ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Методика аналізу рукописних текстів для перевірки на помилки»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

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

_____ Максим БАЙСА
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-62
Максим БАЙСА

Керівник: Богдан ХУДІК
доктор філософії (PhD)

Рецензент: _____
науковий ступінь, вчене звання *Ім'я, ПРІЗВИЩЕ*

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Байсі Максиму Юрійовичу

1. Тема кваліфікаційної роботи: «Методика аналізу рукописних текстів для перевірки на помилки»
керівник кваліфікаційної роботи Богдан ХУДІК, доктор філософії (PhD),
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «30» жовтня 2025 р. № 467.
2. Строк подання кваліфікаційної роботи «19» грудня 2025 р.
3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, зображення рукописних текстів, вимоги до точності розпізнавання, вимоги до точності пошуку помилок.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   1. Дослідження методів розпізнавання рукописного тексту, попередньої обробки та граматичної перевірки тексту.
   2. Розробка математичної моделі розпізнавання тексту, попередньої обробки та граматичної перевірки тексту.
   3. Розробка методики розпізнавання рукописного тексту та його граматичної перевірки з використанням попередньої обробки тексту – МРОРТ.

4. Розробка мобільного застосунку із використанням запропонованої методики МРОРТ.
5. Проведення експериментального дослідження та порівняльного аналізу МРОРТ та аналогів за критеріями часу роботи, точності розпізнавання тексту та точності граматичної перевірки.
6. Висновки, рекомендації щодо впровадження та подальших досліджень.

5. Перелік ілюстративного матеріалу: *презентація*
1. Методика розпізнавання та граматичної перевірки МРОРТ
2. Математична модель МРОРТ.
3. Алгоритм розпізнавання та граматичної перевірки тексту.
4. Попередня обробка тексту.
5. Діаграма класів мобільного застосунку.
6. Екранні форми мобільного застосунку.
7. Порівняльний аналіз.

6. Дата видачі завдання «31» жовтня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	31.10-04.11.2025	
2	Вивчення методів розпізнавання рукописного тексту та граматичної перевірки	05.11-10.11.2025	
3	Розробка математичної моделі МРОРТ	11.11-13.11.2025	
4	Розробка методики розпізнавання рукописного тексту та граматичної перевірки із застосування попередньої обробки тексту (МРОРТ)	14.11-24.11.2025	
5	Розробка мобільного застосунку із застосуванням МРОРТ	25.11-05.12.2025	
6	Проведення експериментального дослідження та порівняльного аналізу	06.12-10.12.2025	
7	Оформлення роботи: вступ, висновки, реферат	11.12-17.12.2025	
8	Розробка демонстраційних матеріалів	18.12-19.12.2025	

Здобувач вищої освіти _____ Максим БАЙСА
(підпис)

Керівник
кваліфікаційної роботи _____ Богдан ХУДІК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 75 стор., 12 табл., 34 рис., 33 джерел.

Мета роботи – автоматизація пошуку помилок у тексті, написаному від руки, за рахунок поєднання різних методів розпізнавання та обробки тексту.

Об'єкт дослідження – процес розпізнавання рукописного тексту та пошуку помилок у ньому.

Предмет дослідження – технології розпізнавання рукописного тексту, методи обробки природньої мови та пошуку помилок у тексті.

У роботі використано різноманітні методи, такі як розпізнавання тексту за допомогою сервісу OCRSpace, граматична перевірка із сервісом LanguageTool, нормалізація тексту, визначення меж речень, забезпечення завершення речень, виправлення OCR артефактів.

Проведено аналіз сучасних методів розпізнавання рукописних текстів та перевірки на помилки.

Розроблено методику МРОРТ, математичну модель та мобільний застосунок.

Проведено експерименти та порівняльний аналіз із сформуванням висновків, які доводять практичну важливість роботи.

КЛЮЧОВІ СЛОВА: OCR, ГРАМАТИЧНА ПЕРЕВІРКА, NLP, ПОПЕРЕДНЯ ОБРОБКА ТЕКСТУ, ШІ, ANDROID, КОМП'ЮТЕРНИЙ ЗІР.

ABSTRACT

Text part of the master's qualification work: 75 pages, 34 pictures, 12 tables, 33 sources.

The purpose of the work – automation of error detection in handwritten text through the combination of various text recognition and processing methods.

Object of research – the process of recognizing handwritten text and detecting errors in it.

Subject of research – handwritten text recognition technologies, natural language processing methods and text error-detection techniques.

The work employs a variety of methods, including text recognition using the OCRSpace service, grammatical checking with LanguageTool, text normalization, sentence boundary detection, sentence completion and OCR artifacts correction.

An analysis of modern handwritten text recognition methods and error-detection techniques has been conducted.

The MPOPT methodology, a mathematical model and a mobile application have been developed.

Experiments and comparative analysis were carried out, resulting in conclusions that demonstrated the practical significance of the work.

KEYWORDS: OCR, GRAMMAR CHECK, NLP, TEXT PREPROCESSING, AI, ANDROID, COMPUTER VISION.

ЗМІСТ

ВСТУП

Розпізнавання тексту, написаного від руки – актуальна тема. Попри значне оцифрування у сучасному світі, багато де потрібно писати текст від руки. Це включає освіту, написання документів, ведення щоденників, запис нотаток та інше. Найпростіший приклад – написання тексту під час проведення уроку з української мови у школі. Або ж ведення особистого щоденника.

Також за довгий проміжок часу, коли люди не використовували друкарські машини, або цифрові пристрої, накопичилася величезна кількість документів, книг та інших фізичних носіїв рукописних текстів. Тільки на європейському порталі культурної спадщини знаходить більше 466 тисяч манускриптів [1]. А ще є велика кількість бібліотек та інших закладів зберігання документів. Серед таких закладів є Інститут рукопису Національної бібліотеки України імені В. І. Вернадського, в якому лише рукописних книг зберігається понад 8 тисяч [2].

Проте на етапі розпізнавання рукопису виникають значні труднощі. Варіативність почерків, якість зображень, особливості мови – усе це значно впливає на якість розпізнавання тексу [3].

Тому важливо розвивати та вдосконалювати технології розпізнавання та обробки рукописів. Адже це допоможе швидко оцифрувати безліч документів та манускриптів, книг, а використовуючи приклад з навчанням у школі, це допоможе учителям сфокусуватися на поясненні матеріалу і помилок, які допускають учні, замість самого пошуку цих помилок.

Для зручності, можна імплементувати алгоритм розпізнавання тексту та обробки у мобільному застосунку. Це дозволить легко перевірити якість його роботи.

Мета роботи – автоматизація пошуку помилок у тексті, написаному від руки, за рахунок поєднання різних методів розпізнавання та обробки тексту.

Об'єкт дослідження – процес розпізнавання рукописного тексту та пошуку помилок у ньому.

Предмет дослідження – технології розпізнавання рукописного тексту, методи обробки природньої мови та пошуку помилок у тексті.

Для досягнення мети вирішувалися наступні задачі:

1. Проведення огляду та аналізу існуючих підходів до розпізнавання та граматичної перевірки рукописного тексту.
2. Розробка математичної моделі розпізнавання рукописного тексту, попередньої обробки тексту та його граматичної перевірки.
3. Розробка методики розпізнавання рукописного тексту, попередньої обробки тексту та його граматичної перевірки (МРОРТ).
4. Розробка мобільного застосунку із використанням МРОРТ для автоматизації перевірки рукописного тексту на помилки.
5. Проведення експериментального дослідження та виконання порівняльний аналізу з існуючими рішеннями.
6. Формулювання висновків та рекомендацій щодо подальших досліджень.

Практичне значення має розроблений застосунок при використанні педагогами, для самоосвіти, або спеціалістам, які працюють з документами для швидкого пошуку помилок у написаних від руки текстах.

Апробація результатів – за темою дослідження опубліковано 2 тези на конференціях.

1 ОГЛЯД ТЕХНОЛОГІЙ РОЗПІЗНАВАННЯ ТЕКСТУ ТА ГРАМАТИЧНОЇ ПЕРЕВІРКИ. АНАЛІЗ НАЯВНИХ МЕТОДІВ

1.1 Технології розпізнавання тексту

OCR (Optical Character Recognition – оптичне розпізнавання тексту) – технологія розпізнавання тексту у зображеннях для конвертації у цифровий тип даних [4]. Типові моделі OCR заснована на виявленні різних елементів у зображенні, збільшенні певних частин зображення, та виявленні символів. Такі моделі досить неточні при, а розпізнавання прописного тексту практично неможливе.

В останні роки значного розвитку зазнали LVLMs (великі оптичні мовні моделі), які сильно підвищили точність розпізнавання тексту. Розвиток і досі продовжується, адже необхідно підвищувати точність розпізнавання, додати підтримку багатьох мов.

На основі OCR розроблено низку інструментів. Деякі з них призначені для розробників: Google ML Kit Text Recognition API, Google Cloud Vision API, Google Cloud Document AI, Microsoft Azure Read API. Дані інструменти дозволяють зчитувати текст, проводити попередню обробку, визначати мову, виконувати переклад.

Google ML Kit – це сервіс на основі машинного навчання, який надає можливість імплементувати функції на основі машинного навчання у мобільні застосунки [5, с. 13]. Сервіс пропонує готові рішення для:

– опису зображення;

– розпізнавання тексту;

– розпізнавання обличчя;

– сканування штрих-кодів;

– перекладу тексту та інших.

Саме розпізнавання тексту важливе для розроблюваного застосунку. Text Recognition API дає змогу визначати текст та його мову різними системами правопису – японська, корейська, китайська, девангарі та латинська системи правопису. Кирилиця не входить до переліку, що означає що українська мова не підтримується. Також API може визначати мову тексту, в тому числі й українську. Список підтримуваних мов можна проглянути в офіційній документації.

Система розпізнавання розділяє текст на сегменти (див. рис. 1.1):

– блок – набір рядків, наприклад абзац;

– ряд – набір слів на одній вісі;

– елемент – набір символів, слово;

– символ.

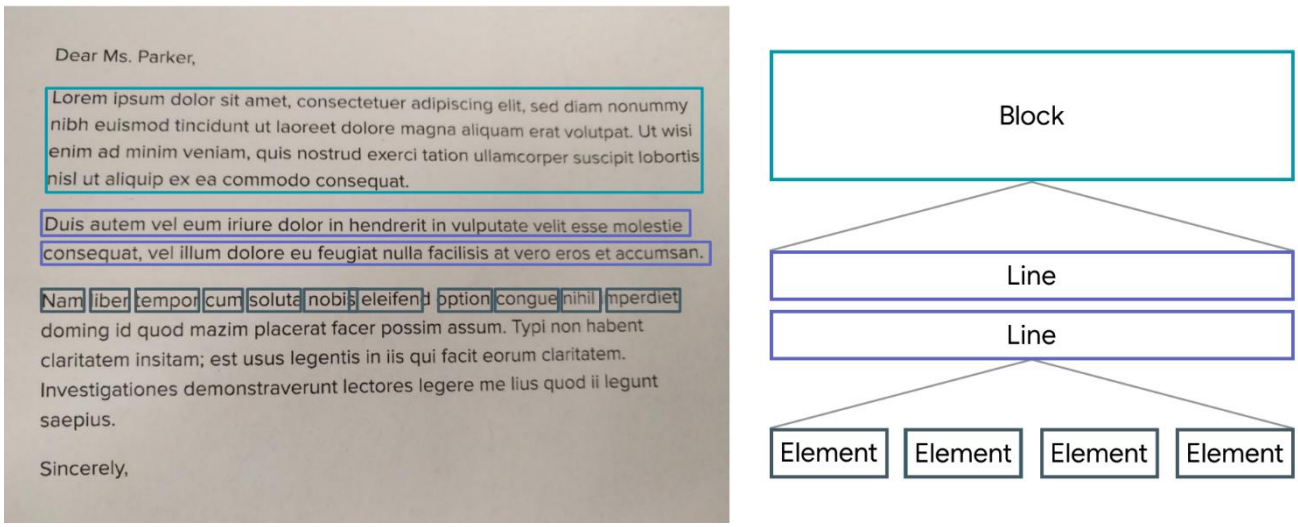


Рис. 1.1 Сегментація тексту

Головним недоліком виступає відсутність повної підтримки української мови. Перевага у тому, що сервіс націлений саме на мобільні пристрої, в тому числі Android, тому інтеграція повинна бути легкою.

Document AI – інструмент платформи Google Cloud, що використовує машинне навчання для аналізу та видобування інформації із структурованих даних в документах, таких як PDF, відскановане зображення та інших форматах [6].

Обробляючи текст, зображення та інші елементи документів, інструмент може визначити основну інформацію, таку як імена, адреси та інше, щоб автоматизувати робочі процеси та створити звіти. У функції Document AI входить оптичне розпізнавання тексту.

Інструмент пропонує декілька моделей:

– Document Understanding AI: сервіс для аналізу і видобування даних з документа використовуючи моделі машинного навчання;

– Form Parser: інструмент для видобутку структурованих даних з форм;

– Procurement DocAI: сервіс для автоматизації закупівель після видобування даних з замовлень та чеків;

– Lending DocAI: сервіс для автоматизації обробки кредитування за допомогою видобутку даних х документів і програмного забезпечення;

– Healthcare DocAI: сервіс для автоматизації процесів, пов’язаних із забезпеченням здоров’я, на основі видобутих даних із медичних звітів, записів та страхових форм.

Також Document AI можна інтегрувати з іншими сервісами платформи Google Cloud.

Read API – OCR інструмент платформи Microsoft Azure (аналог Google Cloud Platform). Сервіс може розпізнавати прописний текст, проте тільки дев’ятьма мовами, серед яких немає української. Нажаль сервіс не має підтримки української мови і для друкованого тексу, попри велику кількість інших підтримуваних мов, в тому числі на кирилиці. [7]

З переваг даного сервісу – швидкодія і велика кількість підтримуваних мов та хороша точність розпізнавання – не менше 95% точності, хоча це залежить від якості зображення і почерку [8].

1.2 Методи обробки та аналізу тексту

Обробка природної мови – важка задача для комп’ютера. Для її виконання розроблено технології NLP (Natural Language Processing). NLP складається з 5 основних компонентів:

– морфологічний аналіз – аналіз внутрішньої структури слова, один із методів обробки природної мови;

– синтаксичний аналіз – це процес аналізу вхідної послідовності символів, з метою розбору граматичної структури згідно із заданою формальною граматикою;

– семантичний аналіз – це процес визначення смислу та інтерпретації значення слів та фраз у контексті, в якому вони вживаються;

– прагматичний аналіз – полягає у вивченні людських вимірів комунікації, пов'язаності їх з мовними структурами;

– дискурсивний аналіз – це аналіз дискурсу як результату (тексту) і процесу мовленнєво-мисленнєвої діяльності з метою виявлення екстралінгвістичних, семантичних, когнітивних і мовленнєвих аспектів його формування, з'ясування функціональної спрямованості і соціальної організації [7].

Кожен із цих компонентів включає під-завдання:

– декомпозиція тексту – виділення основи похідного слова або одного з компонентів основ складного слова та оформлення його в самостійне слово;

– правопис – сукупність загальновизнаних і загальнообов'язкових правил, що встановлюють способи відтворення мовлення та інші прийоми словесної передачі інформації на письмі.;

– морфологічний розбір – розбір, покликаний посприяти засвоєнню і систематизації ознак частин мови, їхніх граматичних категорій, виробленню міцних практичних навичок словозміни іменних слів і дієслова, підготовці до вивчення синтаксису як учення про зв'язки слів у словосполученні й реченні;

– стемінг – процес скорочення слова до основи, шляхом відкидання допоміжних частин, таких як суфікс чи закінчення.

Існують різні NLP інструменти для різних мов програмування або як сервіси: NLTK для Python, OpenNLP для Java, KotlinNLP для Kotlin, Natural Language AI, NLP Cloud, Allen NLP, PyText та інші.

Підвищити якість та глибину обробки природної мови можна за допомогою великої мовної моделі штучного інтелекту. Мовна модель – це тип штучного інтелекту, розроблений для обробки та генерування природної мови [9]. Мовна модель можна назвати статистичною або машинною моделлю, що має можливість передбачати ймовірність наступних слів у послідовності, виходячи з попереднього контексту. Мовна модель заснована на архітектурі нейронних мереж.

Найпопулярніші мовні моделі:

– GPT (Generative Pre-trained Transformer);

– BERT (Bidirectional Encoder Representations from Transformers);

– Gemini;

– LLaMA (Large Language Model Meta AI).

Для використання розробниками цих моделей доступні API.

Google Cloud Natural Language API дозволяє розробникам легко виконувати обробку природної мови в їхньому програмному забезпеченні, включаючи аналіз синтаксису, сутностей, семантичний аналіз, класифікація вмісту та інше [10]. Даний API підтримує основні мови програмування, такі як:

– C#;

– Go;

– Java;

– Node.js;

– PHP;

– Python;

– Ruby та інші.

Apache OpenNLP – відкрита Java бібліотека для обробки природної мови.

Можливості цієї бібліотеки:

– виявлення речення: полягає у виявленні початку і кінця речення;

– токенізація: полягає у поділі речення на менші частини – токени (зазвичай це слова, числа або знаки пунктуації);

– розпізнавання сутності: полягає у знаходженні названих у тексті сутностей, таких як люди, місця, організації та інше;

– ідентифікація типу слова (іменник, дієслово, прикметник, однина/множина);

– лематизація;

– поділ слів за їх типами;

– визначення мови у тексті [11].

NLP Cloud [12] – сервіс на основі штучного інтелекту який пропонує хороший набір інструментів для обробки природної мови: аналіз настроїв, NER, класифікація текстів, тегування частин мови та розбір залежностей, відповіді на запитання, стислий виклад тексту, генерація тексту, переклад, розпізнавання мови, токенізація, лематизація. Перевагою NLP Cloud над Google Cloud Natural Language API є політика ціноутворення, де є більше простору для тестування, розробки та дослідження. Також, як і в інших сервісах в NLP Cloud можна налаштувати власну модель або підключити уже налаштовану.

1.3 Вплив штучного інтелекту

За останні роки значно розвинулися технології штучного інтелекту. Окрім того, нейронні мережі, мовні моделі, машинне навчання посилюють інструменти у інших сферах, в тому числі і в OCR та в обробці тексту. Тому багато сучасних програм використовують штучний інтелект для посилення своїх функцій. Серед компаній, які активно впроваджують штучний інтелект є і такі гіганти, як Google, Microsoft, Apple та інші. Найвідоміші продукти:

– ChatGPT;

– Gemini;

– Copilot;

– Grok та інші.

Деякі компанії відкривають доступ до своїх мовних моделей. Серед таких компаній OpenAI – розробник ChatGPT. OpenAI API надає доступ до мовної моделі GPT та багатьох інших можливостей, включно з розпізнаванням тексту (OCR). Для цього OpenAI пропонує низку спеціалізованих моделей, кожна з яких має власні функції: генерація тексту, виклик функцій, структуровані та обдумані відповіді, OCR, створення зображень, перетворення аудіо в текст і навпаки, векторизація слів, а також модерація контенту [13].

Google в свою чергу надає свої інструменти для розробників [14]:

1) Gemini;

2) Imagen;

3) Veo;

4) Gemma.

1.4 Огляд аналогів

Існує різне програмне забезпечення для зчитування, аналізу та подальшої обробки тексту. Деяким програмним забезпечення багато хто уже користувався. Онлайн перекладачі, такі Google Translate, мають функції зчитування тексту, в тому числі написаного від руки. Також є багато різних застосунків-сканерів, або інших застосунків які мають такий функціонал, що уміють зчитувати тексти. До таких застосунків належать Google Keep, Microsoft OneNote, CamScanner, Adobe Scan, TextGrabber і навіть звичаний застосунок-камера Samsung Camera.

Серед програмного забезпечення, що аналізує текст є Grammarly, LanguageTool, ProWritingAid, SlickWrite. Дане програмне забезпечення аналізує

текст, та шукає в ньому граматичні, стилістичні, орфографічні помилки та пропонує як їх вирішити.

Стандартний перед-встановлений застосунок для керування камерою мобільного пристрою Samsung, який після пакету оновлень зі штучним інтелектом отримав можливість сканувати зображення для створення приміток, копіювання тексту на зображенні або створення документу. Підтримує українську та англійську мови при скануванні тексту, а також може зчитувати рукописний текст, хоча і з помилками.

На рисунку 1.2 продемонстровано зчитування тексту, написаного від руки українською мовою у друкованому на прописному варіантах. У результаті застосунок безпомилково зчитав та скопіював текст до приміток.

