НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

«На правах рукопису»
УДК 004.93’1

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
«___»_____2025 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

зі спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Програмне забезпечення для виявлення та розпізнавання тексту в документах нетекстового формату»

Виконав (-ла):
студент (-ка) ІІ курсу, групи ІП-з41мп
Дишкант Лариса Леонідівна _____

Керівник:
доцент кафедри ІПІ, к.т.н., доц.,
Крамар Юлія Михайлівна _____

Рецензент:
доцент кафедри ІСТ, к.т.н.,
Амонс Олександр Анатолійович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2025 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення інформаційних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Едуард ЖАРІКОВ

«___»_____2025р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Дишкант Ларисі Леонідівні

1. Тема дисертації «Програмне забезпечення виявлення та розпізнавання тексту в документах не текстового формату», науковий керівник дисертації Крамар Юлія Михайлівна, к.н.т., доц., затверджені наказом по університету від «06» листопада 2025 р. № 4842-с

2. Термін подання студентом дисертації «15» грудня 2025 р.

3. Об’єкт дослідження – програмне забезпечення розпізнавання тексту

4. Предмет дослідження – метод побудови програмного забезпечення виявлення та розпізнавання тексту в документах не текстового формату

5. Перелік завдань, які потрібно розробити – аналіз проблеми та існуючих рішень; розробка програмного забезпечення; дослідження ефективності розробленого програмного забезпечення.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу – 2-3 плакати

7. Орієнтовний перелік публікацій – одна публікація

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9.Дата видачі завдання «1» вересня 2025 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання	Примітка
1	Видача завдання	1.09.2025	
2	Визначення структури магістерської дисертації	5.09.2025	
3	Аналіз існуючих методів та програмних засобів	8.09.2025	
4	Проектування програмного забезпечення	26.09.2025	
5	Реалізація моделі виявлення ф розпізнавання текст	13.10.2025	
6	Розробка програмного забезпечення	24.10.2025	
7	Виконання експериментальних досліджень	15.11.2025	
8	Оформлення пояснювальної записки	26.11.2025	
9	Подання дисертації на попередній захист	26.11.2025	
10	Подання дисертації на захист	15.12.2025	

Студент Лариса ДИШКАНТ

Науковий керівник Юлія КРАМАР

РЕФЕРАТ

Розмір пояснювальної записки – 161 аркуш, містить 46 ілюстрацій, 24 таблиці, 6 додатків, 57 посилань на джерела.

Актуальність теми. У роботі розглянуто проблему зумовлена зростанням обсягів цифрової інформації, зокрема сканованих документів. Наявні програмні аналоги не забезпечують очікувано бажаний результат. Щоб отримати коректний текст, що буде відображати повністю оригінальний документ потрібно ефективні інструменти для його обробки та аналізу. Постійний прогрес в області машинного навчання та глибокого навчання відкриває нові можливості для підвищення точності та швидкості розпізнавання тексту. Технології OCR дозволяють автоматизувати рутинні завдання, такі як введення даних, архівування документів, пошук інформації, що підвищує продуктивність, знижує витрати та заощаджує час. Виявлено потребу в розробці програмного забезпечення виявлення та розпізнавання тексту в документах нетекстового формату шляхом застосування сучасних нейронних мереж.

Мета дослідження. Основною метою є покращити точність виявлення та розпізнавання тексту в документах нетекстового формату.

Об’єкт дослідження: програмне забезпечення розпізнавання тексту.

Предмет дослідження: метод, алгоритми та архітектура програмного забезпечення виявлення та розпізнавання тексту в документах нетекстового формату спрямовані на підвищення точності.

Для реалізації поставленої мети **сформульовані наступні завдання**:

— проаналізувати наявні рішення для визначення потрібних покращень точність виявлення та розпізнавання тексту в документах нетекстового формату;

— розробити метод із застосуванням сучасних нейронних мереж для виявлення та розпізнавання тексту в документах нетекстового формату;

— розробити програмне рішення;

— дослідити та оцінити ефективність запропонованого рішення.

Наукова новизна результатів магістерської дисертації полягає в тому, що запропоновано узагальнену модель програмної системи OCR, яка описує повний цикл обробки зображення, від попередньої підготовки до післяобробки результату, та адаптована до роботи з різнорідними типами документів. Удосконалено підхід до виявлення тексту в нетекстових документах шляхом поєднання традиційних методів попередньої обробки та застосування глибинних архітектур нейронних мереж, що забезпечує підвищення точності розпізнавання тексту, швидкодії та продуктивності праці.

Практичне значення отриманих результатів полягає в тому, що реалізовані та поєднані методи обробки зображення, використання двох OCR-двигунів і автоматичного failback та фреймворку PyQt5 в межах одного застосунку простого використання та нативного інтерфейсу користувача. Дана система є корисна для державних підприємств при роботі з сканованими документами та зображеннями, що дозволить оптимізувати рутинні завдання з введення даних, архівування та підвищить продуктивність і заощадить час.

Зв’язок з науковими програмами, планами, темами. Робота виконувалась на кафедрі інформатики та програмної інженерії Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського".

Апробація. Наукові положення дисертації пройшли апробацію на IX Міжнародній науково-практичній конференції молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології» (SoftTech-2025) – м. Київ.

Публікації. Наукові положення дисертації опубліковані в:

1) Дишкант Л.Л. Програмне забезпечення виявлення та розпізнавання тексту в документах не текстового формату / Дишкант Л.Л., Крамар Ю.М. // Матеріали

IX Міжнародної науково-практичної конференції молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології» (SoftTech-2025) – м. Київ: НТУУ «КПІ ім. Ігоря Сікорського», 26-28 листопада 2025 р.

Ключові слова: OCR, ВИЯВЛЕННЯ, РОЗПІЗНАВАННЯ ТЕКСТУ, CRNN, ОБРОБКА ЗОБРАЖЕННЯ, НЕЙРОННІ МЕРЕЖІ

ABSTRACT

Explanatory note size – 161 pages, contains 46 illustrations, 24 tables, 6 applications, 57 references.

Topicality. The paper addresses the problem caused by the growth in the volume of digital information, in particular scanned documents. Existing software analogues do not provide the expected desired result. To obtain a correct text that fully reflects the original document, effective tools for its processing and analysis are needed. Continuous progress in the field of machine learning and deep learning opens up new opportunities for improving the accuracy and speed of text recognition. OCR technologies allow you to automate routine tasks such as data entry, document archiving, and information search, which increases productivity, reduces costs, and saves time. There is a need to develop software for detecting and recognizing text in non-text documents using modern neural networks.

The aim of the study. The main target is to improve the accuracy of text detection and recognition in non-text documents.

The object of research: text recognition software.

The subject of research: methods, algorithms, and architecture of software for detecting and recognizing text in non-text documents aimed at improving accuracy.

