [Online]. Доступний у: <https://huggingface.co/suko/nsfw>
 8. «Thank you for the support of Lobe! The team has loved seeing what the community built with our application and appreciated your interest and feedback. We wanted to share with you that the Lobe desktop application is no longer under development.» Дата звернення: 13, Грудень 2024. [Online]. Доступний у: <https://github.com/lobe>
 9. Liu Z., Mao H., Wu C.-Y., Feichtenhofer C., Darrell T., i Xie S., «A ConvNet for the 2020s», arXiv.org. Дата звернення: 02, Лютий 2024. [Online]. Доступний у: <https://arxiv.org/abs/2201.03545v2>
 10. Zhang K., Zhang Z., Li Z., i Qiao Y., «Joint face detection and alignment using multitask cascaded convolutional networks», *IEEE signal processing letters*, вип. 23, вип. 10, с. 1499–1503, 2016.
 11. Steiner A., Kolesnikov A., Zhai X., Wightman R., Uszkoreit J., i Beyer L., «How to train your ViT? Data, Augmentation, and Regularization in Vision Transformers», 23, Червень 2022, *arXiv*: arXiv:2106.10270. doi: 10.48550/arXiv.2106.10270.
 12. Chen X., Hsieh C.-J., i Gong B., «When Vision Transformers Outperform ResNets without Pretraining or Strong Data Augmentations», *arXiv preprint arXiv:2106.01548*, 2021.
 13. Dosovitskiy A., Beyer L., Kolesnikov A., Weissenborn D., Zhai X., Unterthiner T., Dehghani M., Minderer M., Heigold G., Gelly S., Uszkoreit J., i Houlsby N.,

- «An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale». 2021.
14. «google/vit-base-patch16-224-in21k · Hugging Face». Дата звернення: 29, Травень 2024. [Online]. Доступний у: <https://huggingface.co/google/vit-base-patch16-224-in21k>
 15. Irandoust S., Durand T., Rakhmangulova Y., Zi W., i Hajimirsadeghi H., «Training a Vision Transformer from scratch in less than 24 hours with 1 GPU», 09, Листопад 2022, *arXiv*: arXiv:2211.05187. doi: 10.48550/arXiv.2211.05187.
 16. Wu B., Xu C., Dai X., Wan A., Zhang P., Yan Z., Tomizuka M., Gonzalez J., Keutzer K., i Vajda P., «Visual Transformers: Token-based Image Representation and Processing for Computer Vision». 2020.
 17. Moon H.-M., Shin J., Shin J., i Pan S. B., «User authorization method based on face recognition for auto network access in home network system», *Research Briefs on Information and Communication Technology Evolution*, вип. 1, с. 170–182, 2015.
 18. Lugaresi C., Tang J., Nash H., McClanahan C., Uboweja E., Hays M., Zhang F., Chang C.-L., Yong M. G., Lee J., Chang W.-T., Hua W., Georg M., i Grundmann M., «MediaPipe: A Framework for Building Perception Pipelines», 14, Червень 2019, *arXiv*: arXiv:1906.08172. doi: 10.48550/arXiv.1906.08172.
 19. Veluri R. K., Sree S. R., Vanathi A., Aparna G., i Vaidya S. P., «Hand Gesture Mapping Using MediaPipe Algorithm», представлена на Proceedings of Third International Conference on Communication, Computing and Electronics Systems: ICCCES 2021, Springer, 2022, с. 597–614.
 20. Vaswani A., «Attention is all you need», *Advances in Neural Information Processing Systems*, 2017.
 21. Radford A., Wu J., Child R., Luan D., Amodei D., i Sutskever I., «Language Models are Unsupervised Multitask Learners», 2019.