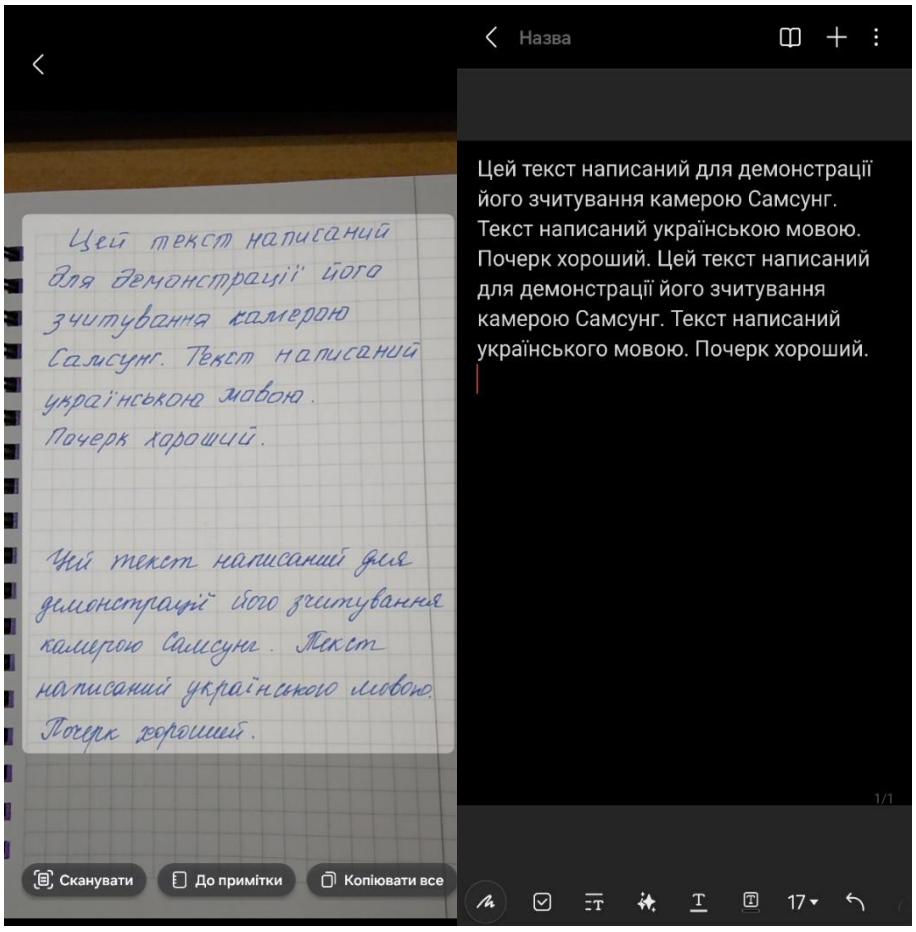


Рис. 1.2 Сканування тексту з Samsung Camera

CamScanner – застосунок для сканування документів. У функціонал входить сканування зображення, покращення скану за допомогою штучного інтелекту, створення нотаток та документів, витягування тексту зі скану [15].

Нажаль CamScanner не зміг розпізнати українську мову. Після ручного встановлення мови застосунок не зміг коректно розпізнати текст і вивів невідомий текст латиницею (див. рис. 1.3). При скануванні прописного тексту англійською застосунок зміг встановити мову, проте допустився багатьох помилок у словах. Найкращого результату досяг при скануванні друкованого тексту англійською, де майже не допустився помилок (див . рис. 1.4).

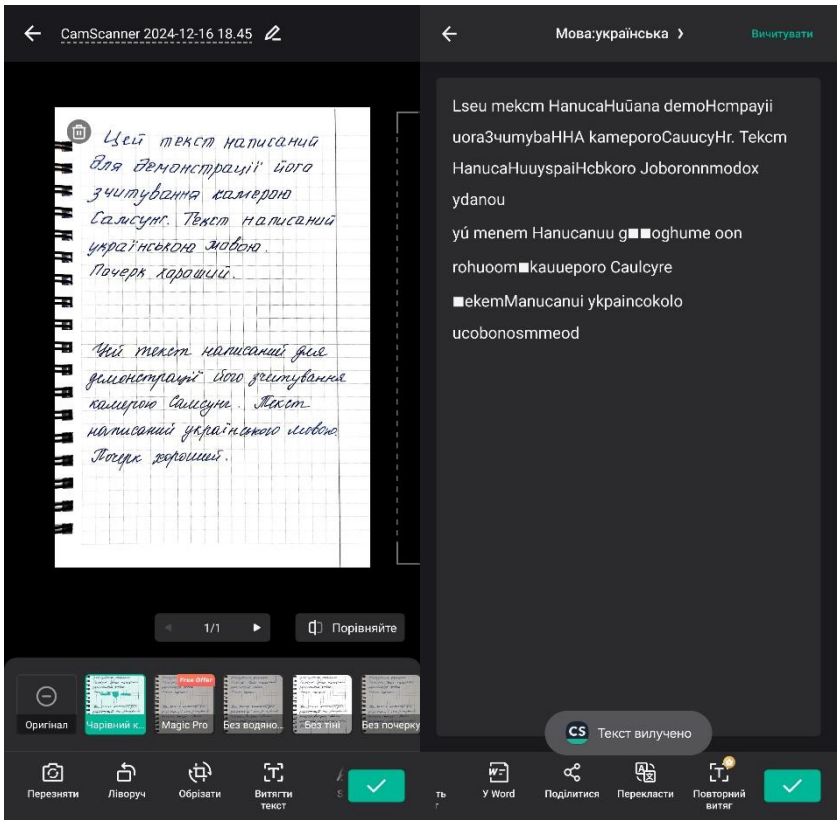


Рис. 1.3 Сканування української з CamScanner

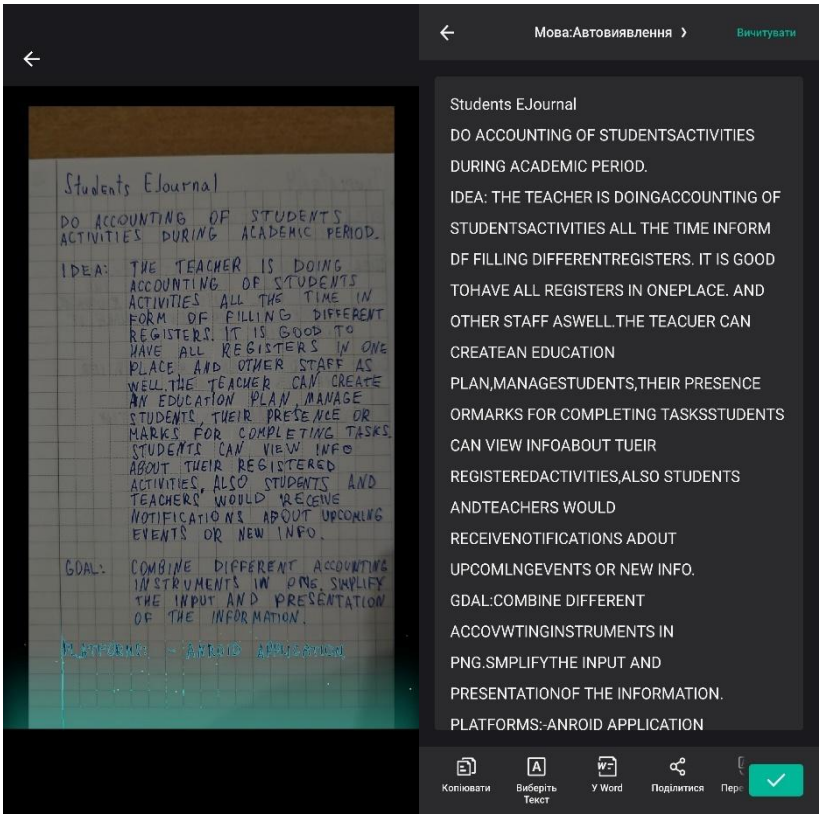


Рис. 1.4 Сканування англійської з CamScanner

Adobe Scan – ще один застосунок для сканування документів. Відскановані документи можна зберігати у форматі PDF, також можна скопіювати текст, який в документів [16]. При спробі зчитати уже використаний текст українською до буферу обміну копіювався текст з випадковими символами латиницею. Текст друкованою англійською застосунок розпізнав, проте зробив набагато більше помилок чим CamScanner (див. рис. 1.5).

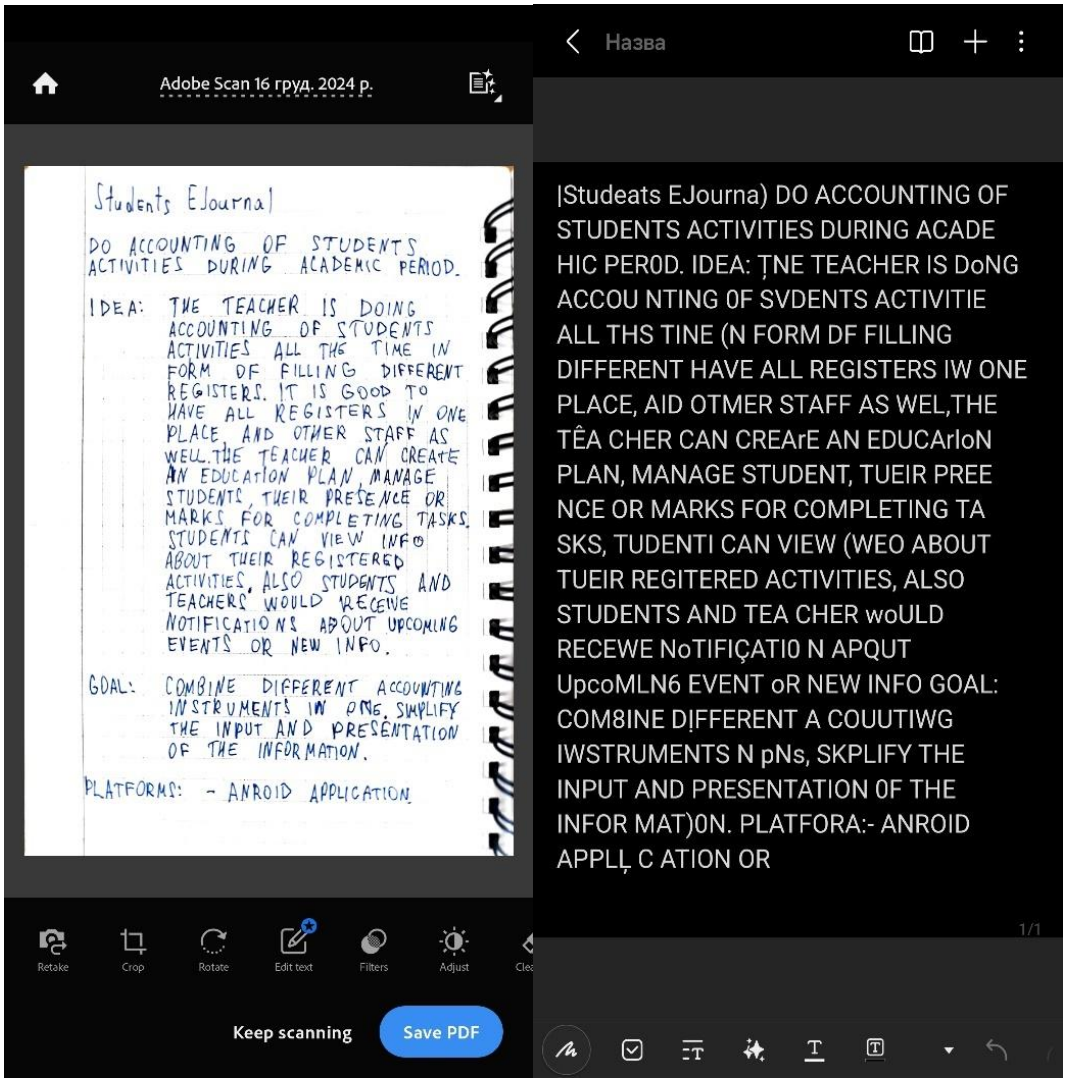


Рис. 1.5 Сканування англійської з Adobe Scan

Grammarly – популярний інструмент для перевірки тексту на помилки. Позиціонується як асистент для написання тексту: шукає та виправляє помилки, пропонує кращу структури речення допомагає писати листи, має інтеграцію з багатьма популярними застосунками [17].

Вставимо простий текст англійською і подивимося на результат (див. рис. 1.6). Застосунок одразу пропонує виправлення на основі свого аналізу.

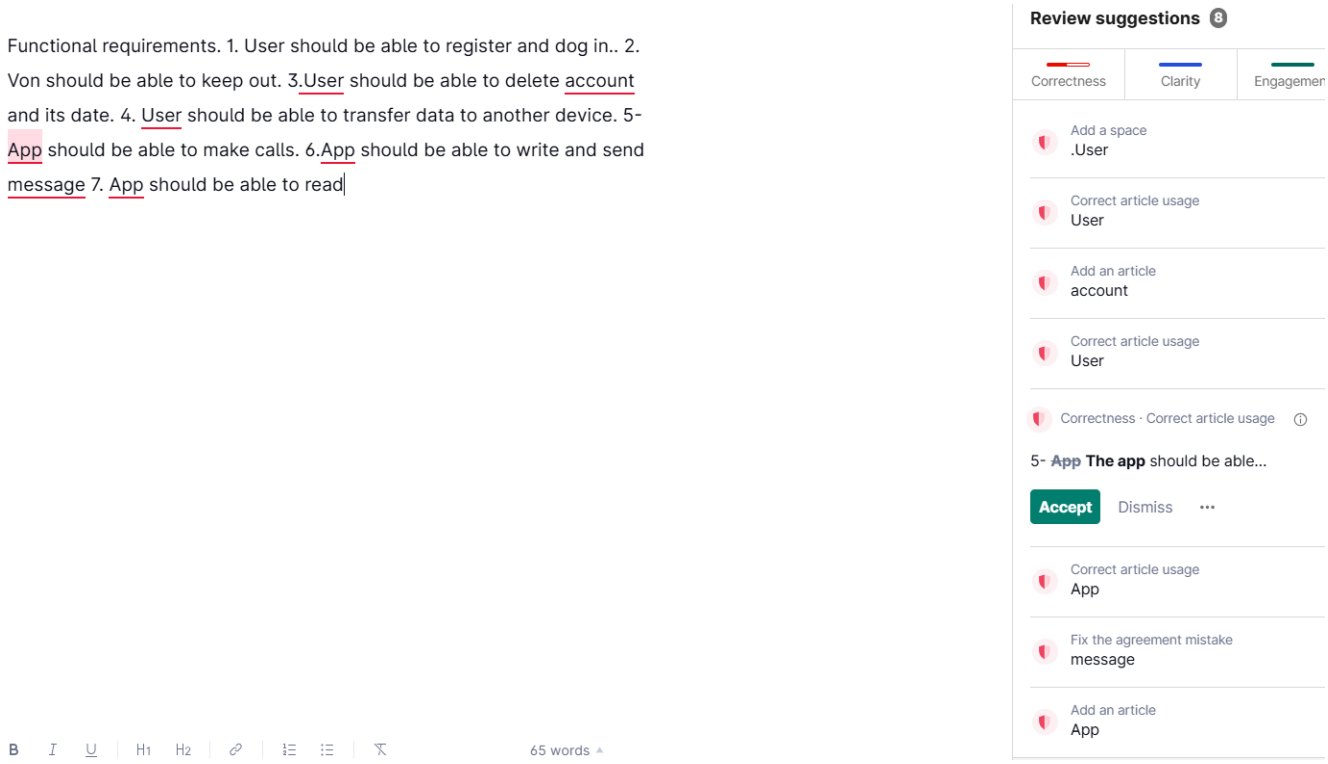


Рис. 1.6 Аналіз тексту з Grammarly

ProWritingAid – прямий конкурент Grammarly. Застосунок володіє тими ж функціями – аналіз тексту на помилки, пропозиції для виправлення та інша обробка тексту [18]. Так само є інтеграція з різними популярними застосунками, які користувачі зазвичай використовують в повсякденні. Як видно на рисунку 1.7, застосунок надав менше пропозицій до тексту ніж Grammarly.

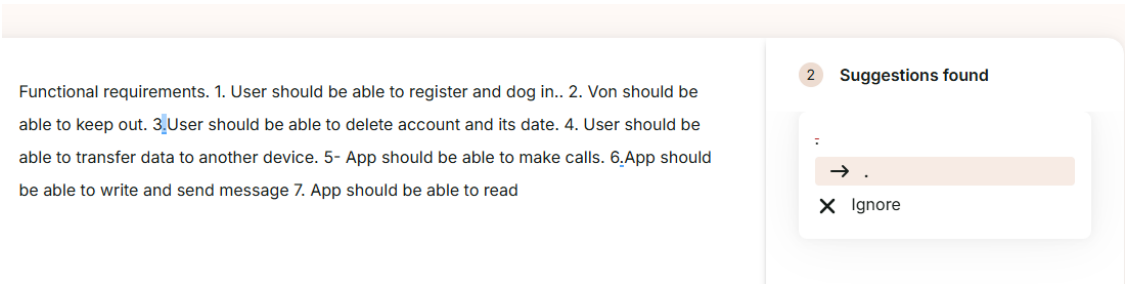


Рис. 1.7 Аналіз з ProWritingAid

SlickWrite – застосунок для перевірки тексту на граматичні, стилістичні та інші помилки [19]. Також застосунок показує детальний аналіз (див. рис. 1.8) тексту та статистику (див. рис. 1.9). У застосунку є розділ з демонстрацією можливостей.

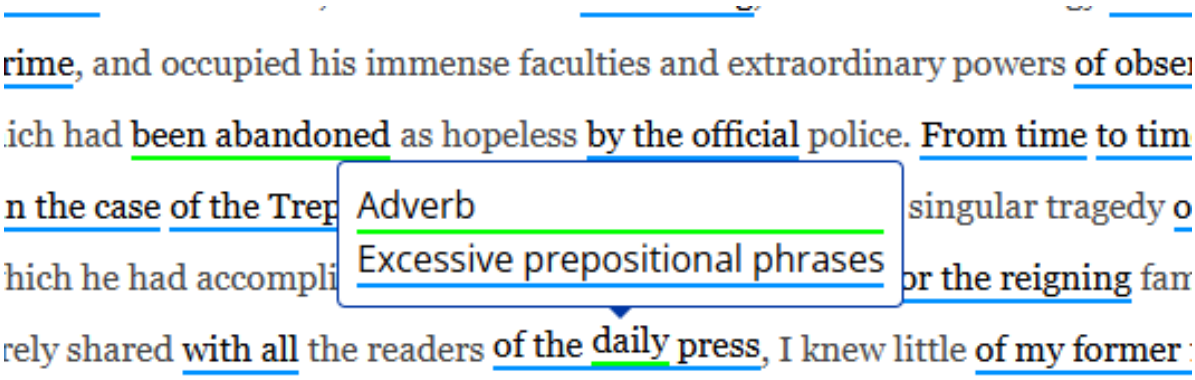


Рис. 1.8 Аналіз з SlickWrite

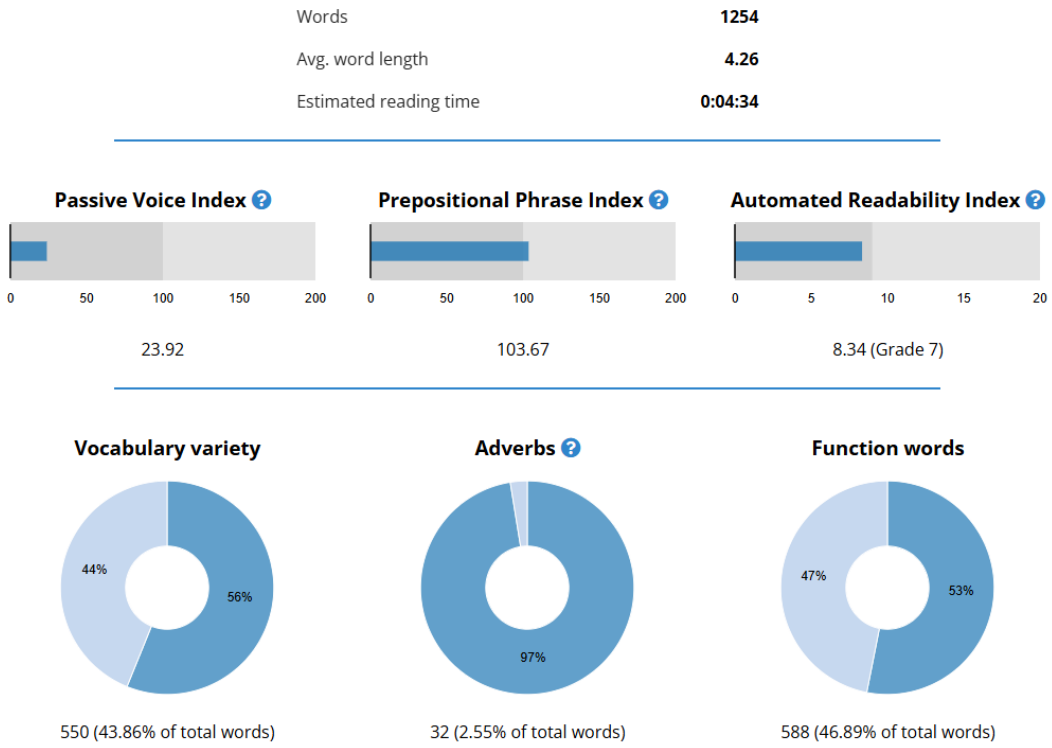


Рис 1.9 Статистика SlickWrite

Застосунок IMGProof – прямий аналог для розпізнавання та граматичної перевірки. На вході застосунок отримує зображення, розпізнає текст та виконує граматичну перевірку. Проте даний застосунок повільно виконує свою роботу, а також неправильно підсвічує помилки на зображенні.

Робота застосунку продемонстрована на рисунку 1.10. Цей застосунок є лише у web-версії, і має обмежені можливості без оплати.

Не варто відкидати підходи на основі штучного інтелекту. Серед таких підходів Gemini та Copilot (див. рис. 1.11-1.12). Вони чудово виконують розпізнавання тексту та його обробку, проте можуть потребувати значного часу на виконання таких операцій. Перевагою Gemini, Copilot та інших застосунків із ШІ є можливість «живого» спілкування з увімкненням камери, що дозволяє відправляти запити у режимі реального часу.