To achieve this goal, the **following tasks** were formulated:

- analyze existing solutions to identify necessary improvements accuracy of text detection and recognition in non-text format documents;
- develop a method using modern neural networks to detect and recognize text in non-text format documents;
- develop a software solution;
- research and evaluate the effectiveness of the proposed solution.

The scientific novelty of the results of the master's dissertation is that a generalized model of the OCR software system is proposed, which describes the complete image processing cycle, from preliminary preparation to post-processing of the result, and is adapted to work with different types of documents. The approach to detecting text in non-text documents has been improved by combining traditional pre-processing methods and the use of deep neural network architectures, which increases text recognition accuracy, speed, and productivity.

The practical value of the obtained results lies in the fact that the implemented and combined image processing methods, the use of two OCR engines and automatic failback, and the PyQt5 framework are integrated within a single application with a simple user interface and native user interface. This system is useful for state-owned enterprises when working with scanned documents and images, as it will optimize routine data entry and archiving tasks, increase productivity, and save time.

Relationship with working with scientific programs, plans, topics. Work was performed at the Department of Computer Science and Software Engineering of the National Technical University of Ukraine «Igor Sikorsky Kyiv Polytechnic Institute».

Approbation. The scientific provisions of the dissertation were tested at the IX International Scientific and Practical Conference of Young Scientists and Students "Software Engineering and Advanced Information Technologies" (SoftTech-2025) - Kyiv.

Publications. The scientific provisions of the dissertation were published in:

1) Dyshkant L.L. Software for detecting and recognizing text in non-text format documents/ Dyshkant L.L., Kramar Y.M. // Proceedings of the IX International Scientific and Practical Conference of Young Scientists and Students "Software Engineering and Advanced Information Technologies" (SoftTech-2025) – Kyiv: Igor Sikorsky Kyiv Polytechnic Institute, November 26-28, 2025.

Keywords: OCR, DETECTION, TEXT RECOGNITION, CRNN, IMAGE PROCESSING, NEURAL NETWORKS

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

OCR — Optical Character Recognition — Оптичне розпізнавання символів (розпізнавання тексту);

NLP — Natural Language Processing — Обробка природної мови;

CNN — Convolutional Neural Network — Згорткова нейронна мережа;

RNN — Recurrent Neural Network — Рекурентна нейронна мережа;

CRNN — Convolutional Recurrent Neural Network — Згортково-рекурентна нейронна мережа;

LSTM — Long Short-Term Memory — Довга короткотривала пам’ять;

BiLSTM — Bidirectional LSTM — Двонаправлена LSTM;

CTC — Connectionist Temporal Classification — З’єднувальна тимчасова класифікація;

YOLO — You Only Look Once — алгоритм детекції об’єктів;

CLAHE — Contrast Limited Adaptive Histogram Equalization — Адаптивна еквалізація гістограми з обмеженням контрасту;

GUI — Graphical User Interface — Графічний інтерфейс користувача;

MVC — Model-View-Controller — Модель-Представлення-Контролер;

MVVM — Model-View-ViewModel — Модель-Представлення-Модель представлення;

CER — Character Error Rate — Помилка на рівні символів;

WER — Word Error Rate — Помилка на рівні слів;

PDF — Portable Document Format — Портативний формат документа;

GPU — Graphics Processing Unit — Графічний процесор;

HTR — Handwritten Text Recognition — Розпізнавання рукописного тексту;

SVM — Support Vector Machine — Метод опорних векторів.

ВСТУП

Сучасний світ генерує колосальні обсяги цифрової інформації, цифрових даних, значну частину якої становлять документи не текстового формату, а саме: скановані копії документів, фотографії текстів, знімки екранів тощо. За даними аналітичних агентств, щорічний приріст цифрової інформації зростає на 40–60%, при цьому левова частка цих даних залишається недоступною для автоматизованої обробки через відсутність текстового представлення, лише тому, що текст у них присутній виключно у вигляді зображення. В умовах цифрової трансформації бізнес–процесів та державного управління виникає нагальна потреба в ефективних інструментах автоматизованого перетворення документів не текстового формату у структурований текст, придатний для подальшої обробки, пошуку та аналізу. Наявні на ринку програмні рішення або мають обмеження у роботі з документами, або є комерційними продуктами з високою вартістю ліцензій, що обмежує їх доступність для малих підприємств та користувачів.

Автоматизація процесів розпізнавання тексту має значний практичний потенціал для різних сфер застосування: електронний документообіг, архівування та цифровізація бібліотечних фондів, автоматизована обробка бізнес–документів, аналіз історичних документів тощо. Впровадження ефективних рішень для розпізнавання тексту дозволяє підвищити продуктивність праці, знизити операційні витрати, а також забезпечити швидкий доступ до інформації та мінімізувати ризики втрат даних. Таким чином, розробка програмного забезпечення виявлення та розпізнавання тексту в документах не текстового формату з використання сучасних методів є актуальною науково–практичною задачею, яка має як теоретичне, так і прикладне значення.

Метою даного дослідження є покращити точність виявлення та розпізнавання тексту в документах не текстового формату.

Об’єктом дослідження — програмне забезпечення розпізнавання тексту.

Предметом дослідження — метод, алгоритми та архітектура програмного забезпечення виявлення та розпізнавання тексту в документах не текстового формату спрямовані на підвищення точності.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

— проаналізувати існуючі методи та програмні засоби виявлення та розпізнавання тексту, визначити їх переваги та недоліки застосування;

— дослідити сучасні архітектури нейронних мереж, придатні для задач детекції текстових областей та розпізнавання символів;

— розробити метод виявлення та розпізнавання тексту, що забезпечить підвищену точність порівняно з існуючими аналогами;

— реалізувати програмне забезпечення, що втілює запропонований метод, з урахуванням вимог до продуктивності, масштабованості та зручності використання;

— провести експериментальні дослідження та виконати порівняльний аналіз з існуючими рішеннями;

— оцінити ефективність запропонованого рішення за критеріями точності, швидкодії розпізнавання.

1 АНАЛІЗ МЕТОДІВ ТА ПРОГРАМНИХ ЗАСОБІВ ДЛЯ ВИЯВЛЕННЯ ТА РОЗПІЗНАВАННЯ ТЕКСТУ В ДОКУМЕНТАХ НЕ ТЕКСТОВОГО ФОРМАТУ

1.1 Опис предметної області

Розпізнавання тексту в документах не текстового формату є комплексною задачею, що включає декілька взаємопов'язаних етапів обробки інформації. До документів даного формату відносять скановані документи у форматах зображень: jpeg, png, tiff, фотографії документів, прінтскріни екрану, pdf-файли, а також відео з текстовим вмістом.