22. Deng J., Dong W., Socher R., Li L.-J., Li K., i Fei-Fei L., «Imagenet: A large-scale hierarchical image database», в *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, с. 248–255.
23. Ali-Maq, *Ali-Maq/Food-Classification-Volume-Estimation-and-Calorie-Estimation*. (15, Липень 2023). Python. Дата звернення: 29, Травень 2024. [Online]. Доступний у: <https://github.com/Ali-Maq/Food-Classification-Volume-Estimation-and-Calorie-Estimation>
24. Imambi S., Prakash K. B., i Kanagachidambaresan G., «PyTorch», *Programming with TensorFlow: Solution for Edge Computing Applications*, с. 87–104, 2021.
25. «🤗 Transformers». Дата звернення: 12, Грудень 2024. [Online]. Доступний у: <https://huggingface.co/docs/transformers/index>
26. Sanders J. i Kandrot E., *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.
27. Tanaka A., Tomiya A., Hashimoto K., Tanaka A., Tomiya A., i Hashimoto K., «Basics of neural networks», *Deep Learning and Physics*, с. 35–55, 2021.
28. «Falconsai/nsfw_image_detection · Hugging Face». Дата звернення: 10, Грудень 2024. [Online]. Доступний у: https://huggingface.co/Falconsai/nsfw_image_detection
29. «LukeJacob2023/nsfw-image-detector · Hugging Face». Дата звернення: 10, Грудень 2024. [Online]. Доступний у: <https://huggingface.co/LukeJacob2023/nsfw-image-detector>
30. Komajou H., «Mediapipe landmark face/hand/pose sequence number list view», Medium. Дата звернення: 28, Липень 2024. [Online]. Доступний у: <https://medium.com/@hotakoma/mediapipe-landmark-face-hand-pose-sequence-number-list-view-778364d6c414>
31. Viola P. i Jones M., «Rapid object detection using a boosted cascade of simple features», представлена на Proceedings of the 2001 IEEE computer society

- conference on computer vision and pattern recognition. CVPR 2001, Ieee, 2001, с. I–I.
32. Amit Y., Geman D., i Wilder K., «Joint induction of shape features and tree classifiers», *IEEE transactions on pattern analysis and machine intelligence*, вип. 19, вип. 11, с. 1300–1305, 1997.
 33. Dalal N. i Triggs B., «Histograms of oriented gradients for human detection», представлена на 2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05), Ieee, 2005, с. 886–893.
 34. Gradilla R., «Multi-task Cascaded Convolutional Networks (MTCNN) for Face Detection and Facial Landmark Alignment», Medium. Дата звернення: 05, Лютий 2024. [Online]. Доступний у: <https://medium.com/@iselagradilla94/multi-task-cascaded-convolutional-networks-mtcnn-for-face-detection-and-facial-landmark-alignment-7c21e8007923>
 35. «A Deep Dive Into Non-Maximum Suppression (NMS) | Built In». Дата звернення: 05, Лютий 2024. [Online]. Доступний у: <https://builtin.com/machine-learning/non-maximum-suppression>
 36. Cevikalp H., Neamtu M., Wilkes M., i Barkana A., «Discriminative common vectors for face recognition», *IEEE Transactions on pattern analysis and machine intelligence*, вип. 27, вип. 1, с. 4–13, 2005.
 37. Jia X. i Nixon M. S., «Extending the feature vector for automatic face recognition», *IEEE Transactions on Pattern Analysis and Machine Intelligence*, вип. 17, вип. 12, с. 1167–1176, 1995.
 38. «How does face recognition work? | Nuwiz». Дата звернення: 05, Лютий 2024. [Online]. Доступний у: <https://nuwiz.io/blog/face-recognition>
 39. Xu M., Chen D., i Zhou G., «Real-time face recognition based on Dlib», представлена на Innovative Computing: IC 2020, Springer, 2020, с. 1451–1459.