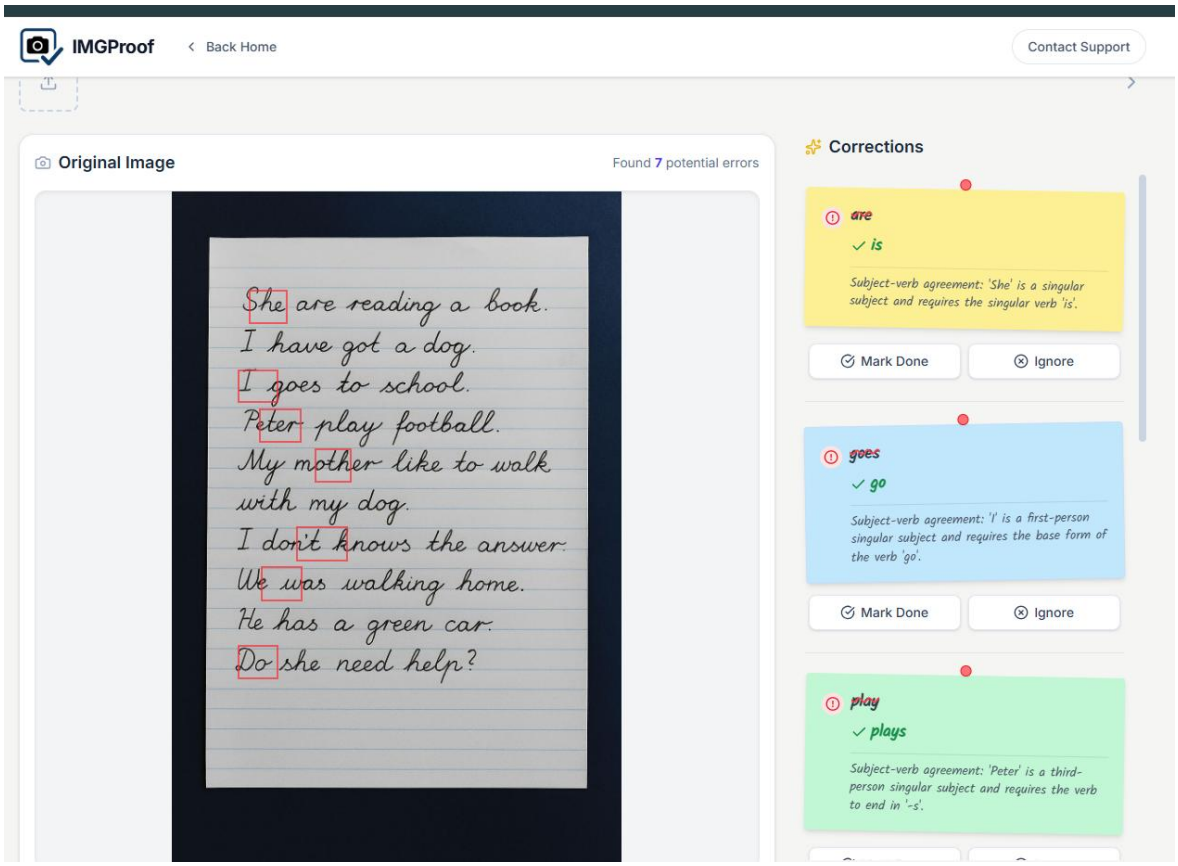


Рис. 1.10 IMGProof

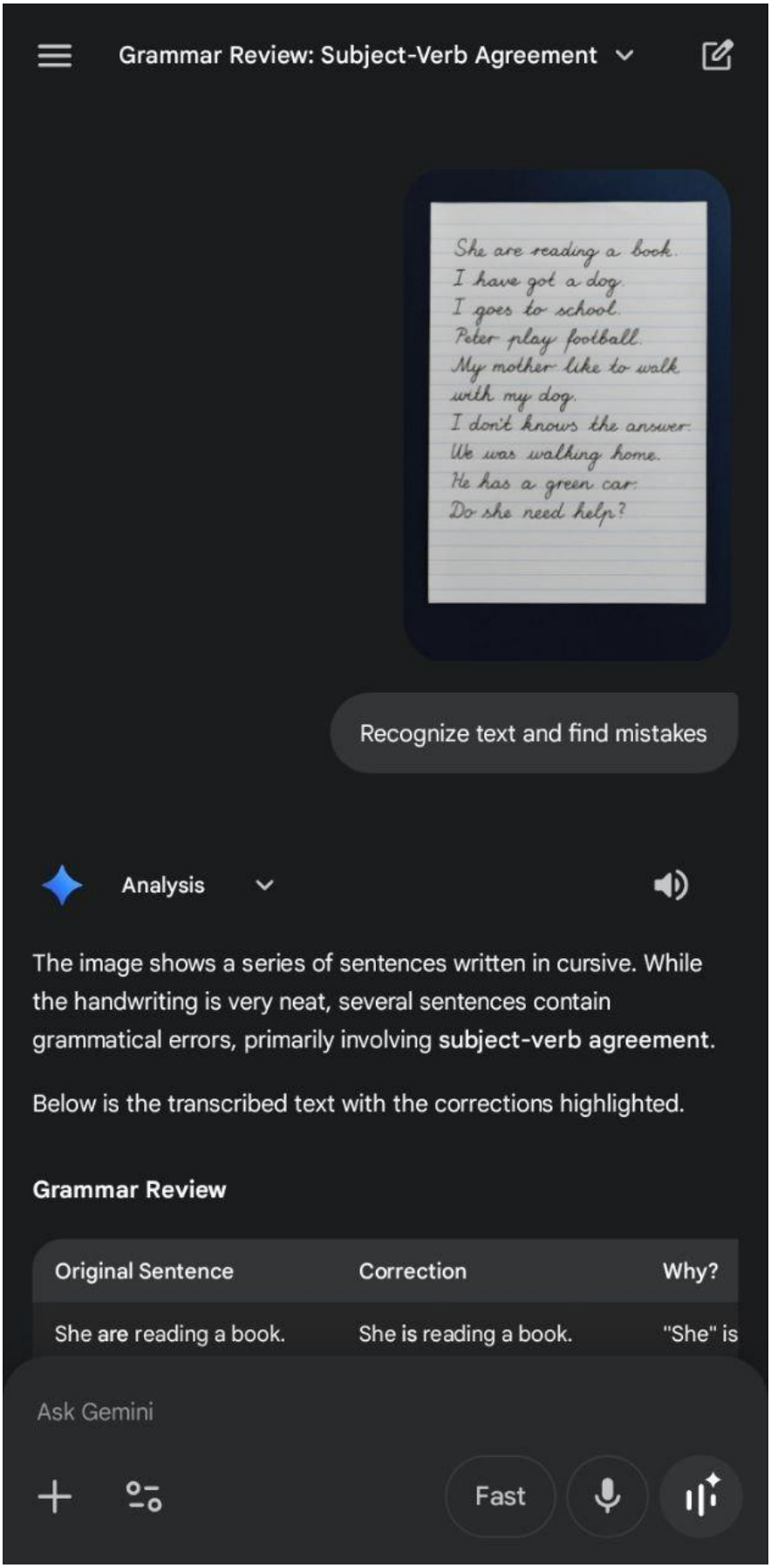


Рис. 1.11 Gemini

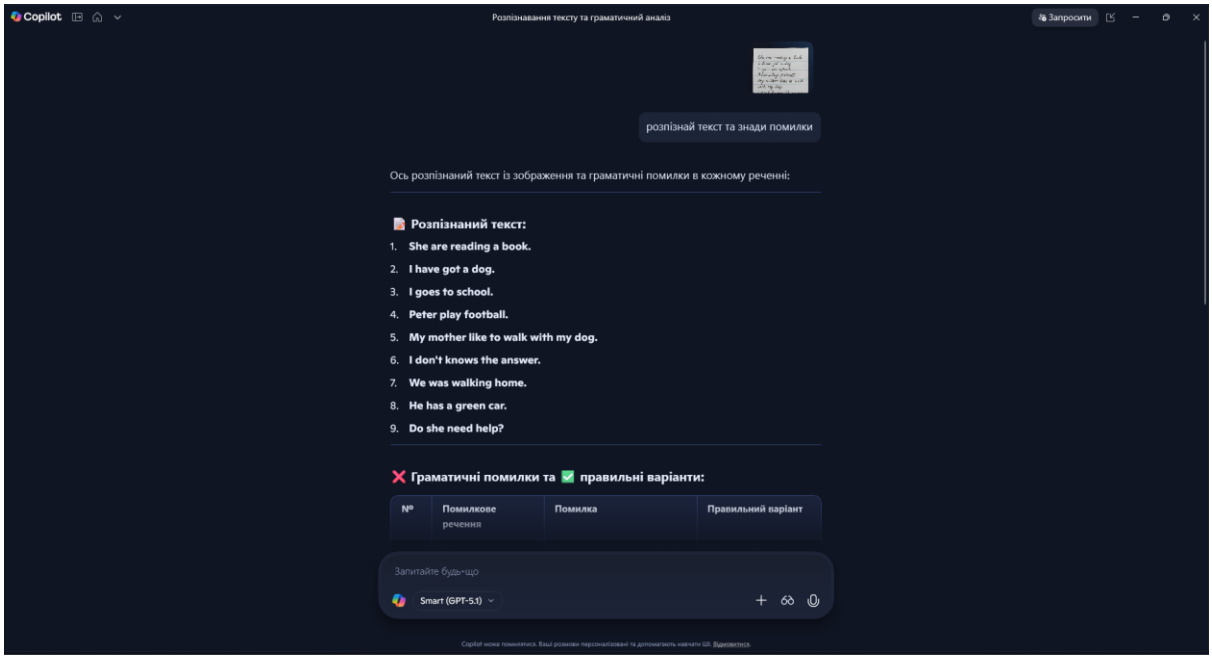


Рис. 1.12 Copilot

1.5 Висновки

Попри значний розвиток технологій та прорив у комп’ютерному баченні, все ще є необхідність у покращенні наявних методів розпізнавання та обробки рукописних текстів. Це наглядно видно при тестуванні наявних систем на рукописних текстах різної якості та різних мов.

Методи на основі штучного інтелекту мають залежність від натренованості моделі та займають більше часу на отримання результату, проте, в той же час показують кращі результати чим методи без використання штучного інтелекту.

2 РОЗРОБКА МЕТОДИКИ РОЗПІЗНАВАННЯ РУКОПИСНИХ ТЕКСТІВ ТА ГРАМАТИЧНОЇ ПЕРЕВІРКИ ТЕКСТУ ІЗ ЗАСТОСУВАННЯМ ПОПЕРЕДНЬОЇ ОБРОБКИ

2.1 Аналіз літературних джерел

Із переходом на електронні документи з’явилася необхідність оцифрування друкованих та написаних від руки документів. Оцифрування необхідне для комп’ютерної обробки, такої як пошук, сортування та фільтрування, узагальнення та інше [20].

Оцифрування документів має 3 варіанти реалізації. Найпростіший – ручне введення тексту через клавіатуру. Проте, зважаючи на обсяги інформації, це досить довгий і неефективний варіант. Згідно дослідження IMPACT Project таке оцифрування однієї сторінки коштує 1 EUR [21].

У дослідженні «Wich OCR toolset is good and why? A comparative study» [22] OCR інструменти розділили на 3 типи:

1) Proprietary OCR Toolset: зазвичай платні інструменти з повною підтримкою розробниками (ABBYY FineReader, Transym OCR, Readiris, Adobe Acrobat, OmniPage, Microsoft Office Document Imaging);

2) Open source: проекти з відкритим кодом, підтримувані ентузіастами, та іншими бажаючими (Tesseract, Ocrad, GOCR, OCRopus, CuneiForm, Calamari);

3) Online services: сервіси, які не потрібно завантажувати для використання; теж можуть бути з відкритим або закритим кодом (ABBYY Cloud OCR, Google Docs, Free-Online-OCR, Online OCR).

У підсумку даного дослідження, яке включало експериментування з різними інструментами, порівняно ефективність різних інструментів. Загалом платні OCR інструменти показали кращу ефективність ніж інструменти з відкритим кодом.

Якщо брати лише категорію open source, то найкращий результат показали Tesseract та OCRopus, хоча останній значно довше конвертує зображення.

У статті «Assessing the Impact of OCR Quality on Downstream NLP Tasks» [23] досить добре описано методи, які використовуються для дослідження OCR впливу на якість виконання NLP (Natural language processing – з англ. Обробка природньої мови) задач. Тестування NLP задач відбувалося у двох версіях. Перша – на вибірці даних, сформованій різними OCR інструментами. Друга – на тих самих даних, але з виправленнями інших людей. Так автори визначили вплив неточностей та помилок OCR на обробку тексту.

Дані для роботи взято у відкритому доступі з різних джерел та інших досліджень, в тому числі в напрямку OCR. Далі визначено якість OCR. Для цього існує декілька способів. Автори використовували словник та розраховували схожість *Левенштейна*.

Після цього проведено самі тести з NLP задачами:

– Сегментація: процес визначення меж речення між словами у різних реченнях; помилки OCR мають величезний вплив на сегментацію речення (що не дивно, адже навіть зміна одного символу може призвести на поділ речення на два), навіть для тексту, де майже не має помилок, ефективність сегментації досить низька.

– Визначення названої сутності: процес визначення згадок про названу сутність у тексті; вплив OCR менший ніж при сегментації.

– Парсинг залежностей: процес визначення граматичної структури тексту; ефективність парсингу залежить від типу залежності, і чим довша залежність, тим гірший результат парсингу.

Проблема розпізнавання тексту, написаного від руки, досить велика. Кожен почерк має свої властивості, стиль. Також не допомагає і те, що при написанні від руки, люди часто роблять помилки, помарки та неточності. Буває так, що навіть людині важко прочитати такий текст.

Автори статті «Handwritter Optical Character Recognition (OCR): A Comprehensive Systematic Literature Review (SLR)» [24] оглянули уже існуючі дослідження на тему розпізнавання тексту написаного від руки. Метою даного огляду було:

– підсумувати наукові дослідження про розпізнавання тексту, написаного від руки, різними мовами;

– виділити недоліки досліджень, щоб відкинути їх у результаті подальших досліджень;

– визначити нові напрямки досліджень в області OCR технологій.

Для виконання даного огляду, автори розробили протокол огляду. Цей протокол визначає поле огляду, стратегію пошуку, видобування даних, питання для дослідження, критерії для визначення якості досліджень.

Для того, щоб до огляду допускалися лише ті статті та дослідження, які стосуються теми огляду, складено критерії включення та виключення. До огляду було включено статті, опубліковані в проміжку від січня 2000го року по грудень 2019го року. Допущено статті написані на англійській, урду, арабській, перській, індійській та китайській мовах.

Проте кількість статей на початковому етапі була величезна. Тому потрібно було вручну відсіяти багато статей, які:

– не стосувалися теми дослідження;

– були дублікатами;

– не доступний повний текст;

– дослідження не стосувалося ніякого зі складених запитань.

Такий ретельний підхід до вибору джерел допоміг звузити коло до 176 статей.

Основні питання дослідження:

– Які типи методів для видобування та класифікації використовуються в OCR для рукописних текстів?

– Які є бази даних або вибірки даних для досліджень?

– Які нові домени для досліджень в області OCR?

В системах OCR для рукописних текстів, алгоритм навчений на вибірці даних. При навчанні, алгоритм вивчає як правильно категоризувати або класифікувати букви та цифри.

Основні методи класифікації:

– Artificial neural networks (ANN). Нейрони працюють разом щоб змоделювати вхідні дані та відобразити їх у визначеному класі. Для зменшення помилки роботи «вагу» зв’язків між нейронами можна регулювати.

– Kernel методи. Зазвичай базуються на векторному аналізі зображення.

– Статистичні методи. Бувають двох типів – параметричні (проста структура) та непараметричні (структура ускладнюється з кількістю вхідних даних).

– Техніка співпадіння шаблону. Зображення, або частина зображення співпадає зі шаблоном. Оскільки у кожному почерку символи можуть мати різні деформації, можна використовувати деформативні шаблони (див. рис. 2.1).

– Розпізнавання патерну структури. Популярний, до винайдення Kernel методів. Базуються на класифікації об’єктів на основі зв’язків між ними.

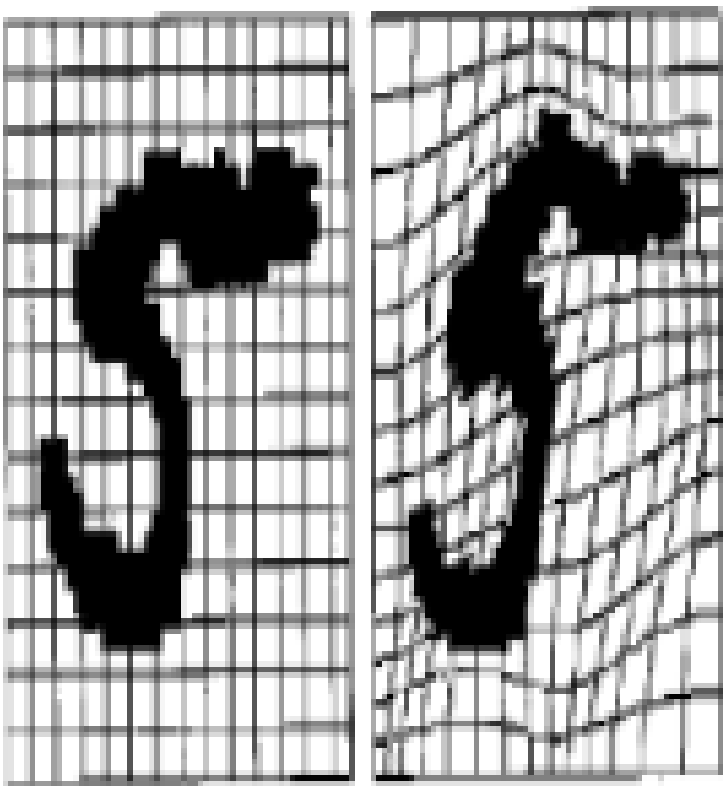


Рис. 2.1 Деформативний шаблон цифри «5»

У статті описано результати для кожної з 6ти визначених мов. За різними дослідженнями, після впровадження нейронних мереж у OCR, середня точність зросла з 86% до 90%, а при використанні більшої вибірки даних, помилка складала лише 4%. У випадку з фарсі (перська мова), різні дослідники спромоглися досягти неймовірних 99% точності розпізнавання.

У дослідженнях з мовою урду, точністю варіюється від 90 до 99 відсотків, окрім одного дослідження, де методом на основі нейронних мереж вдалося досягти лише 74% точності. Розпізнавання китайської та індійської мов теж варіюється в діапазоні 90-99.

Дослідники дійшли висновку, що розвиток нейронних мереж дозволив значно підвищити точність розпізнавання тексту. Більшість досліджень пропонують рішення для покращення точності для конкретної мови або набору мов.

У статті «Analysis of Recent Deep Learning Techniques for Arabic Handwritten-Text OCR and Post-OCR Correction» [25] перераховуються різні складності для розпізнавання рукописного тексту:

– Складність «сцени»: відділення тексту від решти може бути важко у зв’язку із забрудненням (наприклад, різні лінії на фоні, які схожі на символи, але не є ними).

– Нерівномірне освітлення: наявність тіней чи нерівномірного освітлення погіршує точність OCR.

– Кут нахилу: якщо робити фото під неправильним кутом, то рядки будуть йти з неправильною орієнтацією.

– Блюр та деградація: неякісне та нечітке зображення погіршує точність OCR.

– Різний розмір символів: змушує виконувати різні процедури для пошуку тексту.

– Стилі: кожен стиль різний, і навіть в межах одного почерку, той хто пише може по різному виконувати написання одного і того ж символу.

– Багатомовність: часто текст включає в себе слова або цілі речення іншими мовами.

Також описуються складнощі саме для арабської мови, з чого стає зрозуміло, що OCR потрібно використовувати різні методи для різних мов. Наприклад арабська пишеться з права на ліво, в той час як більшість мов навпаки. З іншого боку в арабській немає маленьких чи великих букв, тому не потрібно їх відрізняти.

У своїй роботі «Розпізнавання кириличного тексту» [26], Костянтин Лєпєшов провів тренування, а потім тестування моделі на базі даних із 82000 зображень із текстом з математичної олімпіади. Похибка у результаті тестування була 4.8%.

Після отримання цифрової версії тексту, необхідно виконати його обробку. А саме пошук синтаксичних, граматичних та інших помилок. Найлегшим прикладом є автовиправлення слів під час написання з клавіатури. Користувач пише слово, робить помилку, наприклад пише не ту букву, і система автоматично заміняє на це слово на правильне. Таке виправлення може не просто шукати найближче до написаного користувачем реальне слово, а з допомогою штучного інтелекту може перевірити щоб заміна підходила і за значенням. Про автовиправлення тексту з використанням штучного інтелекту написано у статті «Real-time Automatic Text Correction Using Artificial Intelligence» [27]. Звичайно обробка тексту після OCR трішки інший випадок, проте технології, які застосовуються в цій обробці можуть використовуватися і в даному випадку.

Більш складний випадок це перевірка всього тексту, а не окремих слів. У статті «Grammatical Error Correction: A Survey of the State of the Art» [28] описано проблеми, які виникають при обробці природньої мови у контексті пошуку та виправлення граматичних помилок:

– декілька варіантів правильного написання;

– різний контекст;

– мінімальні виправлення чи повна перебудова речення;

Для тренування та тестування моделей використовуються уже наявні вибірки даних – datasets. Таким чином розробникам не потрібно витрачати час на створення власних вибірок даних.

Серед таких вибірок англійською:

– FCE: 1244 текси з близько 531 тис. слів.

– NUCLE/CoNLL: 1397 текстів з близько 1.16 млн. слів.

– Lang-8: 100 тис. текстів на 11.8 млн. слів.

– JFLEG: 1501 речення на 28.1 тис. слів.

– W&I+LOCNESS: 3600 есе на 755 тис. слів.