Ключовими аспектами предметної області є:

— тип вхідних документів, тобто система повинна обробляти різноманітні типи документів: офіційні документи (договора, посвідчення), бізнес–документи (рахунки, накладні, звіти), архівні матеріали, книги та журнали, рукописні записи, таблиці та форми, багатомовні документи. Кожен тип документів має свої структурні особливості та оформлення, а отже потенційні проблеми розпізнавання;

— якість даних, тобто якість зображення документів може значно варіюватися залежно від джерела. Наприклад якщо професійно відскановані документи мають високу роздільну здатність від 300 до 600 dpi, рівномірне освітлення та мінімальні спотворення, то фотографії документів зроблені смартфоном містять нерівномірне освітлення, часто блиски, розмиття. Також старі, архівні або пошкоджені документи можуть мати плями, різні потертості, вицвілий текст чи розриви. Ці всі фактори впливають на складність розпізнавання:

а) структура документів, тобто зазвичай документи мають різну структуру та компонування:

1) одно чи багатоколонковий текст;

2) таблиці;

3) поєднання тексту з графічними елементами;

4) примітки, колонтитули.

Правильне визначення структури документа критично важливо для коректного відтворення послідовного тексту;

б) мова, тобто система розпізнавання повинна підтримувати роботу з різними мовами та системами письма: латиниця, кирилиця та інші. Особливу складність становлять багатомовні документи, де в одному тексті поєднуються різні алфавіти, наприклад англійська та українська.

Завдання предметної області:

— локалізація областей зображення, що містять текст, відокремлення їх від графічних елементів, визначення меж текстових блоків, орієнтації тексту. Дана задача може бути особливо складною коли текст розташований під кутом чи документ має складний макет;

— сегментація тексту, поділ тексту на рядки, рядки на окремі слова чи символи, визначення ліній тексту. Якість сегментації сильно впливає на точність наступного розпізнавання;

— ідентифікація окремих символів або слів, визначення їх класу, тобто що саме це буква чи цифра чи спецсимвол, розпізнавання з урахуванням контексту;

— постобробка результатів, а це корекція помилок розпізнавання з використанням словників чи мовних моделей, форматування вихідного тексту, відновлення структури документа, перевірка орфографії.

Сфери застосування технологій розпізнавання тексту надзвичайно широкі. Електронний документообіг — автоматизоване введення паперових документів у систему, пошук по змісту сканованих документів, класифікація документів, а також

робота з сканованими документами. Обробка чеків чи розпізнавання платіжних документів у банківській сфері. Оцифрування медичних карт, обробка результатів аналізів чи розпізнавання рецептів в медицині. Робота з правовими документами, судовими рішеннями чи договорами в юридичній справі. Оцифрування навчальних матеріалів та створення електронних бібліотек в освіті.

Отже проаналізувавши предметну область, основними викликами будуть:

— різноманітність стилів та розмірів шрифтів, наявність декоративних елементів, рукописні шрифти;

— низька роздільна здатність, шум, розмиття, нерівномірне освтлення, проблеми якості зображення;

— геометричні спотворення, нахил;

— складні фони;

— одночасне розпізнавання різних мов та систем письма;

— математичні формули, нотації, спецсимволи.

Критично важливим для розробки ефективного рішення є розуміння специфіки предметної області, що буде враховувати всі вище перераховані фактори та забезпечувати високу точність розпізнавання в різноманітних умовах застосування.

1.2 Аналіз існуючих рішень

На ринку представлено значну кількість програмних рішень для розпізнавання тексту, які відрізняються за функціональністю, точністю, вартістю та підходами до обробки документів. Проаналізуємо основні категорії існуючих рішень та їх представників і окреслимо сильні та слабкі сторони.

1.2.1 ABBYY FineReader

Американська IT–компанія ABBYY є однією з найпопулярніших, яка розробила та вдосконалює програмний продукт FineReader (рисунок 1.1), що є одним з просунутих комерційних продуктів для розпізнавання тексту. Цей інструмент орієнтований насамперед на корпоративних користувачів та організації, що обробляють великі обсяги документів складної структури.

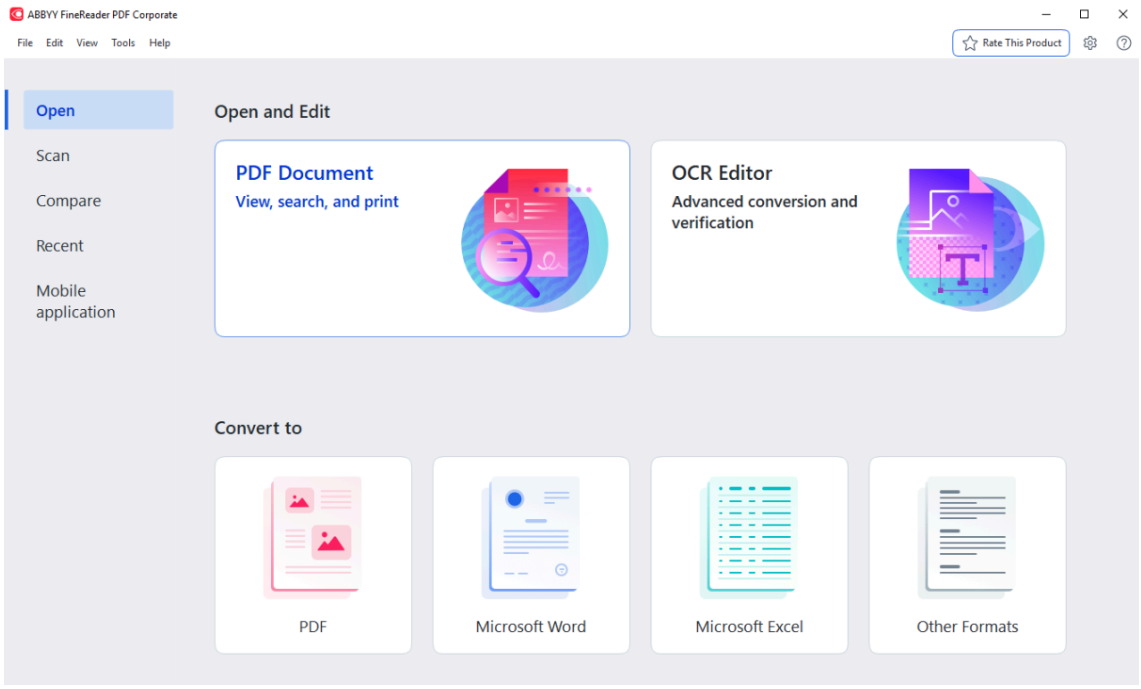


Рисунок 1.1 — Головна сторінка застосунку ABBYY FineReader

Основними можливостями ABBYY FineReader:

— розпізнавання тексту підтримуючи понад 190 мов розпізнавання;

— збереження оригінального форматування документів, включаючи макети з таблицями, колонками, списками;

— можливість пакетної обробки великих обсягів документів;

— інтеграція з Word, Excel, PDF, HTML (рисунок 1.2);

— автоматичне виправлення геометричних спотворень та корекція якості зображень;

— перевірка орфографії на основі вбудованих словників.