40. Suwarno S. i Kevin K., «Analysis of face recognition algorithm: Dlib and opencv», *Journal of Informatics and Telecommunication Engineering*, вип. 4, вип. 1, с. 173–184, 2020.
41. «Papers with Code - Labeled Faces in the Wild Benchmark (Face Verification)». Дата звернення: 05, Лютий 2024. [Online]. Доступний у: <https://paperswithcode.com/sota/face-verification-on-labeled-faces-in-the>
42. «Papers with Code - VGGFace2 Dataset». Дата звернення: 05, Лютий 2024. [Online]. Доступний у: <https://paperswithcode.com/dataset/vggface2-1>
43. «Blog – Ars Futura - Face Recognition with FaceNet and MTCNN». Дата звернення: 05, Лютий 2024. [Online]. Доступний у: <https://arsfutura.com/blog/face-recognition-with-facenet-and-mtcnn>
44. «Safetensors». Дата звернення: 12, Грудень 2024. [Online]. Доступний у: <https://huggingface.co/docs/safetensors/index>
45. Sanh V., Debut L., Chaumond J., i Wolf T., «DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter», в *NeurIPS EMC² Workshop*, 2019.
46. Tang J., Shivanna R., Zhao Z., Lin D., Singh A., Chi E. H., i Jain S., «Understanding and Improving Knowledge Distillation». 2021. [Online]. Доступний у: <https://arxiv.org/abs/2002.03532>
47. Black S., Leo G., Wang P., Leahy C., i Biderman S., *GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow*. (Березень 2021). Zenodo. doi: 10.5281/zenodo.5297715.
48. Brown T. B., Mann B., Ryder N., Subbiah M., Kaplan J., Dhariwal P., Neelakantan A., Shyam P., Sastry G., Askell A., Agarwal S., Herbert-Voss A., Krueger G., Henighan T., Child R., Ramesh A., Ziegler D. M., Wu J., Winter C., Hesse C., Chen M., Sigler E., Litwin M., Gray S., Chess B., Clark J., Berner C., McCandlish S., Radford A., Sutskever I., i Amodei D., «Language Models are Few-Shot Learners». 2020. [Online]. Доступний у: <https://arxiv.org/abs/2005.14165>

49. Schäfer A., Ravfogel S., Hofmann T., Pimentel T., i Schlag I., «The Role of Language Imbalance in Cross-lingual Generalisation: Insights from Cloned Language Experiments». 2024. [Online]. Доступный у: <https://arxiv.org/abs/2404.07982>
50. «What is W&B? | Weights & Biases Documentation». Дата звернення: 23, Грудень 2024. [Online]. Доступний у: <https://docs.wandb.ai/guides/>
51. Team G., «Gradio Documentation». Дата звернення: 29, Травень 2024. [Online]. Доступний у: <https://www.gradio.app/docs>
52. «Quickstart — Flask Documentation (3.0.x)». Дата звернення: 29, Травень 2024. [Online]. Доступний у: <https://flask.palletsprojects.com/en/3.0.x/quickstart/>
53. Gage P., «A new algorithm for data compression», *The C Users Journal*, вип. 12, вип. 2, с. 23–38, 1994.
54. developers O. R., «ONNX Runtime». 2021. [Online]. Доступний у: <https://onnxruntime.ai/>
55. «🤗 Optimum». Дата звернення: 12, Грудень 2024. [Online]. Доступний у: <https://huggingface.co/docs/optimum/index>
56. «dima806/face_emotions_image_detection · Hugging Face». Дата звернення: 14, Грудень 2024. [Online]. Доступний у: https://huggingface.co/dima806/face_emotions_image_detection
57. «Face emotions image detection ViT». Дата звернення: 14, Грудень 2024. [Online]. Доступний у: <https://kaggle.com/code/dima806/face-emotions-image-detection-vit>