– CLC: 130 тис. текстів на 29.1 млн. слів.

– EFCamDat: 1.18 млн. тектсів на 83.5 млн. слів.

– WikEd: 626 млн. слів.

– AESW: 35.5 млн. слів.

– GMEG: 122.4 тис. слів.

– та багато інших.

Також використовувалися вибірки арабською, китайською, чеською, німецькою, японською, українською.

Виправлення граматичних помилок включає класифікатори, машинний переклад, підхід редагування та мовні моделі.

Класифікатори – один із найпопулярніших способів виправлення граматичних помилок. Такий підхід часто застосовують до артиклів, препозицій, іменників чисел, форм дієслів. Це тому, що виправлення помилок пов’язаних з перерахованими частинами мови легко передбачити. Їх класифікують, а при отриманні вхідних даних, після чого класифікатори порівнюють передбачуване слово з реальним і заміняють їх за потреби.

Статистичний машинний переклад (SMT) – система, яка спочатку розроблена для перекладу з однієї мови на іншу. Теоретично SMT може правильно виправити усі помилки одночасно без спеціалізованого знання. Простими словами, цей метод перекладає з однієї мови на ту саму, тільки вхідним текстом є текст з помилками, а на виході отримуємо правильний текст.

Нейронний машинний переклад – із появою машинного навчання та нейронних мереж, їх активно почали впроваджувати у багато різних технологій різних сфер, у тому числі NLP. Таку систему потрібно навчати для успішного застосування.

Підхід редагування – замість генерування правильного тексту на основі введеного, виконує серію редагувань введеного тексту. На кожен токен введеного тексту система передбачає операцію редагування. Головною перевагою такого методу є швидкість.

Мовні моделі – на відміну від нейронного машинного перекладу тренувати моделі не потрібно. Головною перевага методу – він вимагає лише анотовані монолінгвістичні дані, які легко поширити на інші мови.

У результаті, різні моделі показували різні результати. Це пов’язано з тим, що вони тренувалися на різних текстах і на різній їх кількості. Варто зазначити, що використання нейронного перекладу досить ресурсозатратний спосіб. Використання декількох навчених архітектур моделей покращує ефективність.

У своїй роботі «Розпізнавання рукописного тексту: сучасні підходити та виклики», Наталія Кунанець та Юліан Баранецький, детально розглянули проблеми та підходи до вирішення для зчитування рукописів [3].

Системи розпізнавання стикаються одразу з декількома проблемами:

– варіативність тексту;

– залежність від стилю письма;

– індивідуальні особливості почерку автора;

– багатомовність;

– якість фізичного носія тексту (текст може бути пошкоджений, що поширено серед історичних та архівних документів);

– незвичні шрифти;

– якість зображення.

Для розпізнавання рукописного тексту використовуються такі підходи:

– Приховані моделі Маркова;

– Статистичні моделі;

– Згорткові нейронні мережі;

– Рекурентні нейронні мережі;

– Довготривала короткочасна пам’ять;

– Трансформери.

Приховані моделі Маркова та статистичні моделі ґрунтуються на припущеннях, що процес генерації тексту модна подати як марківський процес зі скінченною кількістю станів. В основі моделей лежать ймовірнісні розрахунки.

Згорткові нейронні мережі – Це нейронні мережі, побудовані із шарів згорток, пулінгу та повнозв’язних шарів. Їхня перевага у тому, що вони можуть автономно виділяти ознаки зображень різних рівнів.

Рекурентні нейронні мережі – це тип нейронних мереж, що працює із послідовними даними. Суть підходу полягає у запам’ятовуванні інформації про попередні елементи та використання цієї інформації для розпізнавання наступних елементів.

Довготривала короткочасна пам’ять – тип рекурентних мереж, який має спеціальні комірки пам’яті, що дозволяє зберігати контекст.

Для найкращого результату підходи часто комбінують. Використовуючи основні переваги різних підходів можна досягти значних результатів.

2.2 Математична модель МРОТР

Математична модель методики представлена композицією функцій роботи OCR, попередньої обробки тексту, перевірки тексту на помилки та описана формулою 2.1:

$$\mathrm{F} = G \cdot N \cdot R, \tag{2.1}$$

де R – функція розпізнавання рукописного тексту (формула 2.2);

N – функція попередньої обробки тексту (формула 2.3);

G – функція пошуку помилок (формула 2.4).

$$R = X \to T, \tag{2.2}$$

де X – зображення із текстом;

T – розпізнана послідовність символів.

$$N = T \to T', \tag{2.3}$$

де T – розпізнана послідовність символів;

T' – текст після попередньої обробки.

$$G = T' \to E, \tag{2.4}$$

де T' – текст після попередньої обробки;

E – множина знайдених помилок (формула 2.5).

$$E = \{(p_k, e_k)\}_{k=1}^{K}, \tag{2.5}$$

де p_k – позиція помилки;

e_k – опис помилки;

K – множина значень помилок.

Запропонована методика обробки рукописного тексту ґрунтується на послідовному застосуванні трьох функціональних операторів: розпізнавання тексту, попередньої обробки тексту та пошуку помилок. Сукупна дія цих операторів описується композицією функцій (формула 2.1), що відображає логіку проходження даних через усі етапи обробки.

2.3 Сервіс розпізнавання тексту OCRSpace

OCRSpace – це сервіс розпізнавання тексту від a9t9 software, створений для практичного застосування останніх досягнень комп’ютерного зору. Для доступу до сервісу є API з наявною документацією [29].

Принцип роботи з даним сервісом простий – система (користувач) передає зображення або PDF-файл у запиті, сервіс розпізнає текст та повертає результат системі (див. рис. 2.2).

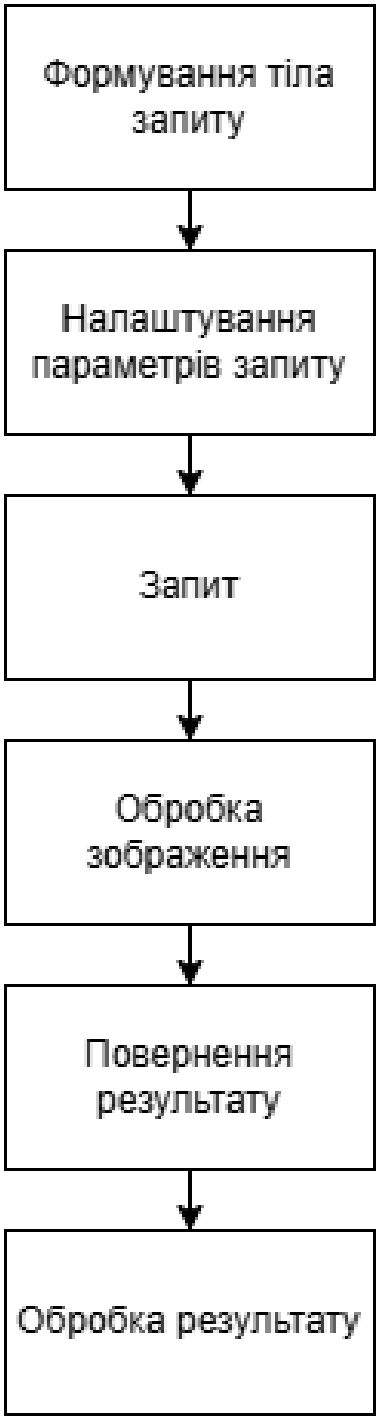


Рис. 2.2 Загальна схема роботи OCR

Запит складається з тіла запиту, типу запиту та посилання. Запит до цього сервісу може бути GET або POST, проте саме POST запит можна детально налаштувати під власні потреби, і саме POST запитом можна передавати файли.

Тіло запиту може містити файли, параметри, JSON, XML, текст. Саме через тіло передаються файли та конфігурація, яка необхідна для налаштування сервісу під власні потреби.

Після відправки запиту виконується його обробка сервісом. Сервіс отримує зображення (або PDF-файл), бере задану конфігурацію із запиту та виконує з цією конфігурацією обробку зображення. Далі повертається результат роботи сервісу із тілом JSON, який система обробляє.

Перевагою OCRSpace є можливість детального налаштування. Перш за все сервіс має 2 двигуни, які мають різну логіку розпізнавання тексту.

Двигун 1 швидший за другий, виконує базове розпізнавання. Цей двигун використовує менше ресурсів, тому може опрацювати більші зображення.

Двигун 2 має більше можливостей:

- автоматичне визначення мови;

- добре розпізнає спеціальні символи (наприклад, §$@$€/()[]{});

- добре розпізнає текст, написаний на незвичному фоні (наприклад на зображенні).

Для задання конфігурації застосовуються параметри. Повний перелік параметрів описано у таблиці 2.1.

Таблиця 2.1

Параметри POST запиту

Параметр	**Опис**
apikey	Ключ доступу до API
url/file/base64Image	• url – посилання на файл зображення чи PDF-файл • file – файл зображення або PDF • base64Image – закодоване зображення у літерал

Продовження таблиці 2.1

Параметри POST запиту

Параметр	**Опис**
detectOrientation	Визначає, чи потрібно визначити поворот зображення та перевернути, щоб текст був правильно відображений на зображенні.
isCreateSearchablePdf	Вказує чи потрібно згенерувати та повернути розпізнаний текст у файлі PDF.
isSearchablePdfHideTextLayer	Визначає, чи буде видно текстовий шар у PDF-файлі.
OCREngine	Визначає двигун
language	Вибір мови, яка на зображенні, або значення «auto» для автоматичного визначення
isOverlayRequired	Визначає, чи потрібно повертати координати кожного слова, наприклад для виділення цих слів на зображенні
filetype	Визначає тип файлу, проте не обов’язково, так як тип визначається автоматично

Перевагою даного сервісу є підтримка великої кількості мов:

– Арабська,

– Болгарська,

– Китайська (спрощена),

– Китайська (традиційна),

– Хорватська,

– Чеська,

– Данська,

– Нідерландська,

– Англійська,

– Фінська,

– Французька,

– Німецька,

– Грецька,

– Угорська,

– Корейська,

– Італійська,

– Японська,

– Польська,

– Португальська,

– Російська,

– Словенська,

– Іспанська,

– Шведська,

– Таїландська,

– Турецька,

– Українська,

– В’єтнамська.

2.4 Сервіс граматичної перевірки LanguageTool

LanguageTool – сервіс граматичної перевірки із відкритим вихідним кодом. Для роботи цього сервісу можна запустити власний сервер, або використовувати API [30].

Алгоритм роботи досить простий (див. рис. 2.3) – система виконує запит до сервісу, із текстом та іншими параметрами у тілі запиту. Сервіс розділяє текст на речення. Кожне речення розділяється на слова. Кожному додаються маркери, які вказують на частину мови, рід, множину та інші характеристики. Далі проаналізований текст перевіряється на співпадіння за створеними правилами із

файлів у сервісі. Після цього отримується результат, який містить список помилок, та їх опис.

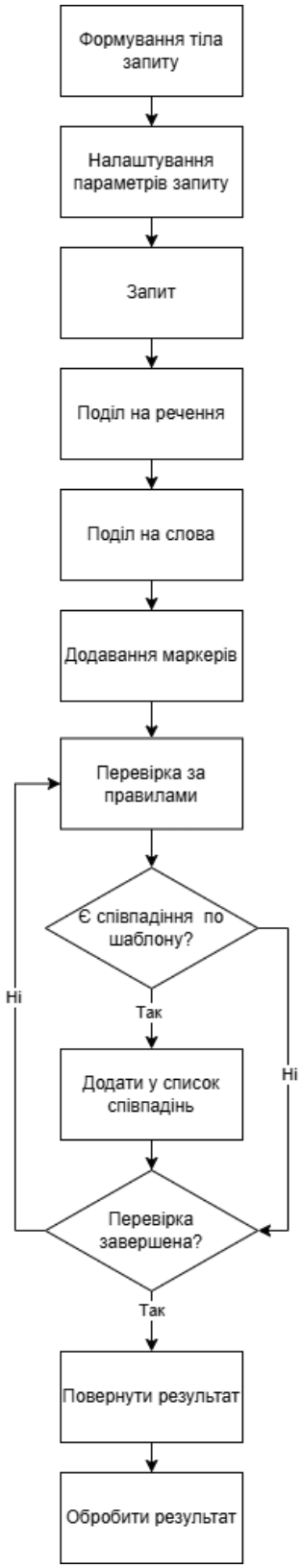


Рис. 2.3 Схема роботи LanguageTool

Великою перевагою є можливість додавання власних правил та модифікації алгоритмів сервісу, а також можливість використання n-gram даних для власного серверу.

Список підтримуваних мов та кількості правил описаний у таблиці 2.2. Правила описані двома мовами, прості у XML файлі, складніші – на Java.

Таблиця 2.2

Підтримувані мови LanguageTool

Мова	**XML правила**	**Java правила**
Арабська	450	16
Астурійська	71	0
Білоруська	66	2
Бретонська	675	3
Каталанська	8251	50
Китайська	1863	1
Кримсько-татарська	93	0
Данська	78	0
Нідерландська	3500	17
Англійська	6074	54
Есперанто	422	1
Французька	6984	22
Галісійська	308	7
Німецька	5224	79
Грецька	55	5
Ірландська	1663	16
Італійська	141	3
Японська	735	0
Кхмер	33	5

Продовження таблиці 2.2

Підтримувані мови LanguageTool

Мова	**XML правила**	**Java правила**
Перська	283	7
Польська	1747	8
Португальська	2919	48
Румунська	457	3
Російська	892	20
Словацька	170	1
Словенська	86	0
Іспанська	1644	23
Шведська	32	2
Тагальська	44	0
Таміл	210	0
Українська	1186	31

Сервіс має декілька запитів для граматичної перевірки, отримання списку підтримуваних мов та 3 запити для керування особистим словником.

Для граматичної перевірки використовується POST запит. Параметри запиту описані у таблиці 2.3.

Таблиця 2.3

Параметри POST запиту

Параметр	**Опис**
text/data	Текст, який потрібно перевірити, або JSON документ із текстом та розміткою

Продовження таблиці 2.3

Параметри POST запиту

Параметр	**Опис**
language	Мова, якою написаний текст, можна встановити автовизначення
username	Облікові дані користувача
apiKey	Ключ доступу API
motherTongue	Мова користувача
preferredVariants	Список варіантів мови, яким надається перевага (наприклад, Австралійська Англійська)
enabledRules	Список правил, які потрібно увімкнути
disabledRules	Список правил, які потрібно вимкнути
enabledCategories	Список категорій, які потрібно увімкнути
disabledCategories	Список категорій, які потрібно вимкнути
enabledOnly	Вмикає лише ті категорії та правила, які перераховані у відповідних списках
level	Рівень перевірки (формальна/неформальна мова)

2.5 Попередня обробка тексту

Попередня обробка тексту – важливий етап після розпізнавання тексту, для усунення дефектів розпізнавання та підготовки до перевірки на помилки. Задачі, які виконує попередня обробка тексту [31]:

- очищення тексту від маркування;
- нормалізація (видалення зайвих пробілів, порожніх рядків);
- відновлення меж речень (визначення початку і кінця речень);
- забезпечення завершення меж речень (якщо якесь речення не закінчується крапкою, чи іншим знаком, ставиться крапка);
- виправлення OCR артефактів (виникають при розпізнаванні внаслідок схожості символів або груп символів).

Алгоритм попередньої обробки продемонстровано на рисунку 2.4. Така обробка дозволяє значно підвищити точність перевірки тексту на помилки.

Детальний алгоритм процесу об’єднання рядків зображений на рисунку 2.5.

На рисунку 2.6 зображений алгоритм видалення зайвих пробілів. На рисунку 2.7 зображений алгоритм визначення меж речень та забезпечення завершень речень.

Для виправлення артефактів OCR потрібно мати певний словник. Такі словники складалися спеціалістами протягом певного часу на основі аналізу роботи систем комп’ютерного зору, тому можна взяти уже готовий словник.

На рисунку 2.8 зображено алгоритм виправлення артефактів.