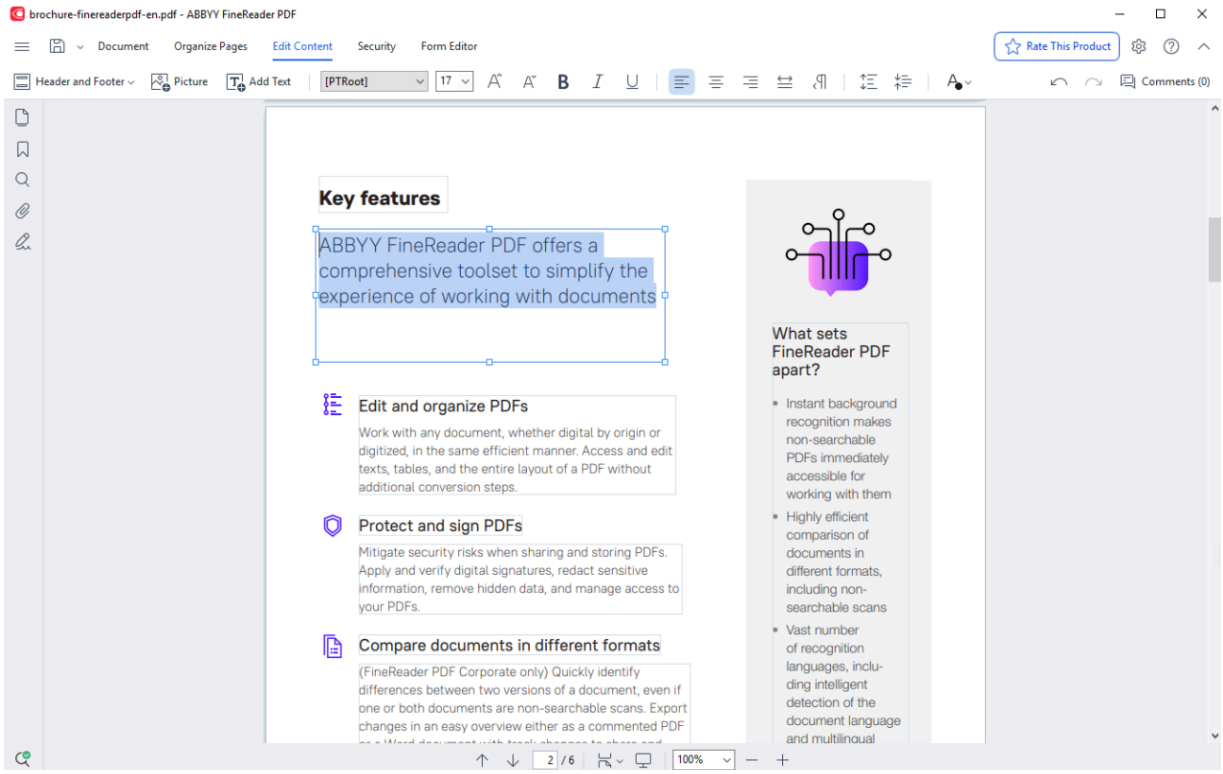


Рисунок 1.2 — Інтеграція з з Word, Excel можливістю редагування тексту

Переваги:

— висока точність розпізнавання навіть для документів складної структури;

— інтуїтивно зрозумілий інтерфейс користувача;

— підтримка багатомовності в документі;

— регулярні оновлення та технічна підтримка.

Слабкі сторони:

— висока вартість ліцензії, пробна безкоштовна версія надається на короткий термін в межах 7 днів для домашнього користування і місяць для комерційного на

один ПК. Після закінчення строку потрібна оплата для домашнього використання 1 користувача — 99 євро та 165 євро для одного ПК для бізнесу;

— суттєве споживання оперативної пам’яті та процесорного часу на складних документах, значні вимоги до обчислювальних ресурсів;

— відсутність можливості тонкого налаштування алгоритмів розпізнавання;

— закритий програмний код, що унеможливлює адаптацію під специфічні потреби;

— обмежені можливості автоматизації через API у базових версіях.

1.2.2 Adobe Acrobat

Adobe — американська компанія заснована Джоном Ворнок і Чарльзом Гешке в лютому 1982 року, щоб здійснити революцію в поліграфії та видавництві завдяки переходу на повністю цифру, як написано на їх сайті. Згідно зі статистикою, користувачі, які активно працюють з PDF–документами обирають Adobe Acrobat (рисунок 1.3). Даний застосунок має декілька варіантів: Adobe Reader, Standard, Pro, Pro Extended, відповідно і різний функціонал та можливості. Саме Adobe Acrobat Pro вже включає функціонал OCR. Основні характеристики:

— інтеграція OCR в робочий процес з PDF;

— автоматичне розпізнавання при відкритті сканованих документів;

— підтримка близько 30 мов;

— можливість редагування розпізнаного тексту безпосередньо в PDF.

Переваги:

— зручність для користувачів, які регулярно працюють з файлами формату PDF;

— збереження структури документів;

— інтеграція з хмарними сервісами Adobe;

— підтримка співпраці над документами в реальному часі.

Рисунок 1.3 — Головна сторінка застосунку Adobe Acrobat

Недоліки:

— високий щорічний платіж за підписку, на домашній ПК близько 200 доларів і 330 доларів на 5 ПК для бізнесу;

— нижча точність розпізнавання порівняно з спеціалізованими OCR—рішеннями;

— обмежена підтримка складних макетів та спецсимволів;

— відсутність гнучких налаштувань процесу розпізнавання;

— великий розмір програми та високі системні вимоги.

1.2.3 OmniPage

OmniPage від компанії Kofax ще одне потужне комерційне рішення, орієнтоване на корпоративний сегмент. Має два варіанта системи: OmniPage Standard та OmniPage Ultimate і відповідно з різним функціоналом. OmniPage Ultimate — OCR–рішення для бізнесу (рисунок 1.4), які прагнуть максимізувати продуктивність. Сумісний з Windows 11, 10, 8, 7 та XP(SP3), перетворює PDF–файли

на документи, які можна редагувати, ділитися або архівувати, а також підтримує практично будь–який сканер та маршрутизацію документів.