ДОДАТКИ

Поворотна нормалізація ключових точок

```
1 export const getHeadRotation = (landmarks: NormalizedLandmarks) => {
2   const leftEyeOuter = landmarks.eyeLeft.left;
3   const rightEyeOuter = landmarks.eyeRight.right;
4   const noseTip = landmarks.nose.tip;
5   const earLeft = landmarks.ear.left;
6   const earRight = landmarks.ear.right;
7
8   const eyeCenter = {
9     x: (leftEyeOuter.x + rightEyeOuter.x) / 2,
10    y: (leftEyeOuter.y + rightEyeOuter.y) / 2,
11    z: (leftEyeOuter.z + rightEyeOuter.z) / 2
12  };
13
14  const yaw = -angle3D(earRight, earLeft).xz;
15  const pitch = -(angle3D(eyeCenter, noseTip).yz + 60);
16  const roll = -angle3D(earRight, earLeft).xy;
17
18  return { yaw, pitch, roll };
19 };
20
21 export const getRotationNormalizedLandmarks = (lmk: Landmarks) => {
22   const { pitch, yaw, roll } =
23   getHeadRotation(getNormalizedLandmarks(lmk));
24
25   const rotatePitch = (point: Point, angle: number): Point => {
26     const rad = (angle * Math.PI) / 180.0;
27     const cosAngle = Math.cos(rad);
28     const sinAngle = Math.sin(rad);
29     return {
30       x: point.x,
31       y: point.y * cosAngle - point.z * sinAngle,
32       z: point.y * sinAngle + point.z * cosAngle,
33     };
34   };
35
36   const rotateYaw = (point: Point, angle: number): Point => {
37     const rad = (angle * Math.PI) / 180.0;
38     const cosAngle = Math.cos(rad);
39     const sinAngle = Math.sin(rad);
40     return {
41       x: point.x * cosAngle + point.z * sinAngle,
```

```

42     z: -point.x * sinAngle + point.z * cosAngle,
43   };
44 };
45
46 const rotateRoll = (point: Point, angle: number): Point => {
47   const rad = (angle * Math.PI) / 180.0;
48   const cosAngle = Math.cos(rad);
49   const sinAngle = Math.sin(rad);
50   return {
51     x: point.x * cosAngle - point.y * sinAngle,
52     y: point.x * sinAngle + point.y * cosAngle,
53     z: point.z,
54   };
55 };
56
57 const rotate = (point: Point): Point => {
58   const rotatedX = rotatePitch(point, pitch);
59   const rotatedXY = rotateYaw(rotatedX, -yaw);
60   const rotatedXYZ = rotateRoll(rotatedXY, roll);
61   return rotatedXYZ;
62 };
63
64 return lmk.map(rotate);
65 }

```

Рис. 1. Код поворотної нормалізації

Системні запити для створення датасету

Мова	Контекст	Модель	Запит
Англійська	Дружній	Gemini Advanced	Create a short and creative hello message to the user based on common interests and details. Pick the most rare common interest to discuss. It should be just a few sentences. Ask an open question in the end. Mention location only if they have different locations and common birth location.
Англійська	Романтичний	Gemini	You are a dating service assistant. You help the user to create first message in a chat. Create a short and creative hello message to the user as a possible future romantic partner. Use flirt to intrigue the user. Also you might use interests and profile details (such as hair color) to make the message more personal. Don't stoop to outright flattery or self-abasement. It should be just a few sentences. Ask an open question in the end. Mention location only if they have different locations and common birth location. Do not use the word "fellow" in your message.
Українська	Дружній	Gemini	Ти асистент на сервісі пошуку друзів. Ти допомагаєш користувачеві написати перше повідомлення в чаті. Напиши коротке і креативне привітання українською мовою для користувача (user) на основі спільних інтересів та цінностей. Це має бути лише кілька речень, додай відкрите запитання у кінець. Використовуй інформацію про статі - свою (me) та людини (user) - для правильного формулювання речення.
Українська	Романтичний	Gemini	Ти асистент на сервісі знайомств. Ти допомагаєш користувачеві написати перше повідомлення в чаті. Напиши коротке і креативне привітання українською мовою для користувача (user) як можливого майбутнього романтичного партнера. Використовуй флірт, щоб заінтригувати. Можеш використати інтереси та деталі профілю (наприклад, колір волосся), щоб зробити повідомлення більш персоналізованим. Не вдавайся до відвертих лестощів або самоприниження. Це має бути лише кілька речень, додай відкрите запитання.