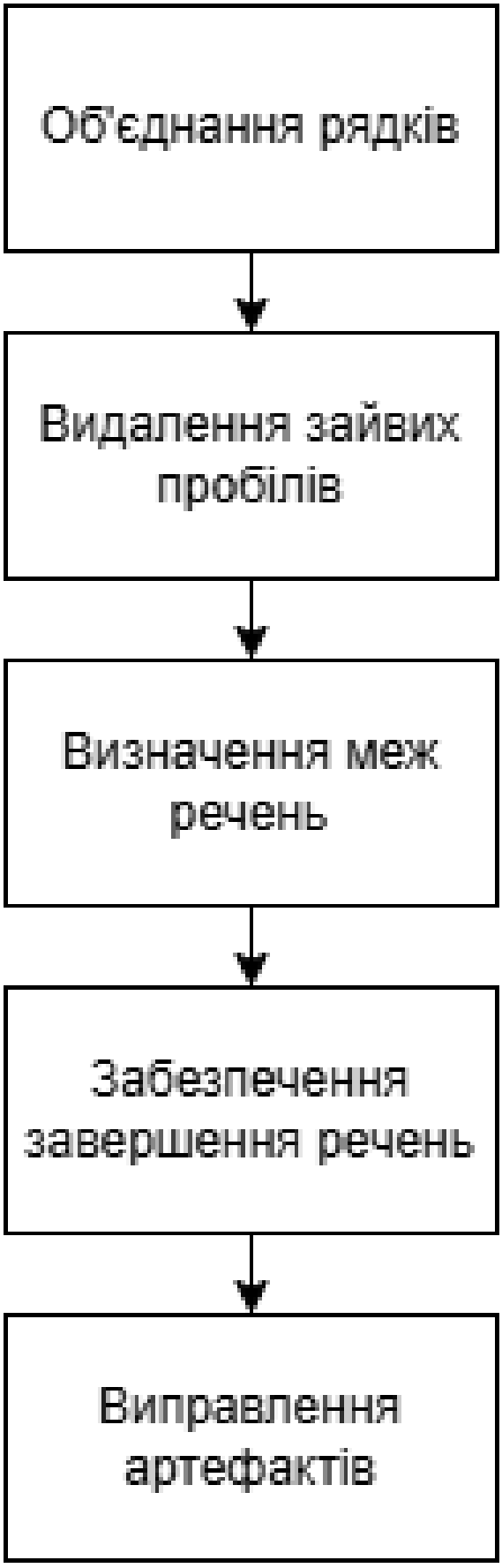


Рис. 2.4 Схема попередньої обробки тексту

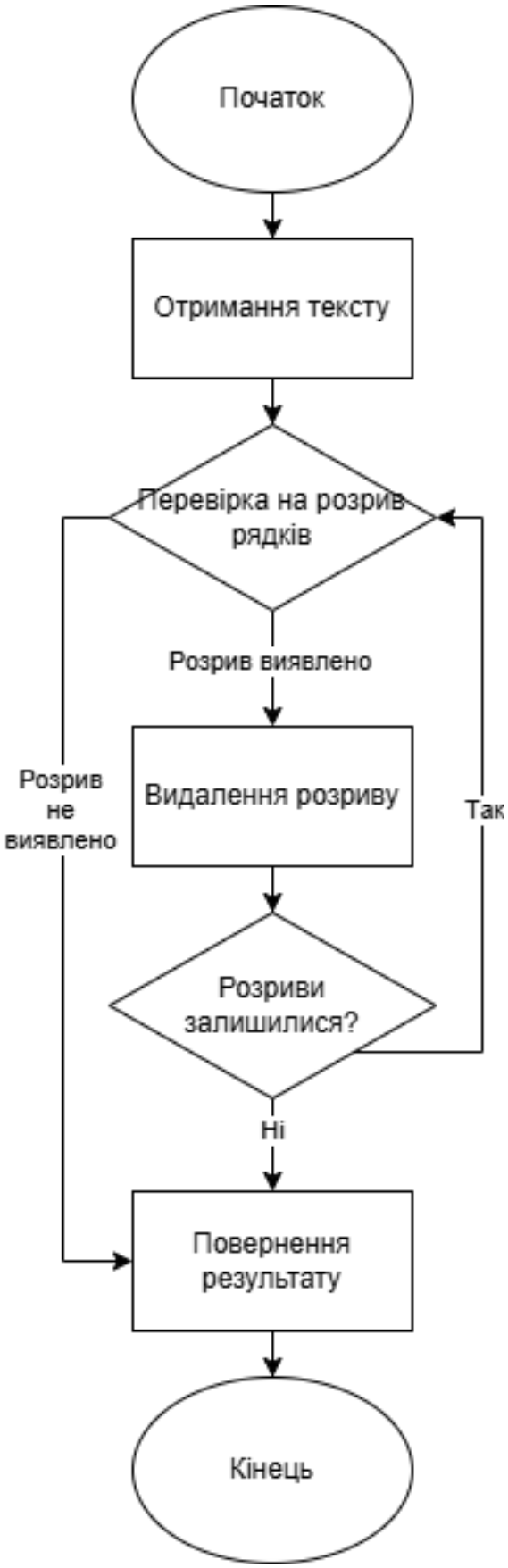


Рис. 2.5 Алгоритм процесу об’єднання рядків

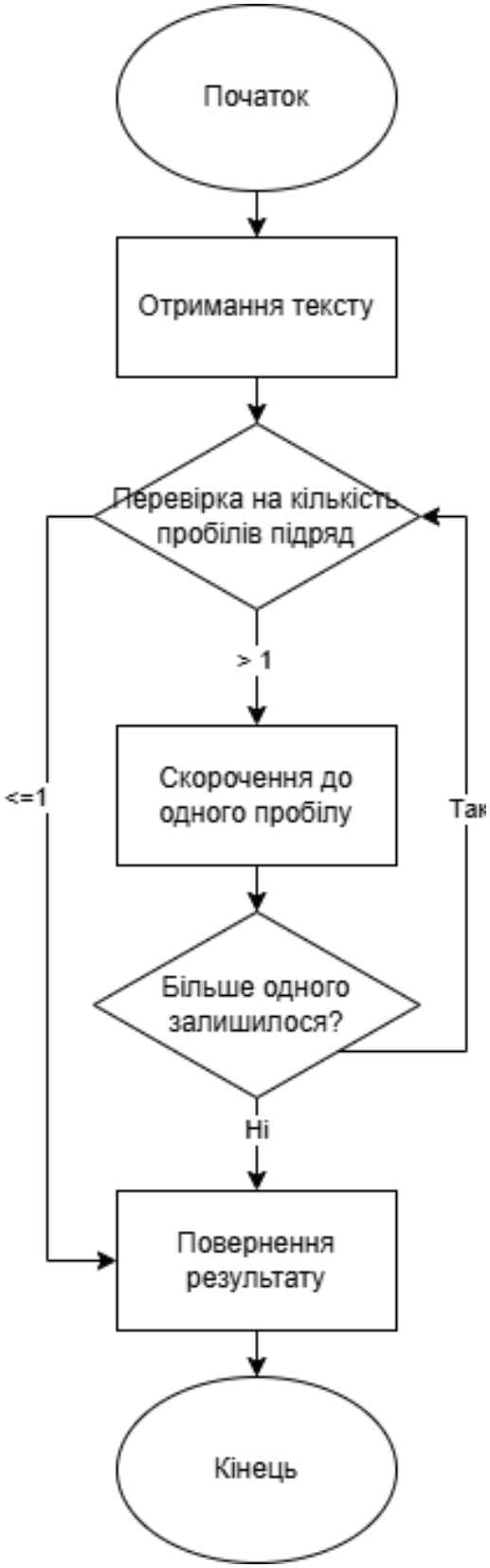


Рис. 2.6 Алгоритм видалення зайвих пробілів

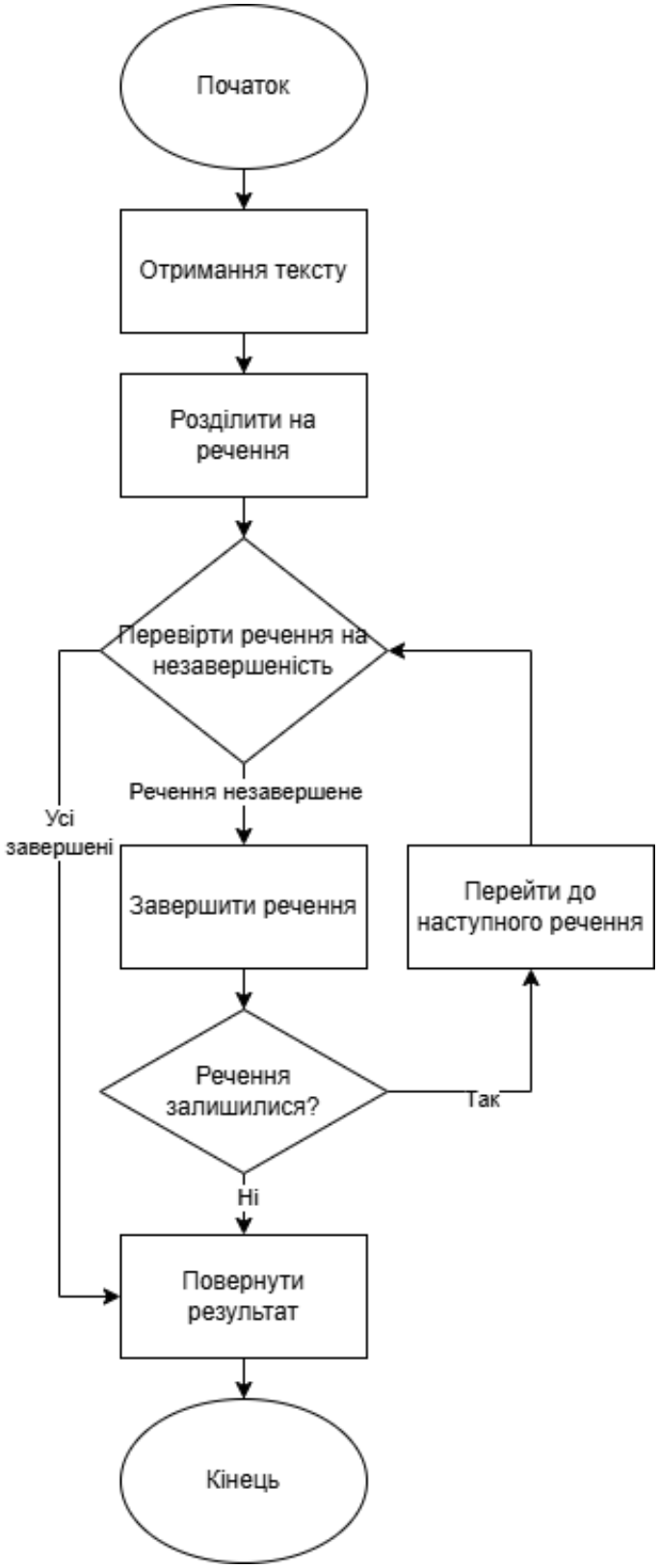


Рис. 2.7 Алгоритм визначення меж речень та завершення

Рис. 2.8 Алгоритм виправлення артефактів

2.6 Висновки

Проведено детальний аналіз науково-технічної літератури. Визначено основні підходи до розпізнавання тексту, попередньої обробки тексту, аналізу тексту на помилки. Виявлено основні переваги і недоліки різних підходів, та труднощі із якими стикаються методи під час розпізнавання тексту. Основними проблемами при розпізнаванні рукописного тексту є варіативність почерку та якість зображення.

Розроблено математичну модель, яка представлена композицією трьох процесів – розпізнавання тексту, попередня обробка тексту, граматична перевірка тексту.

Розроблено методику МРОРТ (методика розпізнавання та обробки рукописного тексту), яка полягає у послідовному застосування алгоритмів розпізнавання тексту, попередньої обробки тексту та граматичної перевірки. Для розпізнавання обрано сервіс OCRSpace, а для граматичної перевірки – LanguageTool. Попередня обробка тексту полягає у нормалізації тексту, забезпеченні завершення речень та усуненні артефактів розпізнавання.

3 РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ. ПРОВЕДЕННЯ ДОСЛІДЖЕННЯ ТА ПОРІВНЯЛЬНОГО АНАЛІЗУ

3.1 Інструменти розробки

Для розробки використано мову програмування Kotlin. Kotlin – це мова програмування на базі Java, яка розробляється та обслуговується компанією JetBrains за підтримки Google. Ця мова програмування першочергово розроблена для розробки мобільних застосунків під операційну систему Android, проте наразі має безліч можливостей, від крос-платформної розробки до розробки серверного програмного забезпечення.

Переваги Kotlin [32]:

– Сумісність з Java: код із Java можна використовувати у Kotlin та навпаки.

– Null-safety: у Kotlin є низка заходів, щоб виникало менше помилок null-pointer – змінні у Kotlin мають чітке визначення того, чи можуть вони приймати значення «null», в такому випадку помилка виникає ще на етапі компіляції, а не при роботі програмного забезпечення.

– Експресивний синтаксис: синтаксис Koltin дозволяє виразити програму більш читабельним та меншим за обсягом кодом.

– Розумні приведення типів: Kotlin автоматично обробляє приведення типів зменшуючи кількість шаблонного коду.

– Стислі класи: у Kotlin є заготовки класів під різні ситуації, такі як клас даних, інтерфейс, об’єкт та інші.

– Корутини: для асинхронного програмування у Kotlin передбачено корутини, перевагою яких є простота та краща продуктивність ніж у потокового програмування.

– Функції розширення: Kotlin дозволяє додавання нових функцій до класів без зміни їхнього коду.

– Перевантаження операторів: Kotlin дозволяє задавати власну реалізацію операторам.

– Підтримка DSL (Domain-Specific Languages): наприклад можна писати HTML код у Kotlin.

У якості інтегровано середовища розробки використано AndroidStudio. Це середовище розроблене на базі IntlliJ IDEA компанією JetBrains за підтримки Google.

Середовище має низку важливих інструментів [33]:

– редактор коду;

– компілятор;

– Android-емулятор (див. рис. 3.1);

– передпоказ інтерфейсу користувача (див. рис. 3.2);

– логування пристрою (див. рис. 3.3);

– аналіз коду (див. рис. 3.4);

– вбудований штучний інтелект;

– інтегрована система контролю версій Git.

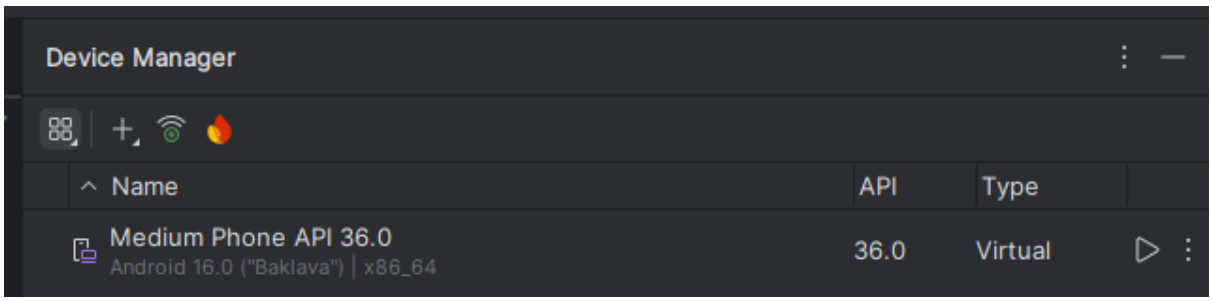


Рис. 3.1 Вікно керування пристроями-емуляторами.

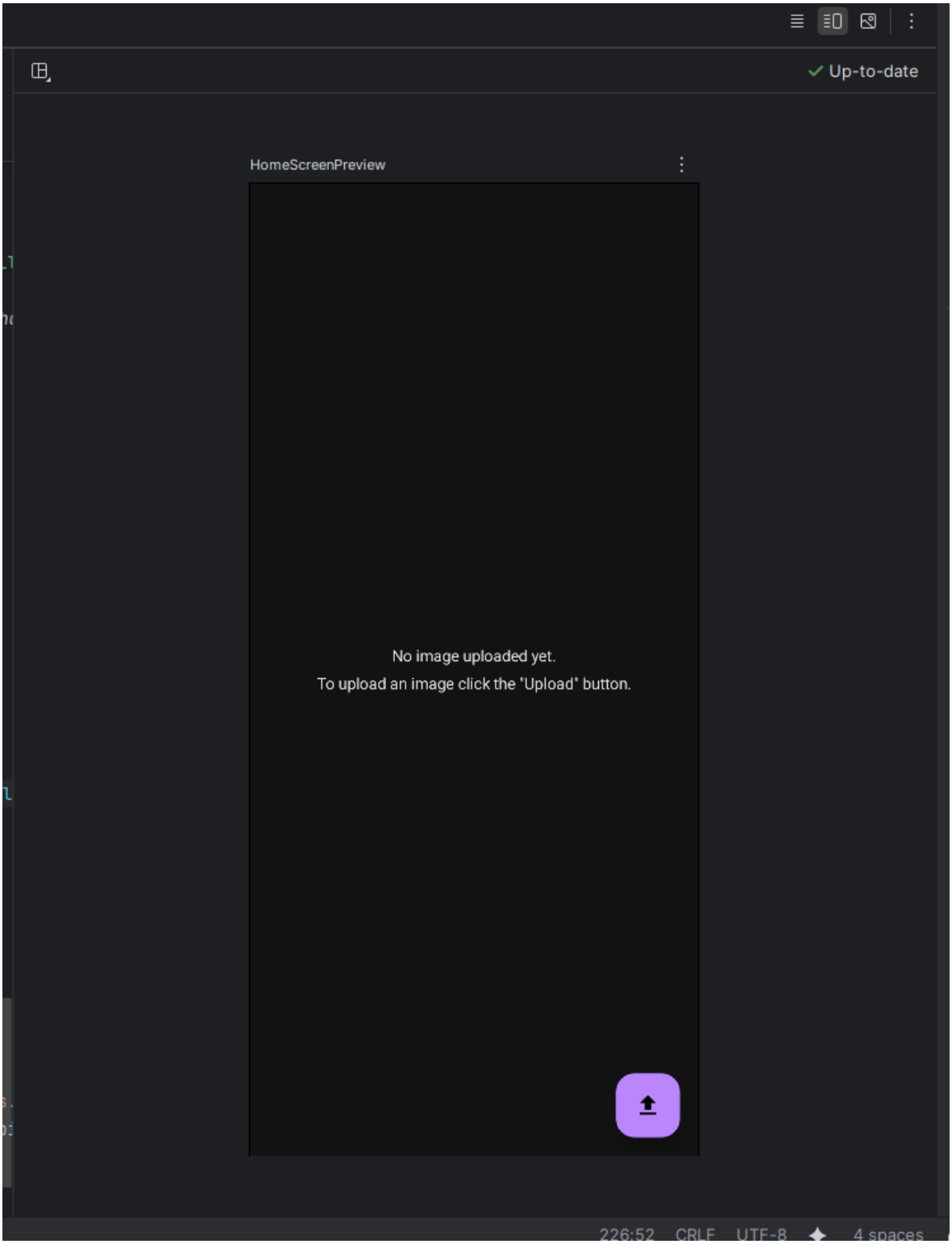


Рис. 3.2 Вікно перед показу ІК.

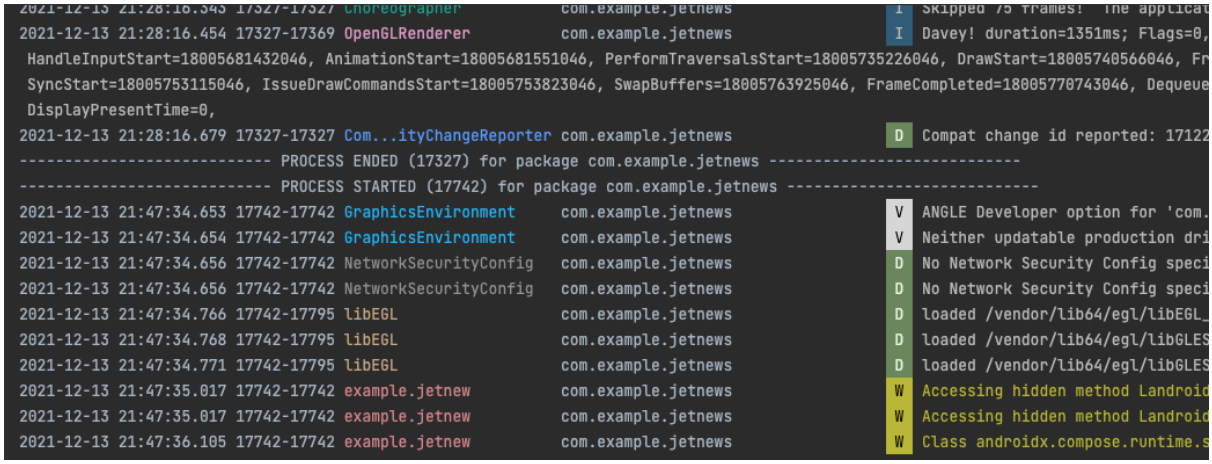


Рис. 3.3 Логування

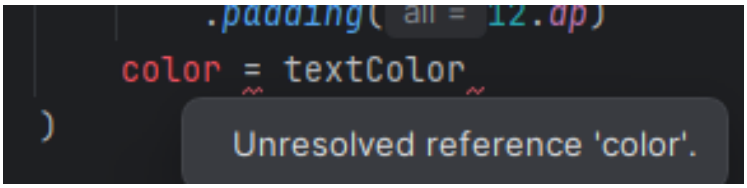


Рис. 3.4 Вікно аналізу коду.

Загалом AndroidStudio має безліч інструментів, які працюють в автоматичному і ручному режимах сильно полегшуючи розробку програмного забезпечення.

3.3 Розробка мобільного застосунку

Для правильної розробки застосунку, потрібно розуміти що він виконує та у якій послідовності. Для цього розроблено діаграму діяльності, що зображена на рисунку. 3.5.

Діаграма діяльності – це тип UML діаграми, на якому наочно видно послідовність дій, переходи між ними та логіку виконання процесу. Така діаграма складається із початку, дій, переходів, розгалужень, паралельних процесів та кінця.

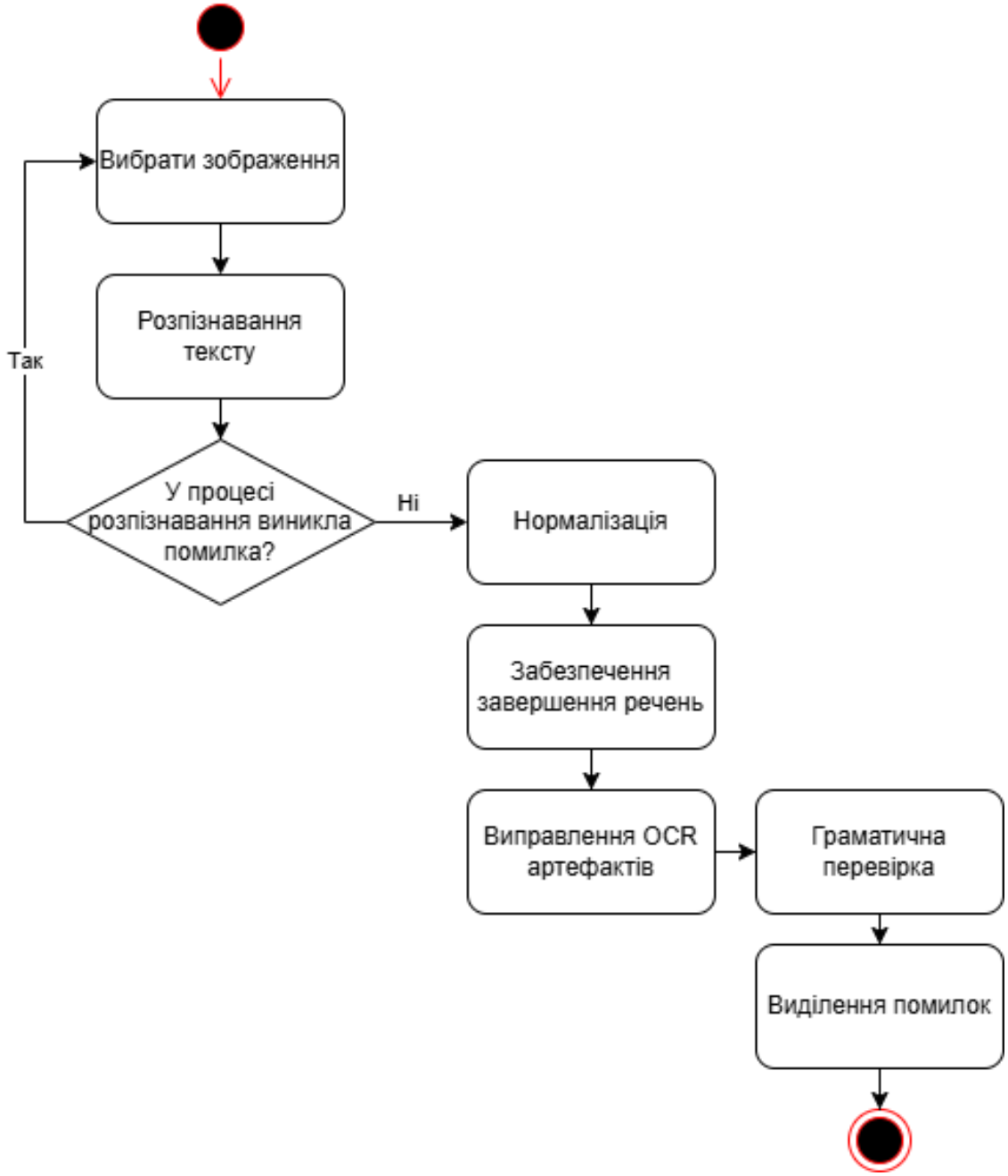


Рис. 3.5 Діаграма діяльності

Як видно із діаграми, користувач обирає зображення, яке обробляється системою. Після вибору зображення відбувається розпізнавання тексту. Якщо текст розпізнано неуспішно (виникла помилка), то модна спробувати ще раз, обравши це саме зображення, або ж інше. Далі виконується попередня обробка тексту – нормалізація, забезпечення завершення речень, виправлення OCR артефактів. Після цього виконується граматична перевірка, та виділення помилок.

Щоб наглядно зрозуміти структуру мобільного застосунку варто розглянути скорочену діаграму класів на рисунку 3.6. Із діаграми виключено класи, задачею яких є створення нових типів даних та зберігання даних (про них описано нижче в цьому підрозділі роботи).