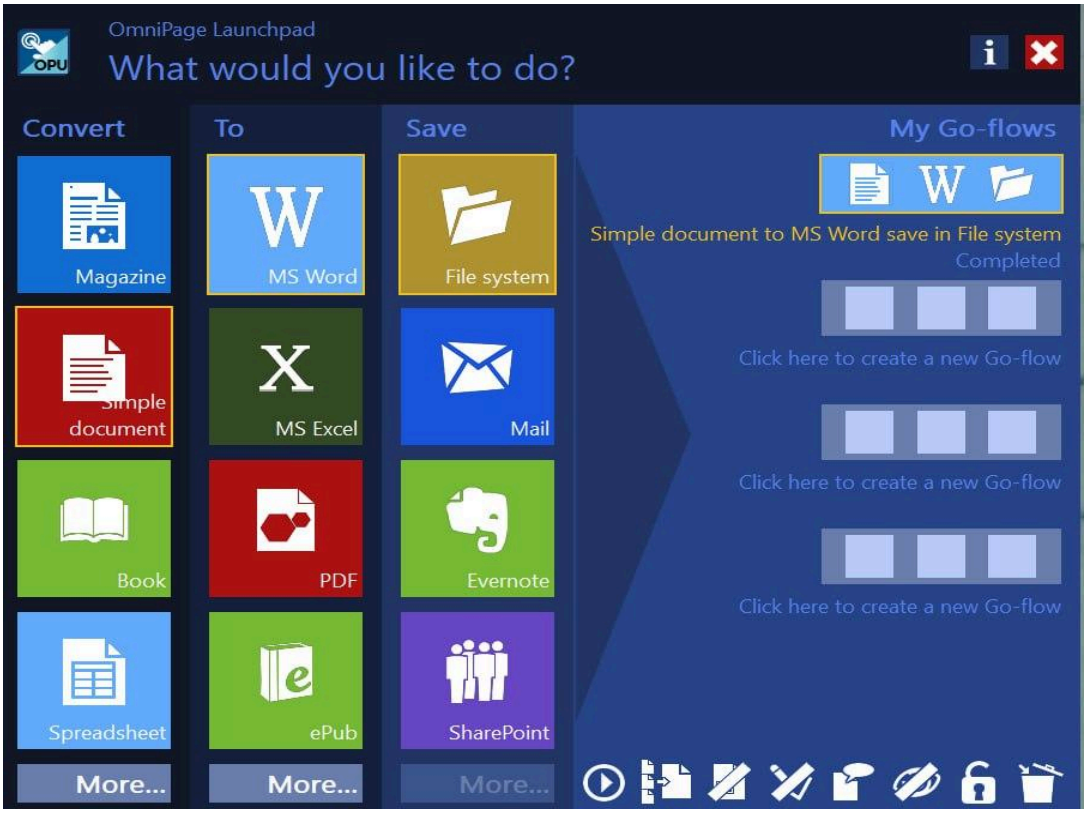


Рисунок 1.4 — OmniPage

Основні характеристики:

— автоматичне визначення структури документа;

— підтримка понад 120 мов;

— розпізнавання рукописного тексту;

— інтеграція з хмарними сховищами;

— працює з різними видами сканерів;

— пакетна обробка документів в реальному часі.

Переваги:

— висока швидкість обробки документів;

— хороша робота з формами та таблицями;

— можливість автоматизації через макроси;

— підтримка різних сканерів та джерел документів.

Недоліки:

— висока вартість корпоративних ліцензій, Standard коштує 149 доларів, а Ultimate вже 499 доларів для ПК;

— складність налаштування для нових користувачів;

— проблеми з розпізнаванням нестандартних шрифтів;

— працює тільки з Windows;

— вимоги для вільного місця на диску не менше 2.7 GB та наявності цифрової камери з автофокусом.

1.3 Аналіз існуючих методів

Методи розпізнавання тексту еволюціонували від простих шаблонних підходів до складних архітектур глибокого навчання. Щоб проаналізувати існуючі методи, які показали найкращі результати, визначимо дослідницькі питання (таблиця 1), відповідно яких проведемо дослідження та фільтрацію наукових статей.

Таблиця 1 — Визначення дослідницьких питань

RQ1	What are the most effective techniques and algorithms for detecting and recognizing text in images?
RQ2	What tools for extracting text from scanned documents and convert it into editable format in software engineering?
RQ3	What performance metrics are used to evaluate the effectiveness of softwa designed for text recognition in non–textual formats?
RQ4	How can OCR improve text recognition in documents?
RQ5	What are the challenges in NLP for text analysis?

Після відбору джерел згідно з сформульованими дослідницькими питаннями (таблиця 1), було проведено детальний аналіз та дослідження виявлених підходів для визначення оптимальної стратегії подальшої розробки.

Зокрема, у контексті роботи зі складними графічними об’єктами, вартим уваги є дослідження: “Text Localization in Scientific Figures using Fully Convolutional Neural Networks on Limited Training Data” [15], присвячене проблематиці наукових діаграм. Автори відходять від традиційних алгоритмів навчання без вчителя (unsupervised), які зазвичай застосовуються за умов дефіциту даних, та пропонують альтернативний метод локалізації та вилучення тексту з наукових діаграм за допомогою двокрокового підходу на базі повністю згорткових нейронних мережах (Fully Convolutional Networks). Ключовою особливістю цього підходу є інтеграція технік аугментації даних, що дозволяє компенсувати нестачу навчальної вибірки та забезпечити високу точність локалізації й вилучення блоків.

Технічна реалізація методу базується на удосконаленні архітектури, нова комбінація залишкової мережі та мережі пропозицій регіонів на основі Faster RCNN, що дозволяє значно точніше детектувати межі текстових блоків. Виділені фрагменти зображення передаються для обробки до OCR–ядра Tesseract 4.0 для розпізнавання тексту. Порівняльний аналіз, проведений на п’яти спеціалізованих датасетах, засвідчив перевагу такого підходу над кращими аналогами навчання без вчителя, демонструючи значний приріст ефективності на рівні 19 – 20%. Оцінка якості здійснювалася комплексно: для етапу локалізації застосовувалися метрики Average Precision і F1 score, а точність розпізнаного тексту перевіряли за допомогою Gestalt Pattern Matching і відстань Левенштейна. Важливим фактором успіху стало використання стратегії аугментації даних, що дозволило нівелювати проблему малої вибірки. Результати цього дослідження обґрунтовують доцільність інтеграції

детекторів на основі Faster RCNN у розроблювану систему, як ефективного інструменту для підвищення загальної точності розпізнавання.

Системний аналіз прогресу у сфері оптичного розпізнавання (OCR) та обробки рукописного тексту (HTR) наведена в роботі [20]. У ній досліджується еволюція методів глибокого навчання: від згорткових нейронних мереж (CNN), рекурентних нейронних мереж (RNN) до гібридних фреймворків CNN–RNN та архітектури на основі трансформерів. Емпіричні дані підтверджують високу ефективність цих підходів у задачах верифікації підписів (SV) та багатомовного аналізу. Сучасні моделі здатні досягати точності 99,98% при розпізнаванні цифр на різних наборах даних та понад 97% розпізнавання рукописних слів. Це стосується моделей, таких як гібридні CNN–RNN та архітектура трансформерів, які продемонстрували свою ефективність у захопленні просторових та часових ознак, а також у адаптації до варіабельності рукописного тексту в письмових та лінгвістичних контекстах.