Рис. 2. Системні запити, що використовувалися для створення датасету

Проекція розміщення ключових точок бібліотеки FaceMesh

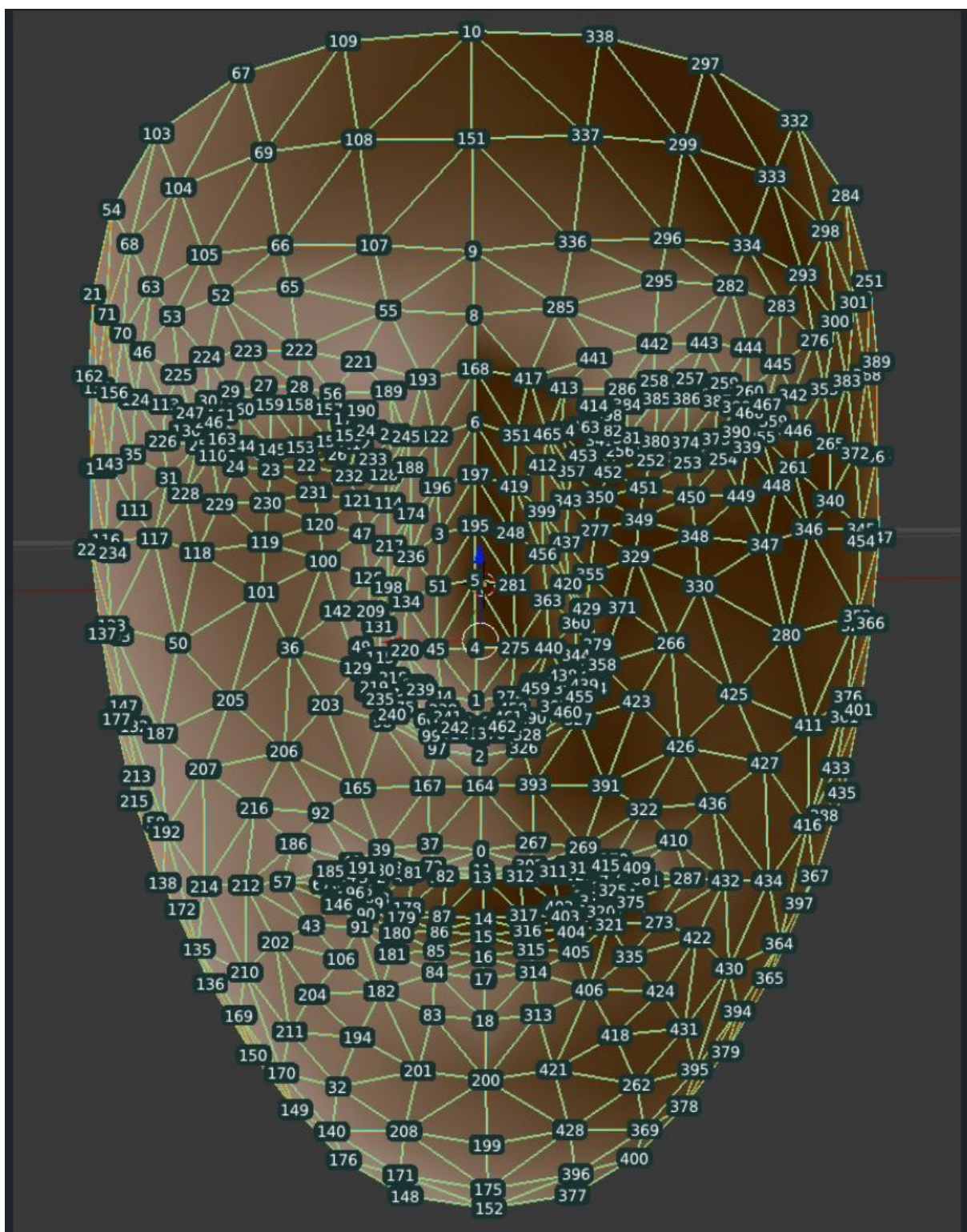


Рис. 3. Мапа ключових точок FaceMesh