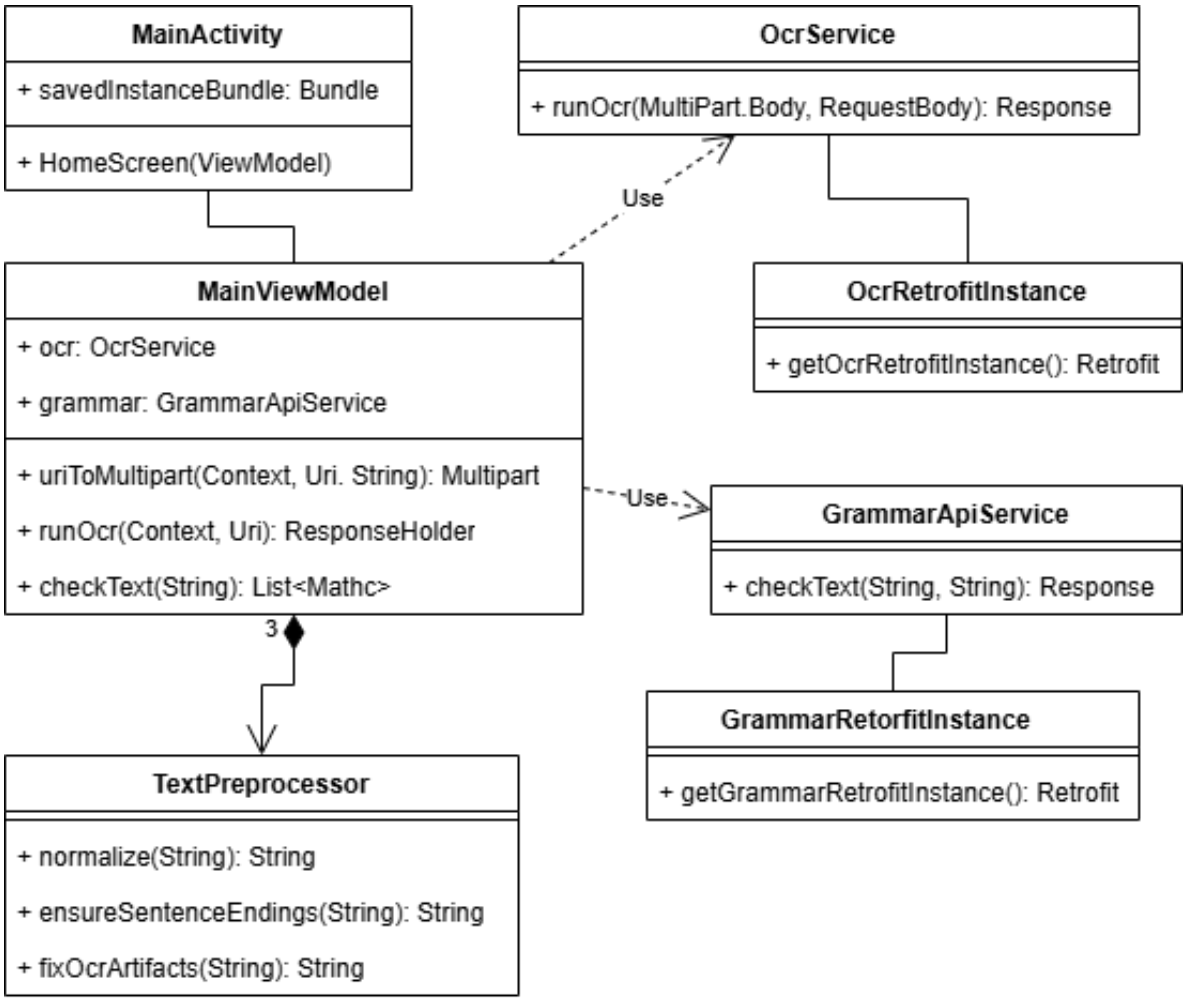


Рис. 3.6 Діаграма класів.

Клас «MainActivity» – вхідна точка до застосунку. У цьому класів задаються всі необхідні для початку роботи дані та виставляється інтерфейс користувача. За встановлення інтерфейсу користувача відповідає функція «HomeScreen».

Клас «MainViewModel» зв’язує логіку роботи застосунку та інтерфейс користувача. Це дозволяє розробити адаптивну систему, яка реагує на події та результати процесів. Функції цього класу описані у таблиці 3.1.

Таблиця 3.1

Функції класу «MainViewModel»

Функція	**Параметри**	**Тип**	**Опис**
uriToMultipart	1. Context 2. Uri 3. String	Part	Перетворює посилання на зображення у файл зображення з форматом «.jpg»
runOcr	1. Context 2. Uri	Unit	Розпочинає процес розпізнавання тексту і при отриманні результату обробляє його та передає до інтерфейсу користувача
checkText	String	Unit	Розпочинає процес попередньої обробки тексту і граматичної перевірки на помилки, після отримання результату обробляє його та передає до інтерфейсу користувача

Об’єкт «TextPreprocessor» – клас, який не має екземпляру, і виконується як «об’єкт». Даний клас відповідає за попередню обробку тексту після його розпізнавання. За це відповідають функції описані у таблиці 3.2.

Таблиця 3.2

Функції класу «TextProcessor»

Функція	**Параметри**	**Тип**	**Опис**
normalize	String	String	Нормалізує текст, видаляє зайві пробіли, пусті рядки.
ensureSentence Endings	String	String	Визначає межі речень та забезпечує завершення речень крапкою.
fixOcrArtifacts	String	String	Виправляє артефакти OCR.

Клас «OcrRetrofitInstance» відповідає за налаштування відправки запитів до сервісу OCRSpace, налаштування конвертації із JSON та обробку результатів запитів.

Інтерфейс «OcrService» відповідає за налаштування самих запитів до сервісу OCRSpace: параметри запитів, тіло запиту, тип запиту та інше.

Клас «GrammarRetrofitInstance» відповідає за налаштування відправки запитів до сервісу LanguageTool, налаштування конвертації із JSON та обробку результатів запитів.

Інтерфейс «GrammarApiService» відповідає за налаштування самих запитів до сервісу LanguageTool: параметри запитів, тіло запиту та інше.

Щоб правильно зберігати та обробляти дані, запити та їхні результати описано класи даних. Всього їх 15, описані вони у таблиці 3.3 та таблиці 3.4. Їх можна розділити на 2 групи – ті, що відповідають за роботу із сервісом OCRSpace (див. табл. 3.3) та ті, що відповідають за роботу із сервісом LanguageTool (див. табл. 3.4).

Таблиця 3.3

Класи даних «OcrResponse»

Клас	**Змінні**	
	Назва	**Тип**
ResponseHolder	parsedResults	List<ParsedResult>
	ocrExitCode	String
	isErroredOnProcessing	Boolean
	processingTimeInMilliseconds	String
	errorMessage	String
	errorDetails	String
	searchablePDFURL	String

Продовження таблиці 3.3

Класи даних «OcrResponse»

Клас	**Змінні**	
	Назва	**Тип**
ParsedResult	textOverlay	TextOverlay
	fileParseExitCode	String
	parsedText	String
	errorMessage	String
	errorDetails	String
TextOverlay	lines	List<Line>
	hasOverlay	Boolean
	message	String
Line	words	List<Word>
	maxHeight	Int
	minTop	Int
Word	wordText	String
	left	Int
	top	Int
	height	Int
	width	Int

Таблиця 3.4

Класи даних «GrammarApiResponse»

Клас	**Змінні**	
	Назва	**Тип**
LanguageToolResponse	software	Software
	language	Language
	matches	List<Match>
Software	name	String
	version	String
	buildGate	String
	apiVersion	Int
	status	String
	premium	Boolean
Language	name	String
	code	String
	detectedLanguage	DetectedLanguage
DetectedLanguage	name	String
	code	String
Match	message	String
	shortMessage	String
	offset	Int
	length	Int
	replacements	List<Replacement>
	context	Context
	sentence	String
	rule	Rule
Replacement	value	String

Продовження таблиці 3.4

Класи даних «GrammarApiResponse»

Клас	**Змінні**	
	Назва	**Тип**
Context	text	String
	offset	Int
	length	Int
Rule	id	String
	subId	String
	description	String
	urls	List<Url>
	issueType	String
	category	Category
Url	value	String
Category	id	String
	name	String

Інтерфейс користувача побудований у функції «HomeScreen». Для розробки інтрефейсу користувача використано фреймворк JetpackCompose, який дозволяє ефективно розробляти зручні та адаптивні інтерфейси. Для розробки ІК використано наступні функції:

– Scaffold: розмітка інтерфейсу на основі Material Design.

– FloatingActionButton: плаваюча кнопка, яка закликає до дії (зазвичай виконує основну дію на конкретному екрані).

– Icon: векторне зображення-іконка за правилами Material Design.

– LazyColumn: вертикальний список із підвантаженням елементів за вимогою.

– AsyncImage: зображення, яке асинхронно промальовується.

– Text.

– Row: горизонтальний список.

– Column: вертикальний список.

– Spacer: пусте місце.

Екранна форма тільки відкритого застосунку зображена на рисунку 3.7. Як видно з рисунку, користувач ще не обрав зображення і застосунок закликає до дій.

На рисунку 3.8 зображено екранну форму, коли користувач уже обрав зображення, система розпізнала текст та перевірила граматику.

На рисунку 3.9 додатково видно опис помилок, які виявила система при перевірці граматики. Також показано скільки часу у мілісекундах зайняло розпізнавання тексту та перевірки граматики з попередньою обробкою тексту.

На рисунках 3.7-3.9 видно такі елементи інтерфейсу користувача:

– Текст;

– Плаваюча кнопка;

– Зображення;

– Коробка.

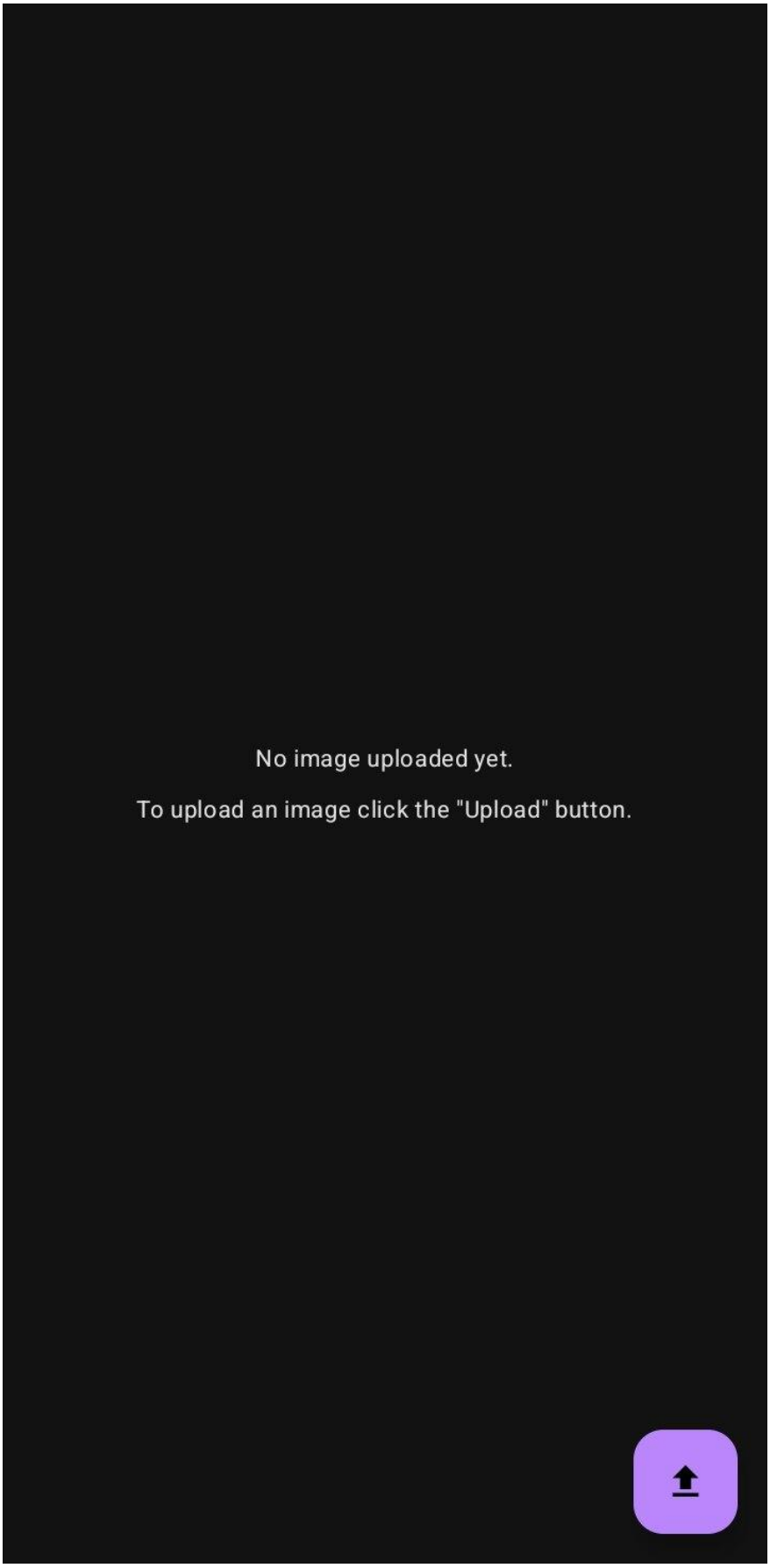


Рис. 3.7 Екранна форма відкритого застосунку

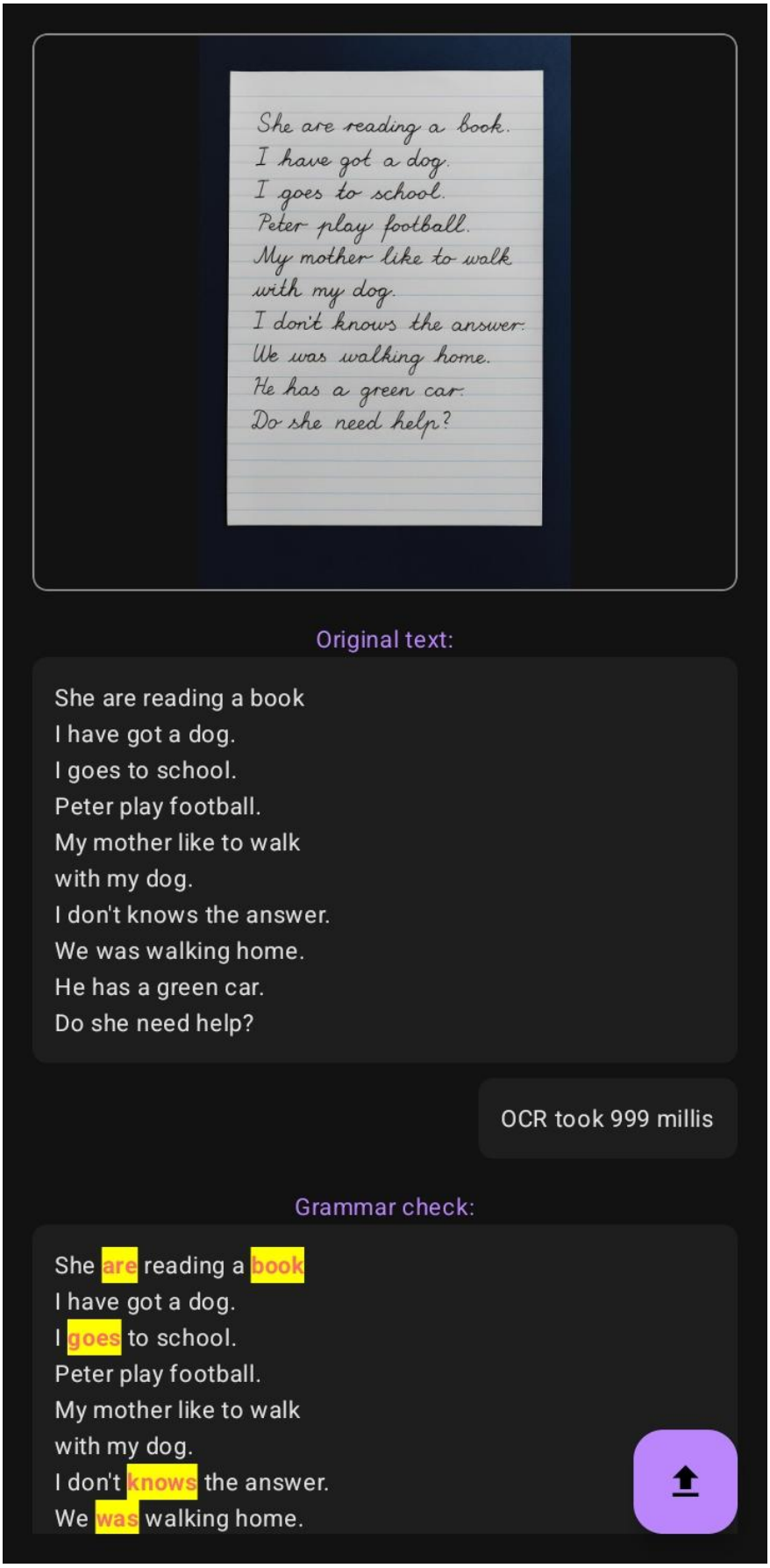


Рис. 3.8 Екранна форма з обраним зображенням

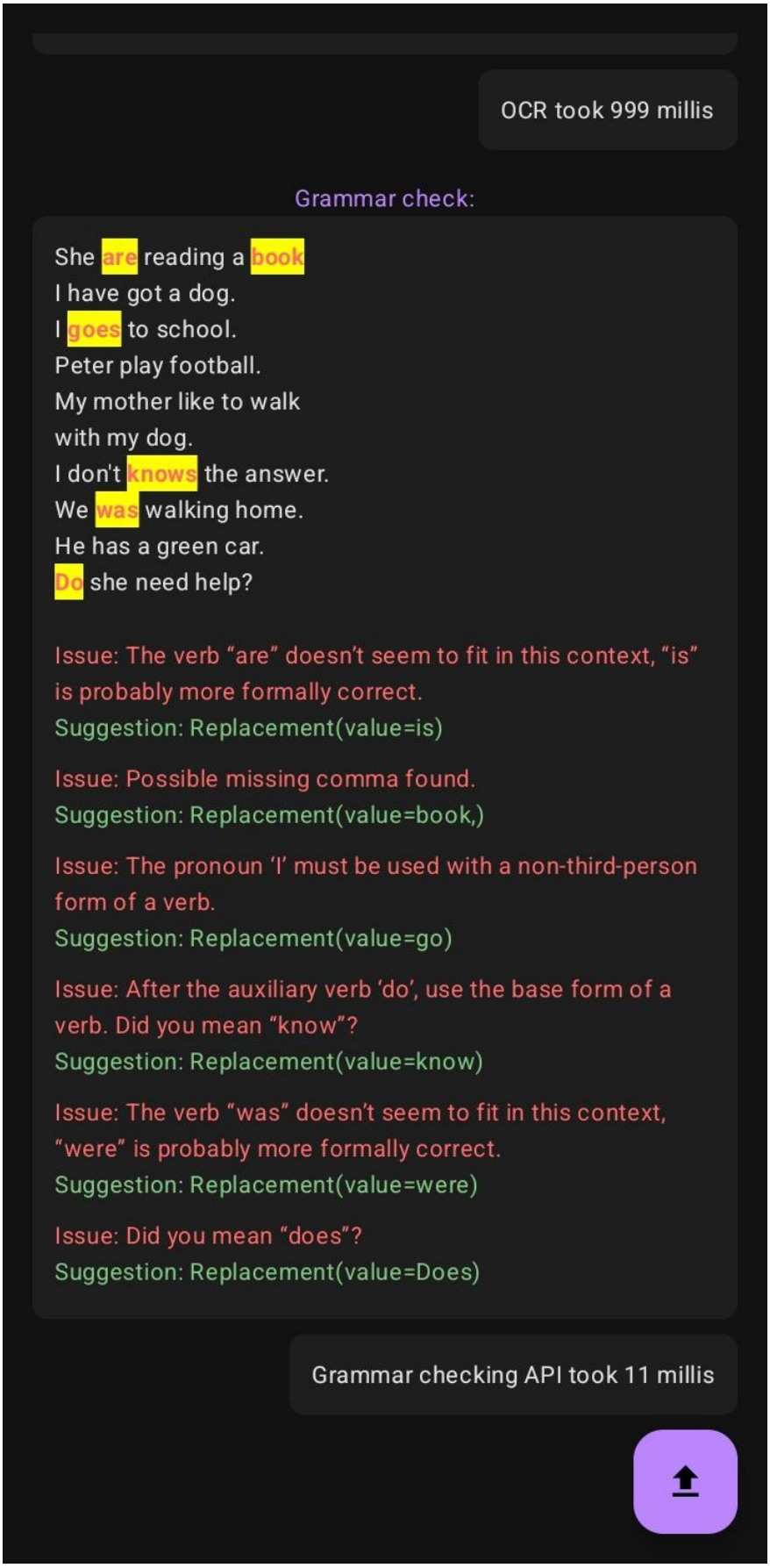


Рис. 3.9 Екранна форма із перевіреною граматикою

Для створення приємного інтерфейсу використовувалися різні прицнипи стилістики MaterialDesign:

– Елементи інтерфейсу поводяться як фізичні об’єкти.

– Використовуються тіні, глибина, поверхні, щоб створити відчуття реального матеріалу.

– Чітка типографіка, сітки, масштаб, колір, контраст.

– Плавні переходи.

– Готові компоненти з чіткими правилами поведінки і станів.

Для реалізації інтерфейсу користувача використано такі параметри як:

– Колір: використано власні значення кольорів.

– Заокруглення: підібрано заокруглення до елементів інтерфейсу користувача, щоб дизайн виглядав більш стильним та сучасним.

– Тіні: застосовано ефекти тіней для надання об’ємності елементам інтерфейсу користувача.

– Відступи: усі елементи мають відступи один від одного, щоб не зливатися та надавати читабельність.

– Вирівнювання: використано різні вирівнювання елементів, щоб забезпечити кращу структурну читабельність.

– Ієрархія: для послідовної демонстрації результатів використано вертикальний «список».

3.4 Проведення експерименту та порівняльний аналіз

Для проведення експерименту обрано низку зображень із текстом друкованим та написаним від руки. Текст містить граматичні помилки.