Успіх таких рішень пояснюється здатністю гібридних CNN–RNN та трансформерних мереж одночасно враховувати просторові ознаки зображення та часові залежності послідовності символів. Водночас автори наголошують, що, попри високі показники на тестових вибірках, проблематика роботи з нелатинськими шрифтами та високою варіативністю індивідуальних почерків залишається актуально. Це окреслює подальший вектор досліджень, спрямований на підвищення робастності моделей та розробку методів їх тонкого налаштування (fine-tuning) для специфічних лінгвістичних контекстів.

У досліджені: “An English Handwriting Evaluation Algorithm Based on CNNs” [21] запропоновано алгоритм автоматизованого оцінювання каліграфічної якості англійського рукописного тексту, що є важливою частиною викладання базової англійської мови. Принциповою відмінністю цього методу від класичних підходів обробки документів є відмова від використання вручну згенерованих ознак на

користь автоматичного вилучення характеристик безпосередньо з піксельного масиву зображення. Експериментальна перевірка підтвердила перевагу даної архітектури над традиційними методами машинного навчання, зафіксувавши точність класифікації на рівні 94%. Слід зазначити, що хоча дана модель фокусується на аналізі візуальної якості написання, а не на розпізнаванні семантичного змісту, розглянуті в ній механізми вилучення графічних патернів за допомогою CNN є цінними для проектування архітектури нашого рішення.

Стаття “Handwritten Amharic Word Recognition With Additive Attention Mechanism" [22] розглядає складну проблему автоматичного розпізнавання рукописних амхарських слів. Амхарська мова є другою за поширеністю семітською мовою у світі, що робить її важливою для збору документів у паперовому вигляді та вимагає технології автоматичного розпізнавання. Автори підкреслюють труднощі, пов'язані з розпізнаванням рукописного амхарського тексту, такі як варіації в індивідуальному почерку, злиття слів без пробілів, схожість форм символів алфавіту та шум від відсканованих зображень. Також зазначається, що, незважаючи на значні досягнення в дослідженнях OCR, амхарському письму приділялося менше уваги. Для вирішення цієї проблеми, дослідження використовує підхід глибокого навчання з механізмом адитивної уваги до рекурентних нейронних мереж (RNN) для досягнення точного розпізнавання рукописних амхарських слів. Архітектура моделі розпізнавання складається з семи згорткових нейронних мереж (CNN) та двох RNN, що використовують стратегію з'єднувальної тимчасової класифікації, що забезпечує ефективне розпізнавання шляхом послідовного вилучення ознак. Дослідження також вирішило проблему недостатності даних для глибокого навчання, використовуючи техніки аугментації для збільшення наборів даних. Був використаний оригінальний набір даних з 12 047 рукописних амхарських слів та доповнений набір даних з 22 000 зображень. Розроблена модель досягла середньої посимвольної помилки (CER) 2,84% та середньої пословної помилки (WER) 9,75% для тестового набору даних.

Автори стверджують, що ці результати є багатообіцяючими і демонструють потенціал цього підходу на основі уваги для розпізнавання рукописних амхарських слів. Це дослідження є значним кроком до подолання розриву в технології OCR та демонструє трансформаційні можливості глибоко навчання у розпізнаванні образів. Аналіз цієї роботи дає чіткі орієнтири щодо вибору метрик якості та підтверджує доцільність використання механізмів уваги при побудові власних моделей розпізнавання неструктурованого тексту.

Проблематика структурного аналізу документів, що є фундаментальним етапом у проектуванні систем розпізнавання, детально висвітлена в роботі “Classification of Text regions in a Document Image by Analyzing the properties of Connected Components” [23]. Дослідження фокусується на ієрархічній класифікації сегментованих регіонів зображення.Основна концепція полягає у первинній бінарній сепарації контенту, тобто після сегментації зображення документа на різні регіони, ці регіони спочатку класифікуються як текст чи нетекст. Надалі застосовується глибока грануляція: нетекстові об’єкти потім додатково поділяються на підкласи (таблиці, зображення, роздільники, графіки, діаграми тощо), тоді як текстові регіони класифікуються як заголовки, абзаци, верхні/нижні колонтитули, підписи, буквиці тощо. Такий підхід дозволяє відтворити логічну структуру документа, а не лише його символьне наповнення.

Методологічною основою запропонованого рішення є аналіз зв'язних компонентів. Алгоритм ідентифікує тип регіону, спираючись на геометричні та топологічні ознаки: просторові координати, метричні характеристики та параметри вирівнювання BINYAS, де порівнювалася ефективність обробки з інтегрованим модулем класифікації та без нього. Результати експерименту доводять критичну важливість цього етапу для коректної інтерпретації складних макетів. Для нашого дослідження цей підхід є особливо цінним, оскільки він надає інструментарій для

детектування та збереження форматування нетекстових об’єктів, що гарантує конвертацію документів у вигляд максимально наближений до оригіналу.

Експоненціальне зростання обсягів мультимедійного контенту, генерованого сучасними мобільними платформами, актуалізує задачу автоматизованого вилучення семантичної інформації з доцільних зображень. Ця проблематика стала предметом глибокого аналізу в роботі "Comparative Analysis of Text Extraction from Color Images using Tesseract and OpenCV" [24]. Дослідження фокусується на побудові ефективного конвеєра обробки даних, який включає етапи детекції текстових блоків, їх локалізації, сегментації та фінального розпізнавання. Центральним елементом розглянутої архітектури є OCR–рушій Tesseract, який позиціонується як індустріальний стандарт завдяки високій швидкості та оптимізації. Проте, автори виокремлюють суттєве обмеження базової версії цього інструменту, що Tesseract добре працює з світлими фонами, але має труднощі з розпізнаванням тексту на темних відтінках або складних кольорових патернів призводить до критичного зниження якості розпізнавання. Для вирішення цієї проблеми запропоновано інтегрувати розширений етап попередньої обробки зображень, різні техніки preprocessing. Методологія включає застосування алгоритмів виявлення границь, адаптивну фільтрацію шумів та використання методів розмиття для зглажування артефактів. Окрім автоматизованих алгоритмів, у роботі представлено концепцію напівавтоматичного налаштування параметрів бінаризації через графічний інтерфейс, що надає оператору можливість калібрувати систему під специфічні умови освітлення. Порівняльний аналіз підтверджує, що запропонована комбінація методів OpenCV та Tesseract дозволяє досягти стабільного розпізнавання тексту на більшості типів фонів, що є критично важливим висновком для проектування системи оптичного розпізнавання тексту.