Порівняння відбувається за декількома критеріями – час роботи, точність розпізнавання, точність пошуку помилок. Час роботи описується як час, який було

потрачено на обробку одного символу. Тобто загальний час роботи системи поділений на кількість символів.

Для проведення експерименту обрано 6 зображень (див. рис. 3.10-3.15). Особливості зображень:

– Зображення 1 написане англійською мовою із розбірливим почерком, декількома простими граматичними помилками та хорошою якістю зображення.

– Зображення 2 написане англійською мовою із розбірливим почерком, декількома помилками у правописі та хорошою якістю зображення.

– Зображення 3 написане англійською мовою із складним почерком, декількома стилістичними та граматичними помилками та хорошою якістю зображення.

– Зображення 4 згенероване штучним інтелектом англійською мовою із середнім почерком, декількома помилками правопису та хорошою якістю зображення.

– Зображення 5 написане українською мовою із хорошим почерком та декількома помилками, якість зображення – низька роздільна здатність.

– Зображення 6 написане українською мовою із нормальним почерком та декількома помилками, якість зображення – хороша.

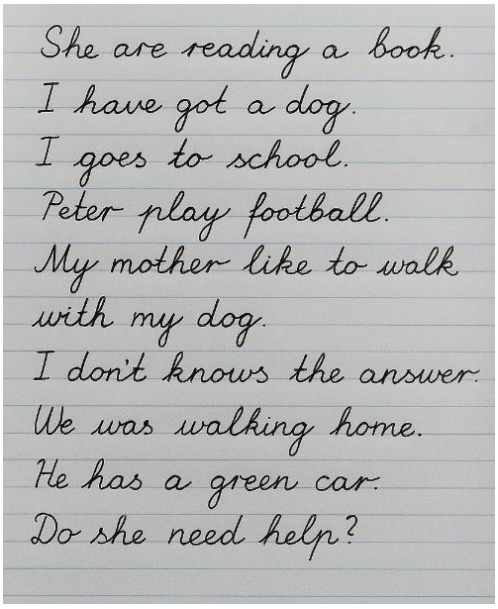


Рис. 3.10 Зображення 1

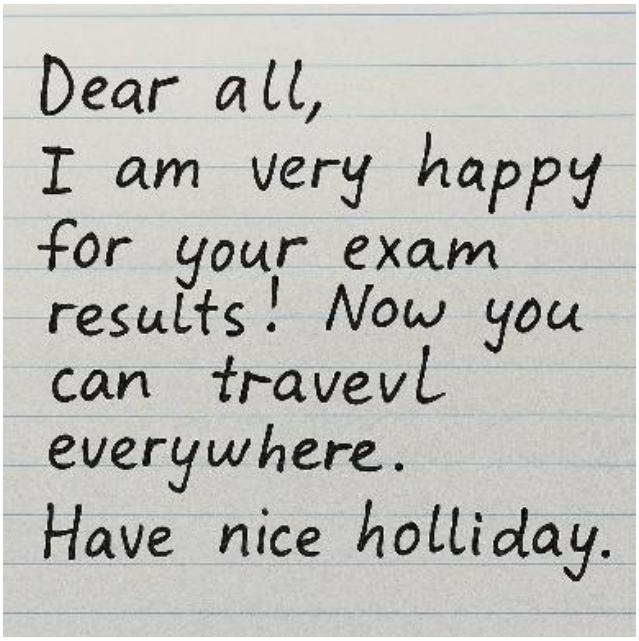


Рис. 3.11 Зображення 2

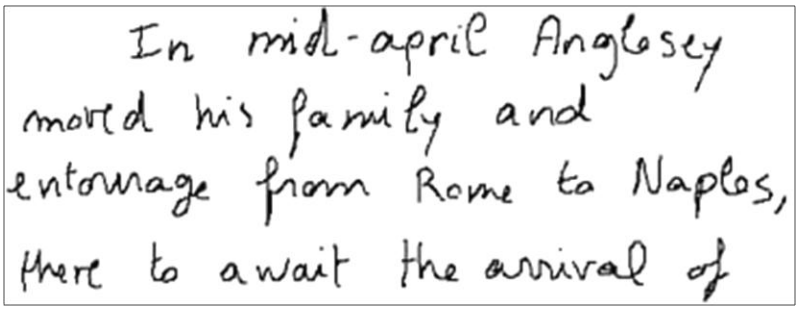


Рис. 3.12 Зображення 3

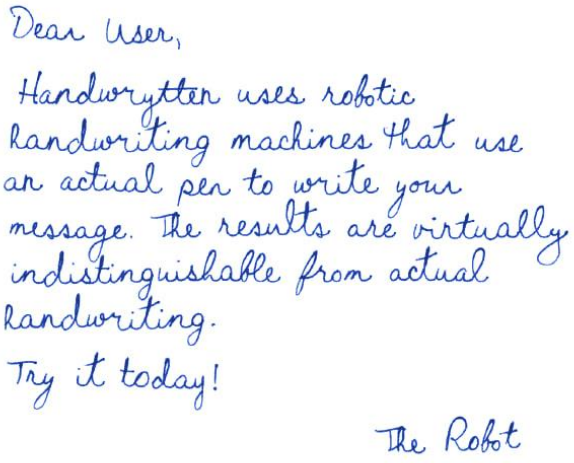
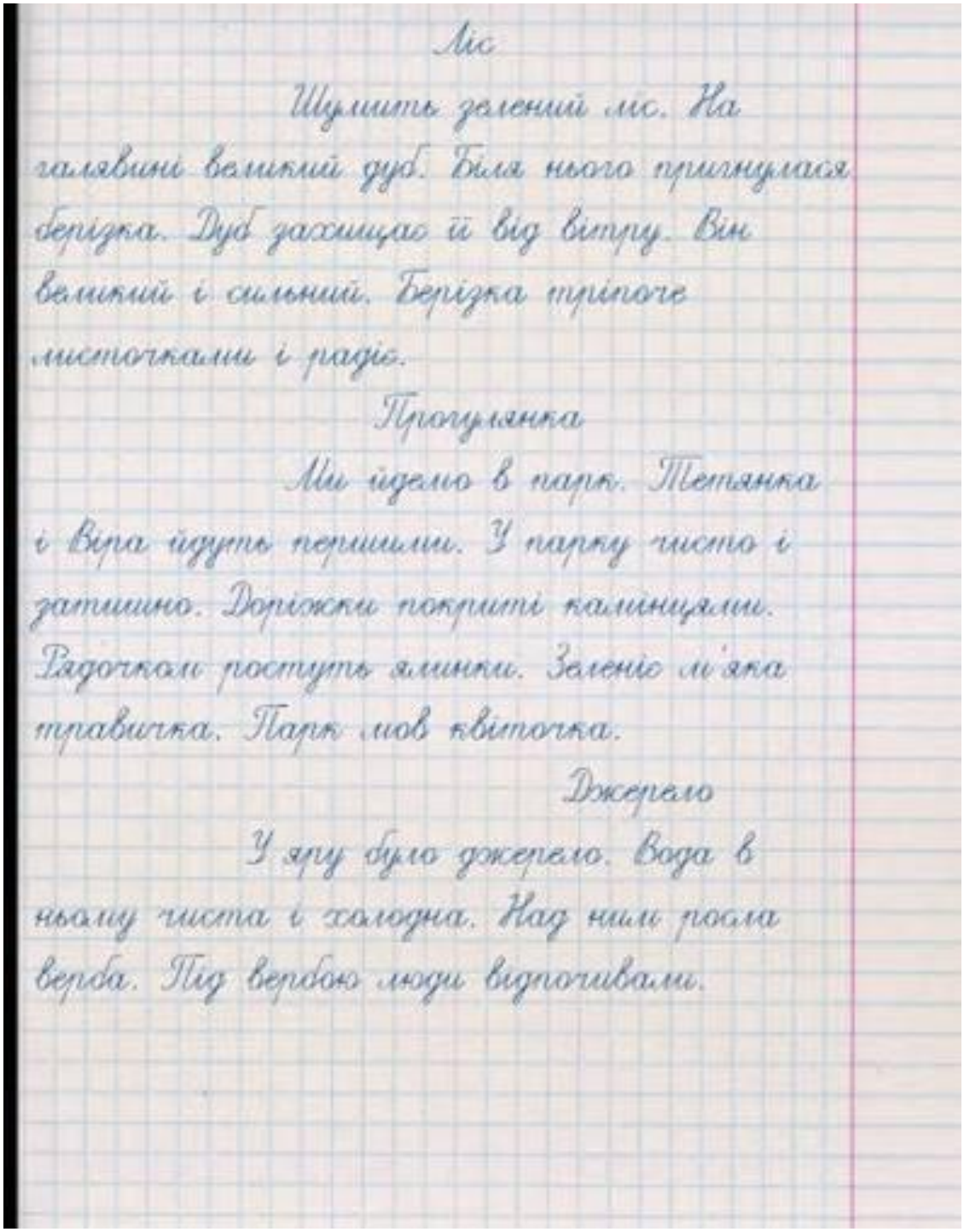


Рис. 3.13 Зображення 4



Ліс

Шумить зелений ліс. На галявині великий дуб. Біля нього пригнулася берізка. Дуб захищає її від вітру. Він великий і сильний. Берізка тріпоче листочками і радіє.

Прогулянка

Ми йдемо в парк. Тетянка і Віра йдуть першими. У парку чисто і затишно. Доріжки покриті камінцями. Рядочком ростуть ялинки. Зеленіє м'яка травичка. Парк мов квіточка.

Джерело

У яру було джерело. Вода в ньому чиста і холодна. Над ним росла верба. Під вербою люди відпочивали.

Рис. 3.14 Зображення 5

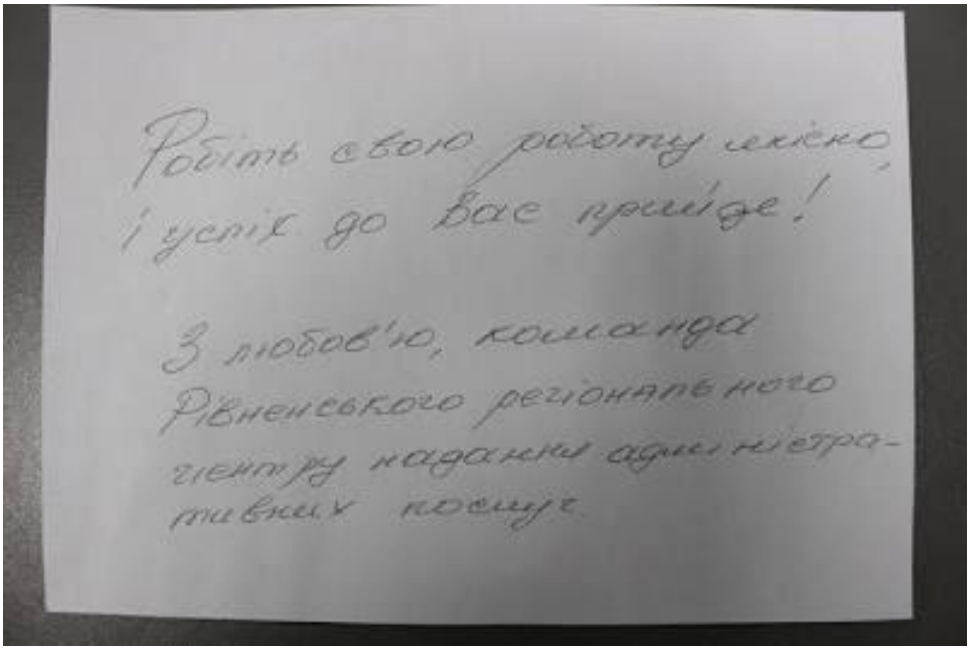


Рис. 3.15 Зображення 6

У таблиці 3.5 описано результати проведення експерименту із використанням зображень 3.10-3.15 над розробленим методом MPOPT.

Таблиця 3.5

Результати проведення експерименту над MPOPT

Зображення	Час	Точність OCR	Точність перевірки
1	0.007	97%	86%
2	0.010	99%	64%
3	0.006	95%	82%
4	0.003	92%	70%
5	0.003	90%	100%
6	0.023	87%	97%

У таблиці 3.6 продемонстровано результати роботи застосунку IMGProof із тими самими зображеннями. Неможливо оцінити точність розпізнавання IMGProof, оскільки застосунок не демонструє розпізнаний текст, а лише вказує на помилки.

Таблиця 3.6

Результати проведення експерименту над IMGProof

Зображення	Час (с/символ)	Точність перевірки
1	0.330	100%
2	1.199	100%
3	0.270	50%
4	0.262	88%
5	0.142	100%
6	0.240	100%

Таблиця 3.7 описує результати тестування Copilot.

Таблиця 3.7

Результати проведення експерименту над Copilot

Зображення	Час	Точність OCR	Точність перевірки
1	0.727	100%	100%
2	0.935	100%	100%
3	0.300	94%	66%
4	0.022	94%	70%
5	0.024	71%	90%
6	0.060	87%	96%

Таблиця 3.8 описує результати тестування Gemini.

Таблиця 3.8

Результати проведення експерименту над Gemini

Зображення	Час	Точність OCR	Точність перевірки
1	0.812	97%	93%
2	1.012	96%	61%
3	0.342	91%	60%
4	0.027	92%	67%
5	0.031	68%	84%
6	0.073	85%	83%

У таблиці 3.9 продемонстровано порівняльний аналіз МРОРТ із аналогами.

Таблиця 3.9

Порівняльний аналіз

Метод	**Час**	**Точність OCR**	**Точність перевірки**
МРОРТ	0.008	94%	83%
IMGProof	0.407	–	89%
Copilot	0.346	91%	87%
Gemini	0.383	88%	75%

У таблиці 3.9 взято середні показники за параметрами. Помітно, що ШІ, або методи із використанням штучного інтелекту помітно довше виконують роботу.

Варто зауважити, що результативність методів на основі штучного інтелекту може варіюватися в залежності від натренованості штучного інтелекту, вибірки даних та інших параметрів.

3.5 Висновки

Оглянуто інструменти розробки програмного забезпечення. Обрана мова програмування – Kotlin, середовище розробки – AndroidStudio. Інструменти мають ряд переваг: простоту, багатий функціонал, офіційна підтримка.

Розроблено мобільний застосунок із застосуванням методики MPOPT. Застосунок має одну екранну форму із варіаціями, та одну дію для користувача. Результати роботи алгоритмів демонструються користувачу послідовно.

Проведено експериментальне дослідження та порівняльний аналіз із аналогами: MPOPT має перевагу у швидкості обробки даних, де різниця у часі роботи може сягати декількох секунд та точності розпізнавання тексту на 3% від найближчого конкурента, проте програє за точністю пошуку помилок Copilot та IMGProof на 4-6%.

ВИСНОВКИ

Проаналізовано наявні методи розпізнавання рукописного тексту та пошуку помилок. Виявлено основні переваги та недоліки. Серед недоліків: низька точність, повільна робота, відсутність у деяких аналогів розпізнавання тексту.

Проведено огляд науково-технічної літератури, що дало змогу визначити напрямок подальшого дослідження, сфокусуватися на конкретних методах – сервіси OCRSpace, LanguageTool та алгоритми попередньої обробки тексту.

Розроблено математичну модель та методику розпізнавання рукописного тексту та його граматичної перевірки із застосування попередньої обробки тексту (МРОРТ).

Розроблено мобільний застосунок із використанням мови програмування Kotlin, середовища розробки AndroidStudio та методики МРОРТ для автоматизації процесу пошуку помилок у рукописному тексті.

Проведено тестування розробленого застосунку та порівняння з аналогами, що показало – даний підхід точніше розпізнає текст та швидше виконує роботу. Враховується 3 критерії оцінювання – швидкість, точність розпізнавання та точність граматичної перевірки. За цією оцінкою МРОРТ переважає швидше обробляє один символ на 0.3с. та 3% точніше розпізнає текст.

Робота пройшла апробацію у вигляді двох опублікованих тез:

1. Байса М. Ю., Худік Б. О. API для перевірки тексту на помилки. VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». – Київ: ДУІКТ, 2025. – С.635-637.

2. Байса М. Ю., Худік Б. О. OCR технології для зчитування тексту. VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». – Київ: ДУІКТ, 2025. – С.639-641.

ПЕРЕЛІК ПОСИЛАНЬ

1. Manuscripts. *Europeana*. URL: https://www.europeana.eu/en/search?qf=collection:manuscript.

2. Фонди інституту рукопису | Національна бібліотека України імені В.І.Вернадського. *Національна бібліотека України імені В. І. Вернадського*. URL: http://www.nbuv.gov.ua/node/6722.

3. Baranetskyi Y., Kunanets N. Recognition of handwritten text: modern approaches and challenges. *Humanities science current issues*. 2025. Vol. 1, no. 83. P. 190–198. URL: https://doi.org/10.24919/2308-4863/83-1-29.

4. Haoran Wei. General OCR Theory: Towards OCR-2.0 via a Unified End-to-end Model. DOI: https://doi.org/10.48550/arXiv.2409.01704.

5. Chi Nguyen. Utilizing Google's Machine Learning Kit in Developing Android Application. Metropolia University of Applied Sciences, 15 April 2024. Режим доступу: https://www.theseus.fi/bitstream/handle/10024/858630/Nguyen_Chi.pdf?sequence=2.

6. Kiruthika Subramani. Document AI in GCP. URL: https://medium.com/@techwithkrithi/document-ai-in-gcp-1d157f8893c1.

7. Prashant Johri, Sunil K. Khatri, Ahmad T. Al-Taani, Munish Sabharwal, Shakhzod Suvanov, Avneesh Kumar. Natural Language Processing: History, Evolution, Application, and Future Work. DOI: http://dx.doi.org/10.1007/978-981-15-9712-1_31.

8. Medium. The ultimate azure OCR guide: a performance comparison of the top text recognition APIs. URL: https://medium.com/@datalab_70093/the-ultimate-azure-ocr-guide-a-performance-comparison-of-the-top-text-recognition-apis-fd2f417bc198.

9. О. Г. Ворочек, І. В. Соловей. Використання мовних моделей штучного інтелекту для генерації публікацій у соціальних мережах. DOI: https://doi.org/10.26642/ten-2024-1(93)-128-134.

10. Nadia Wood. A review of Google's Cloud Natural Language API. URL: https://www.researchgate.net/publication/351233913_A_Review_of_Google's_Cloud_Natural_Language_API_Service.

11. Intro to Apache OpenNLP. URL: https://www.baeldung.com/apache-open-nlp.

12. API Reference. NLP Cloud. URL: https://docs.nlpcloud.com/#introduction.

13. OpenAI Platform. Models. URL: https://platform.openai.com/docs/models.

14. Gemini Developer API | Gemma open models | Google AI for Developers. Google AI for Developers. URL: https://ai.google.dev/.

15. CamScanner. URL: https://www.camscanner.com/?form=MG0AV3.

16. Adobe. Мобільні пристрої. URL: https://www.adobe.com/ua/acrobat/mobile/scanner-app.html.

17. Grammarly. Features. URL: https://www.grammarly.com/features.

18. ProWritingAid. Features. URL: https://prowritingaid.com/features.

19. SlickWrite. URL: https://www.slickwrite.com/#!home.

20. Survey of Post-OCR Processing Approaches / T. T. H. Nguyen et al. ACM Computing Surveys. 2021. Vol. 54, no. 6. P. 1–37. URL: https://doi.org/10.1145/3453476.

21. URL: http://www.impact-project.eu/about-the-project/concept/.

22. Jain P., Taneja D. K., Taneja D. H. Which OCR toolset is good and why? A comparative study. Kuwait Journal of Science. 2021. Vol. 48, no. 2. URL: https://doi.org/10.48129/kjs.v48i2.9589.

23. Van Strien, D., Beelen, K., Ardanuy, M., Hosseini, K., McGillivray, B., & Colavizza, G. (2020). Assessing the impact of OCR quality on downstream NLP tasks. SCITEPRESS – Science and Technology Publications. https://doi.org/10.17863/CAM.52068.

24. Handwritten Optical Character Recognition (OCR): A Comprehensive Systematic Literature Review (SLR) / J. Memon et al. IEEE Access. 2020. Vol. 8. P. 142642–142668. URL: https://doi.org/10.1109/access.2020.3012542.

25.Najam R., Faizullah S. Analysis of Recent Deep Learning Techniques for Arabic Handwritten-Text OCR and Post-OCR Correction. Applied Sciences. 2023. Vol. 13, no. 13. P. 7568. URL: https://doi.org/10.3390/app13137568.

26. Kostiantyn Liepieshov. On Recognition of Cyrillic Text. 2019. URL: https://openreview.net/pdf?id=Ske6GT9c8r.

27. Farrar, Christopher, "Real-time Automatic Text Correction Using Artificial Intelligence", Technical Disclosure Commons, (September 02, 2020). URL: https://www.tdcommons.org/dpubs_series/3575.

28. Grammatical Error Correction: A Survey of the State of the Art / C. Bryant et al. Computational Linguistics. 2023. P. 1–59. URL: https://doi.org/10.1162/coli_a_00478.

29. About the free OCR API. *Free OCR API V2025, Online OCR, Searchable PDF Creator and OCR Software*. URL: https://ocr.space/about.

30. Документація LanguageTool. URL: https://dev.languagetool.org/.

31. Orlovskyi O., Ostapov S. Analysis of the text preprocessing methods influence on the destructive messages classifier. *Advanced information systems*. 2020. Vol. 4, no. 3. P. 104–108. URL: https://doi.org/10.20998/2522-9052.2020.3.14 (date of access: 18.12.2025).

32. Dudka M. Чому Kotlin може бути чудовим вибором для вашого проєкту. *DOU*. URL: https://dou.ua/forums/topic/45457/.

33. Shyian I. O., Krasheninnik I. O. Overview of development platforms and programming languages for learning mobile software development. *Innovate pedagogy*. 2024. No. 75. P. 309–312. URL: https://doi.org/10.32782/2663-6085/2024/75.59.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

«Методика аналізу рукописних текстів для перевірки на помилки»

Виконав: студент групи ПДМ -62 Максим БАЙСА

Керівник: доктор філософії (PhD), доцент кафедри ІПЗ Богдан ХУДІК

Київ - 2025

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: автоматизація пошуку помилок у тексті, написаному від руки, за рахунок поєднання різних методів розпізнавання та обробки тексту.

Об’єкт дослідження: процес розпізнавання рукописного тексту та пошуку помилок у ньому.

Предмет дослідження: технології розпізнавання рукописного тексту, методи обробки природньої мови та пошуку помилок у тексті.

АКТУАЛЬНІСТЬ РОБОТИ

Модель / Метод	Переваги	Недоліки
Grammarly	- Висока точність граматичного аналізу - Підтримка стилістики - API - Інтеграція з Word	- Не працює з зображеннями – повинен бути готовий цифровий текст
Copilot	- Комбінує OCR + граматику - Підтримує українську - Гнучкий API - Працює з зображеннями	- Може плутати контекст - Не завжди стабільна точність на рукописах
Gemini	- Висока точність OCR - Швидка обробка - Багатомовна підтримка - ML-оптимізація	- Обмежена точність граматики - Не завжди коректно обробляє пунктуацію
IMGProof	- Перевіряє текст прямо в зображеннях - Підсвічує помилки на картинці - Швидкий аналіз	- Низька точність пошуку помилок - Слабка підтримка багатомовності

3

МЕТОДИКА РОЗПІЗНАВАННЯ ТА ГРАМАТИЧНОЇ ПЕРЕВРІКИ РУКОПИСНОГО ТЕКТСУ

4

МАТЕМАТИЧНА МОДЕЛЬ МРОРТ

$$F = G \cdot N \cdot R$$

R – функція розпізнавання рукописного тексту,
N – функція попередньої обробки тексту,
G – функція пошуку помилок

$$R = X \to T \qquad N = T \to T' \qquad G = T' \to E$$

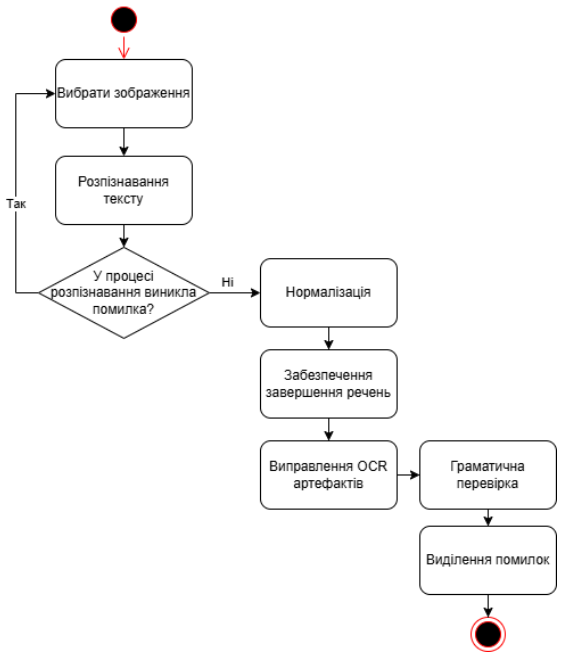
X – вхідне зображення,
T – розпізнаний текст,
T′ – текст після попередньої обробки,
E – множина знайдених помилок

$$E = \{(p_k, e_k)\}_{k=1}^{K}$$

p_k – позиція помилки,
e_k – опис помилки,
K – множина значень помилок

5

АЛГОРИТМ РОЗПІЗНАВАННЯ ТА ГРАМАТИЧНОЇ ПЕРЕВІРКИ ТЕКСТУ

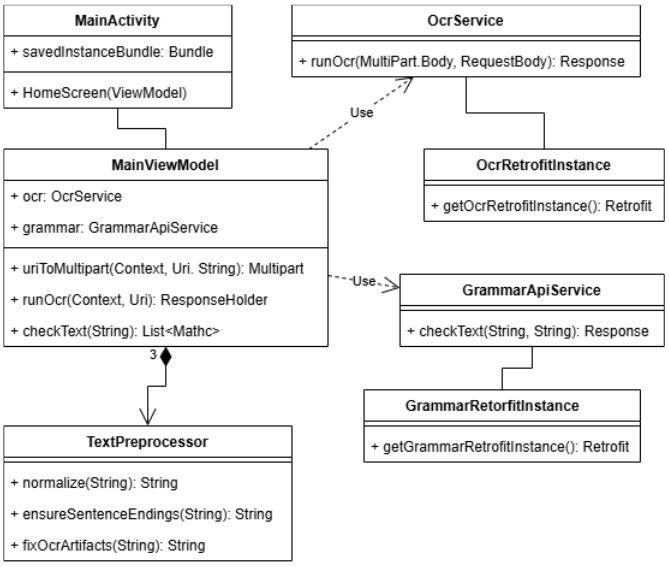


6

ПОПЕРЕДНЯ ОБРОБКА ТЕКСТУ

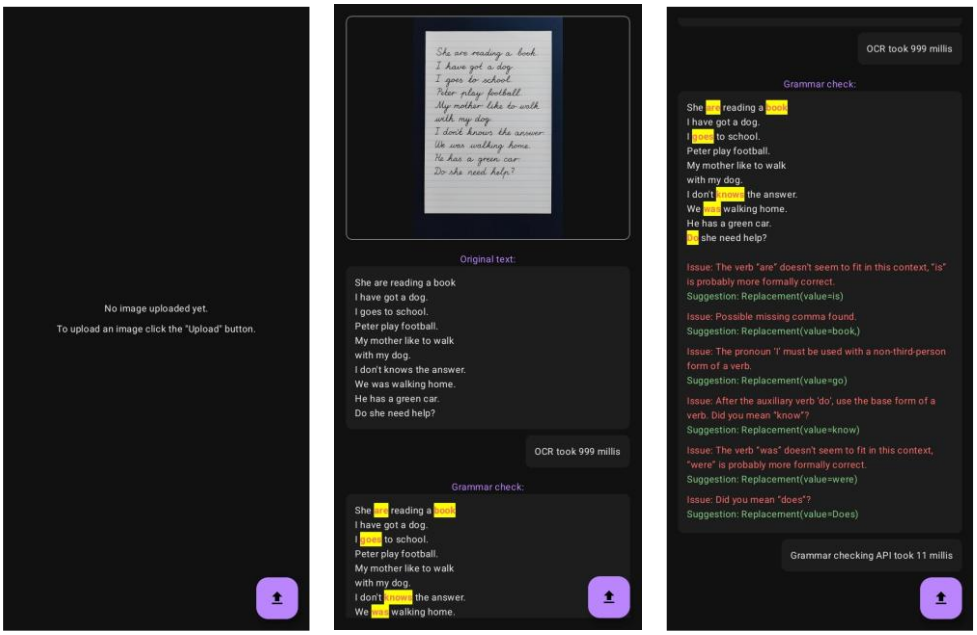
7

ДІАГРАМА КЛАСІВ МОБІЛЬНО ЗАСТОСУНКУ



8

ЕКРАННІ ФОРМИ МОБІЛЬНОГО ЗАСТОСУНКУ



9

ПОРІВНЯЛЬНИЙ АНАЛІЗ

Метод	Час	Точність OCR	Точність перевірки
MPOPT	0.008	94%	83%
IMGProof	0.407	–	89%
Copilot	0.346	91%	87%
Gemini	0.383	88%	75%

ВИСНОВКИ

1. Проведено аналіз наявних методів розпізнавання тексту та пошуку помилок у ньому. Виявлено переваги і недоліки: відсутність розпізнавання тексу (Grammarly), низька точність розпізнавання тексту, низька точність пошуку помилок у тексті.

2. Розроблено методику розпізнавання рукописного тексту та пошуку помилок у ньому на основі сервісів OCRSpace та LanguageTool із додаванням попередньої обробки тексту.

3. Розроблено мобільний застосунок для наочності роботи запропонованої методики. Використано мову програмування Kotlin, інтегроване середовище розробки AndroidStudio.

4. Проведено тестування розробленої системи та порівняння результатів її роботи з наявними методами. Виявлено, що час обробки одного символу приблизно на 0.3 с швидше, точність розпізнавання на 3% вища.

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Тези доповідей:

1. Байса М. Ю., Худік Б. О. API для перевірки тексту на помилки. VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». – Київ: ДУІКТ, 2025. – С.635-637.
2. Байса М. Ю., Худік Б. О. OCR технології для зчитування тексту. VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». – Київ: ДУІКТ, 2025. – С.639-641.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
data class LanguageToolResponse(
    val software: Software,
    val language: Language,
    val matches: List<Match>
)

data class Software(
    val name: String,
    val version: String,
    val buildDate: String,
    val apiVersion: Int,
    val status: String,
    val premium: Boolean
)

data class Language(
    val name: String,
    val code: String,
    val detectedLanguage: DetectedLanguage
)

data class DetectedLanguage(
    val name: String,
    val code: String
)

data class Match(
    val message: String,
    val shortMessage: String?,
    val offset: Int,
    val length: Int,
    val replacements: List<Replacement>,
    val context: Context,
    val sentence: String,
    val rule: Rule
)

data class Replacement(
    val value: String
)

data class Context(
    val text: String,
    val offset: Int,
    val length: Int
)

data class Rule(
    val id: String,
    val subId: String?,
    val description: String,
    val urls: List<Url>,
    val issueType: String,
    val category: Category
)

data class Url(
    val value: String
)

data class Category(
    val id: String,
    val name: String
)

interface GrammarApiService {
    @FormUrlEncoded
    @POST("check")
    suspend fun checkText(
        @Field("text") text: String,
        @Field("language") language: String =
"auto"
    ): Response<LanguageToolResponse>
}

class GrammarRetrofitInstance {
    companion object {
        const val BASE_URL =
"https://api.languagetool.org/v2/"
        fun getGrammarRetrofitInstance(): Retrofit
{
            return Retrofit.Builder()
                .baseUrl(BASE_URL)
.addConverterFactory(GsonConverterFactory.create(Gso
nBuilder().create()))
                .build()
        }
    }
}

data class ResponseHolder(
@SerializedName("ParsedResults")
val parsedResults: List<ParsedResult>?,
@SerializedName("OCRExitCode")
val ocrExitCode: String?,
@SerializedName("IsErroredOnProcessing")
    val isErroredOnProcessing: Boolean?,

@SerializedName("ProcessingTimeInMilliseconds")
    val processingTimeInMilliseconds: String?,
    @SerializedName("ErrorMessage")
    val errorMessage: String?,
    @SerializedName("ErrorDetails")
    val errorDetails: String?,
    @SerializedName("SearchablePDFURL")
    val searchablePDFURL: String?
)

data class ParsedResult(
    @SerializedName("TextOverlay")
    val textOverlay: TextOverlay?,
@SerializedName("FileParseExitCode")
    val fileParseExitCode: String?,
@SerializedName("ParsedText")
```

```
        val parsedText: String?,
@SerializedName("ErrorMessage")
        val errorMessage: String?,
@SerializedName("ErrorDetails")
        val errorDetails: String?
    )

    data class TextOverlay(
        @SerializedName("Lines")
        val lines: List<Line>?,
        @SerializedName("HasOverlay")
        val hasOverlay: Boolean?,
        @SerializedName("Message")
        val message: String?
    )

    data class Line(
        @SerializedName("Words")
        val words: List<Word>?,
        @SerializedName("MaxHeight")
        val maxHeight: Int?,
        @SerializedName("MinTop")
        val minTop: Int?
    )

    data class Word(
        @SerializedName("WordText")
        val wordText: String?,
        @SerializedName("Left")
        val left: Int?,
        @SerializedName("Top")
        val top: Int?,
        @SerializedName("Height")
        val height: Int?,
        @SerializedName("Width")
        val width: Int?
    )
    class OcrRetrofitInstance {
        companion object {
            const val BASE_URL =
"https://api.ocr.space/"
            fun getOcrRetrofitInstance(): Retrofit {
                return Retrofit.Builder()
                    .baseUrl(BASE_URL)

.addConverterFactory(GsonConverterFactory.create(Gso
nBuilder().create()))
                    .build()
            }
        }
    }

    interface OcrService {
        @POST("parse/image")
        @Multipart
        @Headers("apikey: K81750964988957")
        suspend fun runOcr(
            @Part file: MultipartBody.Part,
            @Part("language") language: RequestBody,
            @Part("scale") scale: RequestBody,
            @Part("OCREngine") ocrEngine:
RequestBody
        ): Response<ResponseHolder>

    }

    object TextPreprocessor {
        fun normalize(text: String): String {
            var normalized = text
                .replace("\n", " ")
                .replace("\\s+".toRegex(), " ")
                .trim()
            normalized =
ensureSentenceEndings(normalized)
            return normalized
        }
        private fun ensureSentenceEndings(text:
String): String {
            val sentences =
text.split("(?<=[.!?])".toRegex())
                .map { it.trim() }
                .filter { it.isNotEmpty() }
            return sentences.joinToString(" ") {
sentence ->
                if (sentence.matches(Regex(".*[.!?]$")))
sentence else "$sentence."
            }
        }

        fun fixOcrArtifacts(text: String): String {
            return text
                .replace("rn", "m")
                .replace("0", "o")
                .replace("1", "l")
        }

    }

    @Composable
    fun HomeScreen(viewModel: MainViewModel)
{

        val context = LocalContext.current

        var originalText by rememberSaveable {
mutableStateOf("") }
        var matches by rememberSaveable {
mutableStateOf<List<Match>>(emptyList()) }
        var selectedImageUri by rememberSaveable {
mutableStateOf<Uri?>(null) }
        var ocrJobTime by rememberSaveable {
mutableStateOf("") }
        var grammarJobTime by rememberSaveable {
mutableLongStateOf(0) }

        val imagePickerLauncher =
rememberLauncherForActivityResult(
            contract =
ActivityResultContracts.PickVisualMedia(),
            onResult = { uri ->
                selectedImageUri = uri
                if (uri != null) {
                    viewModel.runOcr(context, uri) {
result ->
```

```
                    if (result != null) {
                        originalText = result.parsedResults?.first()?.parsedText ?: ""
                        if (result.processingTimeInMilliseconds != null)
                            ocrJobTime = result.processingTimeInMilliseconds
                        grammarJobTime = measureTimeMillis {

viewModel.checkText(originalText) { result ->
                            matches = result
                        }
                    }
                }
            }
        }
    )

    val backgroundColor = Color(0xFF121212)
    val surfaceColor = Color(0xFF1E1E1E)
    val textColor = Color(0xFFE0E0E0)
    val accentColor = Color(0xFFBB86FC)
    val errorColor = Color(0xFFFF6B6B)
    val successColor = Color(0xFF81C784)

    Scaffold(
        containerColor = backgroundColor,
        floatingActionButton = {
            FloatingActionButton(
                onClick = {
                    imagePickerLauncher.launch(

PickVisualMediaRequest(ActivityResultContracts.PickVisualMedia.ImageOnly)
                    )
                },
                containerColor = accentColor,
                contentColor = Color.Black
            ) {
                Icon(Icons.Filled.FileUpload, contentDescription = "Upload")
            }
        }
    ) { paddingValues ->

        val modifier = Modifier
            .fillMaxSize()
            .padding(paddingValues)
            .padding(16.dp)

        if (selectedImageUri != null) {
            LazyColumn(
                modifier = modifier,
                verticalArrangement = Arrangement.spacedBy(16.dp),
                horizontalAlignment = Alignment.CenterHorizontally
            ) {
                item {
                    AsyncImage(
                        model = selectedImageUri,
                        contentDescription = "Selected Image",
                        modifier = Modifier
                            .fillMaxWidth()
                            .heightIn(max = 300.dp)

.clip(RoundedCornerShape(8.dp))
                            .border(1.dp, Color.Gray, RoundedCornerShape(8.dp))
                    )
                }

                item {
                    Text("Original text:", color = accentColor)
                    Text(
                        text = originalText,
                        modifier = Modifier
                            .fillMaxWidth()
                            .background(surfaceColor, RoundedCornerShape(8.dp))
                            .padding(12.dp),
                        color = textColor
                    )
                    Row(
                        modifier = Modifier
                            .padding(top = 8.dp)
                            .fillMaxWidth(),
                        horizontalArrangement = Arrangement.End
                    ) {
                        Text(
                            text = "OCR took $ocrJobTime millis",
                            modifier = Modifier
                                .background(surfaceColor, RoundedCornerShape(8.dp))
                                .padding(12.dp),
                            color = textColor
                        )
                    }
                }

                item {
                    Text("Grammar check:", color = accentColor)

                    Column(
                        modifier = Modifier
                            .fillMaxWidth()
                            .background(surfaceColor, RoundedCornerShape(8.dp))
                            .padding(12.dp)
                    ) {
                        val annotatedText = buildAnnotatedString {
                            append(originalText)
                            matches.forEach { s ->
                                addStyle(
                                    style = SpanStyle(
```

```
                                    background = Color.Yellow,
                                    color = errorColor,
                                    fontWeight = FontWeight.Bold
                                ),
                                start = s.offset,
                                end = s.offset + s.length
                            )
                        }
                    }

                    Text(annotatedText, color = textColor)

                    Spacer(modifier = Modifier.height(16.dp))

                    matches.forEach { s ->
                        Column(
                            modifier = Modifier
                                .fillMaxWidth()
                                .padding(vertical = 4.dp)
                        ) {
                            Text("Issue: ${s.message}", color = errorColor)
                            s.replacements.forEach { replacement ->
                                Text("Suggestion: $replacement", color = successColor)
                            }
                        }
                    }
                }

                Row(
                    modifier = Modifier
                        .padding(top = 8.dp)
                        .fillMaxWidth(),
                    horizontalArrangement = Arrangement.End
                ) {
                    Text(
                        text = "Grammar checking API took $grammarJobTime millis",
                        modifier = Modifier
                            .background(surfaceColor, RoundedCornerShape(8.dp))
                            .padding(12.dp),
                        color = textColor
                    )
                }

                Spacer(modifier = Modifier.height(64.dp))
            }
        }
    } else {
        Column(
            modifier = modifier,
            verticalArrangement = Arrangement.Center,
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            Text("No image uploaded yet.", color = textColor)
            Spacer(modifier = Modifier.height(8.dp))
            Text("To upload an image click the \"Upload\" button.", color = textColor)
        }
    }
}

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val mainViewModel = MainViewModel(
            OcrRetrofitInstance.getOcrRetrofitInstance().create(OcrService::class.java),
            GrammarRetrofitInstance.getGrammarRetrofitInstance().create(GrammarApiService::class.java)
        )
        enableEdgeToEdge()
        setContent {
            Thesis2025MockupTheme {
                HomeScreen(mainViewModel)
            }
        }
    }
}

class MainViewModel(
    private val ocr: OcrService,
    private val grammar: GrammarApiService
) : ViewModel() {

    fun uriToMultipart(
        context: Context,
        uri: Uri,
        paramName: String = "file"
    ): MultipartBody.Part {
        val inputStream = context.contentResolver.openInputStream(uri)
        val tempFile = File.createTempFile("upload", ".jpg", context.cacheDir)
        tempFile.outputStream().use { output ->
            inputStream?.copyTo(output)
        }
        val requestFile = tempFile.asRequestBody("image/*".toMediaTypeOrNull())
        return MultipartBody.Part.createFormData(paramName, tempFile.name, requestFile)
    }
```

```
        fun runOcr(context: Context, uri: Uri, onResult: (ResponseHolder?) -> Unit) {
            viewModelScope.launch {
                val filePart = uriToMultipart(context, uri)
                val languagePart = "eng".toRequestBody("text/plain".toMediaTypeOrNull())
                val scalePart = "true".toRequestBody("text/plain".toMediaTypeOrNull())
                val ocrEnginePart = "2".toRequestBody("text/plain".toMediaTypeOrNull())

                try {
                    val response = ocr.runOcr(filePart, languagePart, scalePart, ocrEnginePart)

                    if (response.isSuccessful) {
                        val body = response.body()
                        if (body != null) {
                            Log.d("OcrViewModel", "Parsed results: ${body.parsedResults?.first()?.parsedText}")
                            onResult(response.body())
                        } else {
                            Log.e("OcrViewModel", "Response body is null")
                            onResult(null)
                        }
                    }
                    else {
                        Log.e("OcrViewModel", "Server error: ${response.code()}")
                        onResult(null)
                    }
                } catch (e: Exception) {
                    Log.e("OcrViewModel", "Network error: ${e.message}", e)
                    onResult(null)
                }
            }
        }

        fun checkText(text: String, onResult: (List<Match>) -> Unit) {
            viewModelScope.launch {
                val normalized = TextPreprocessor.normalize(text)
                val finalText = TextPreprocessor.fixOcrArtifacts(normalized)
                try {
                    val response = grammar.checkText(finalText, "en")
                    if (response.isSuccessful) {
                        val matches = response.body()?.matches ?: emptyList()
                        Log.d("GrammarViewModel", "Matches: $matches")
                        onResult(matches)
                    } else {
                        Log.e("GrammarViewModel", "Server error: ${response.code()}")
                        onResult(emptyList())
                    }
                } catch (e: Exception) {
                    Log.e("GrammarViewModel", "Network error: ${e.message}", e)
                    onResult(emptyList())
                }
            }
        }
    }
```