У роботі "A Novel Model for Recognising Handwritten Devanagari Numerals using Machine Learning" [25] представлено комплексний підхід до автоматизації розпізнавання рукописних цифр писемності, розпочинається з попередньої обробки даних, включаючи вилучення ознак та нормалізацію, після чого дані готуються для навчання та оцінки моделей. Автори зіставляють ефективність класичних методів моделей машинного навчання, від традиційних, таких як метод опорних векторів (SVM), до передових технік, таких як випадкові ліси (Random Forests), K-найближчі сусіди (K-Nearest Neighbors) та згорткові нейронні мережі (Convolutional Neural Networks), що забезпечує ретельну оцінку можливостей класифікації. Методи перехресної валідації застосовуються під час навчання та тестування моделей для підвищення надійності. Статистичні тести використовуються для оцінки варіацій продуктивності між моделями, що підвищує надійність аналізу. Візуалзація метрик продуктивності дозволяє чітко диференціювати сильні та слабкі сторони кожного підходу. Це дослідження має на меті ідентифікувати найбільш підходящу модель машинного навчання для класифікації рукописних цифр деванагарі, потенційно просуваючи системи розпізнавання символів та лінгвістичні програми. Дане дослідження описане, робить всебічний порівняльний аналіз, спрямований на розробку ефективних моделей машинного навчання для класифікації рукописних цифр, а також дає можливість побачити як різні класи алгоритмів справляються з задачами розпізнавання символів за умов варіативності почерку.

Рішення, що описане в статті [26] пропонує нову техніку для перетворення текстових даних, що містяться в зображеннях, в редагований формат. Зазначається, що в епоху соціальних мереж більшість даних зберігається у вигляді зображень, і їх правильна обробка може надати багато інформації. Зображення можуть бути отримані за допомогою камери, смартфона або безпосередньо з будь-якого джерела. Запропонована система працює наступним чином:

— застосовується аналіз зв'язних компонентів (Connected Components, CC) у поєднанні з реконструкцією країв на основі ширини штриха, що дозволяє точніше сегментувати символи перед передачею їх до оптичного модуля;

— для перетворення в редагований формат застосовується технологія оптичного розпізнавання символів (OCR) та Maximally Stable Extremal Region (MSER) для подальшої сегментації;

— запропонована система також може вилучати об'єкти з зображень за допомогою штучних нейронних мереж (ANN).

Але повне рішення розроблено з використанням MATLAB, а вихідні дані зберігаються у змінній. За потреби, їх також можна експортувати у документ Word для подальшого редагування. Продуктивність системи вимірюється за двома параметрами: точність (precision rate) та повнота (recall rate). Система демонструє близько 88% точності та 97% повноти, що, як стверджується, вище, ніж у більшості раніше запропонованих методів. Суттєвим недоліком запропонованого рішення є його реалізація в пропрієтарному середовищі MATLAB, що обмежує можливості вільного розгортання та інтеграції продукту. Його можна модернізувати чи на його основі реалізувати більш досконале.

В статті “Read Textual features in Images and convert to Editable form by extended use of Artificial Neural Networks, Deep learning and Maximally Stable Extremal Region” [27] продовжено роботу, що розглядається у попередній статті і зосереджена на вирішенні проблеми перетворення інформації, що зберігається в зображеннях, у редагований формат. Метою роботи є розробка зручного інструменту, який дозволяє витягувати текст з відео або розпізнавати чіткий рукописний текст у відсканованих документах і перетворювати його в редагований формат.

Запропонована методологія характеризується як надійна та здатна забезпечувати високу продуктивність навіть при спотвореннях макета, як описано. У поясненні використовуються такі технології для виявлення тексту, як оптичне розпізнавання символів (OCR), штучні нейронні мережі (ANN), глибоке навчання (Deep Learning) та регіони максимально стабільної екстремальної області (MSER). Вилучення тексту з відео може бути досягнуто за допомогою перетворення відео на кадри, а потім використання показника структурної схожості (Structural Similarity Index Measure, SSIM). Прототип розроблено з використанням MATLAB та надано з графічним інтерфейсом користувача (GUI), який може бути розгорнутий на робочій станції. Такий же висновок робимо, що дане рішення потребує доопрацювання чи розроблення нового на основі цього запропонованого.

Для оцінки моделі, якісно вона працює чи ні дослідимо статтю “Benchmarking NAS for article separation in historical newspapers" [11] Загалом дана стаття присвячена проблемі розділення статей у історичних газетах, що є ключовим етапом у оцифровуванні та забезпеченні доступності культурної спадщини. Розділення статей полягає у виявленні та вилученні окремих статей із відсканованих зображень газет та відновленні їхньої семантичної структури. Це критично важливо для того, щоб історичні газети стали машиночительними та пошуковими, що дозволить виконувати такі завдання, як вилучення інформації, узагальнення документів та текстовий аналіз.

Автори оцінюють (assess) набір даних NewsEye Article Separation (NAS), який є багатомовним набором даних для розділення статей в історичних газетах. Він складається з відсканованих газетних сторінок 19-го та 20-го століть та файлів анотацій німецькою, фінською та французькою мовами. Крім того, набір даних є складним через різноманітність макетів та стилів шрифтів, що ускладнює узагальнення моделей для невидимих даних.

І самим корисним в даній статті є те, що автори також вводять нові метрики (article error rate, article coverage score, proper predicted article та segmentation) для оцінки продуктивності моделей, навчених на NAS, щоб підкреслити актуальність та виклики цього набору даних. Вони вважають, що NAS, який є загальнодоступним, буде цінним ресурсом для дослідників, що працюють над оцифруванням історичних газет. Опрацювавши статтю, дізналися про ще нові метрики оцінки моделі.

1.3.1 Класичні методи розпізнавання символів

Шаблонне зіставлення є одним з найстаріших методів розпізнавання символів, який базується на порівнянні невідомого символу з набором попередньо завантажених шаблонів еталонних символів. Принцип роботи в тому що для кожного символу в базі зберігається один або декілька еталонних зображень. При розпізнаванні обчислюється міра схожості (кореляція або відстань) між вхідним символом та кожним шаблоном. Символ класифікується як той, з яким досягнуто найбільшу схожість.

Перевагами даного методу є:

— простота реалізації та розуміння;

— швидка робота при невеликій кількості шаблонів;

— не потребує навчання;

— добре працює для стандартизованих шрифтів.

Недоліки:

— необхідність великої кількості шаблонів для покриття варіацій;

— погана масштабованість;

— неефективна для рукописних текстів.

Приховані Марковські моделі стали популярними для розпізнавання послідовності символів, особливо рукописного тексту. Принцип роботи полягає в тому, що текст розглядається як послідовність станів (символів), де кожен стан генерує спостереження, а саме ознаки зображення. Модель вчиться ймовірностям переходів між станами та ймовірностям спостережень, що дозволяє знаходити найбільш ймовірну послідовність символів для даного зображення.

Переваги даної моделі:

— здатність моделювати часові або просторові залежності;

— можливість використання контексту для покращення розпізнавання;

— ефективні алгоритми навчання та декодування.

Недоліки:

— складність моделювання довгих залежностей;

— необхідність попередньої сегментації чи витягування ознак;

— обмежена здатність до узагальнення на нові шрифти.

1.3.2 Методи на основі нейронних мереж

Нейронна мережа — комп’ютерна система, яка намагається працювати подібно до людського мозку. Велика кількість простих нейронів з’єднаних між собою. Кожен такий нейрон отримує інформацію від інших, обробляє її та передає далі.

Історично першими для задач класифікації символів застосували повнозв’язні нейронні мережі прямого поширення, багатошарові персептрони. Навчання таких

моделей базується на алгоритмі зворотнього поширення помилки. Мережа складається з кількох шарів нейронів, з’єднаних послідовно:

а) вхідний шар — приймає початкові дані, наприклад характеристики об’єкта. Це просто точки входу, без обчислень;

б) приховані шари — один або декілька шарів між входом та виходом. В яких відбувається обробка. Кожен нейрон з’єднаний з усіма нейронами попереднього шару;

в) вихідний шар — видає фінальний результат.

Так як кожен нейрон одного шару з’єднаний з кожним нейроном наступного шару то шари називають повнозв’язними. Працює відповідно дані рухаються від входу до виходу, тому дана мережа — мережа прямого поширення. А навчається використовуючи алгоритм зворотного поширення помилки, а саме:

— мережа робить прогноз;

— обчислюється помилка;

— помилка йде назад через шари;

— ваги коригуються, щоб зменшити помилку.

Відповідно для задачі розпізнавання, вхідний шар отримує пікселі зображення, часто після нормалізації розміру, один чи декілька прихованих шарів з нелінійними функціями активаціями, вихідний шар з нейронами для кожного класу символів.

Переваги:

— автоматичне навчання ознак без ручного проектування;

— здатність моделювати нелінійні залежності;

— універсальна апроксимуюча здатність;

— швидкість роботи після навчання.

Недоліки:

— велика кількість параметрів;

— відсутність урахування просторової структури зображення;

—схильність до перенавчання без регуляризації;

— потреба у великих обсягах даних для навчання;

— нормалізація вхідних зображень до фіксованого розміру.

CRNN (Convolutional Recurrent Neural Network) — дана модель має гібридну архітектуру, що поєднує переваги CNN та RNN для наскрізного розпізнавання послідовностей тексту. Мережа складається з трьох основних частин, які працюють послідовно:

а) CNN — згорткові шари, витягують візуальні ознаки з зображення. Працюють як очі мережі, які знаходять форми, текстури. Перетворюють картинку на набір ознак і добре розпізнають локальні паттерни;

б) RNN — рекурентні шари, зазвичай використовують LSTM чи GRU, аналізують послідовність ознак зліва направо. Добре розуміють контекст — що одна буква впливає на наступну. Запам’ятовують інформацію з попередніх кроків;

в) Transcription layer — шар транскрипції, перетворює вихід рекурентних шарів у текст, часто використовує Connectionist Temporal Classification та вирішує проблему вирівнювання — де саме на картинці який символ.

В результаті CNN добре бачить що, але не розуміє порядок, RNN розуміє послідовність, але погано працює безпосередньо з зображеннями, а разом отримуємо гарне рішення основним застосуванням якого є розпізнавання тексту, мови з спектрограм та завдання де є зображення плюс послідовність.

Але є недоліки:

— складність навчання через комбінацію різних типів шарів;

— вимоги до обчислювальних ресурсів;

— необхідність великих наборів даних;

— обмеження у моделюванні залежностей між вихідними символами.

Transformer архітектура, трансформерні моделі — сучасний етап розвитку та підхід до розпізнавання тексту, що пов’язаний з адаптацією архітектури та замінює рекурентні мережі механізмом уваги (attention), що дає кращі результати та швидкість. На початку розроблена для задач обробки природної мови та знайшла застосування і в комп’ютерному зорі, включаючи OCR. Основні компоненти:

— Multi–head self–attention, дозволяє моделювати залежності між всіма позиціями в послідовності;

— Position encoding, додає інформацію про позицію елементів;

— Feed–forward мережі, обробляють представлення на кожній позиції;

— Layer normalization residual та connections, стабілізують навчання глибоких моделей.

Переваги трансформерів в OCR:

— здатність моделювати глобальні залежності без рекурсії;

— ефективна паралелізація обчислень;

— можливість трансформувати навчання з великих попередньонавчених моделей.

Недоліки:

— дуже великі вимоги до даних та обчислювальних ресурсів;

— складність архітектури та налаштування;

— квадратична складність по відношенню до довжини послідовності;

— необхідність значних ресурсів для навчання з нуля.

1.3.3 Методи попередньої та постобробки

Попередня обробка зображень:

— бінаризація, перетворення у чорно–біле зображення (методи Otsu, адаптивна бінаризація Sauvola);

— корекція нахилу, виявлення та виправлення повороту сторінки;

— видалення шуму, фільтри (медіанний, Гаусів), морфологічні операції;

— корекція освітлення, вирівнювання гістограми, CLAHE (Contrast Limited Adaptive Histogram Equalization);

— підвищення різкості, оператори Лапласа, unsharp masking;

— корекція перспективних спотворень, використання перетворень гомографії.

Постобробка результатів розпізнавання:

— перевірка орфографії, використання словників для виявлення та корекції помилок;

— мовні моделі, n–грамні моделі або нейронні мовні моделі для оцінки ймовірності послідовностей слів;

— регулярні вирази, для виявлення та валідації специфічних патернів (дати, номери телефонів, email);

— контекстна корекція, використання розуміння структури документа для виправлення помилок;

— об'єднання результатів, ансамблі моделей для підвищення надійності.

1.4 Постановка задачі

На основі проведеного аналізу можна сформувати наступну постановку задачі дослідження.

Мета роботи полягає в розробці програмного забезпечення виявлення та розпізнавання тексту в документах не текстового формату, що забезпечить підвищену точність шляхом застосування архітектур нейронних мереж. Щоб досягти цієї мети поставлено наступні задачі:

а) розробити метод попередньої обробки зображень документів, що адаптивно коригує якість зображення від його характеристик: освітлення, контрастність, наявність шуму, геометричні спотворення;

б) дослідити та створити для оновлення версії систему виявлення текстових областей на основі сучасних архітектур детекції об’єктів, таких як YOLOv5/v8 або подібних;

в) розробити модель розпізнавання послідовностей символів на основі гібридної архітектури, що поєднує згорткові мережі для виявлення ознак, рекурертні мережі для моделювання послідовностей та механізму уваги для покращення точності;

г) створити систему постобробки результатів розпізнавання;

д) реалізувати програмне забезпечення з модульною архітектурою, що забезпечує:

1) підтримку різних форматів вхідних файлів: JPEG, PNG, TIFF, PDF;
2) експорт результатів у різні текстові формати;
3) можливість інтеграції з різними системами;
4) пакетну обробку великих обсягів документів;
5) можливість налаштувати параметри розпізнавання.

е) провести експериментальні дослідження розробленої системи на стандартних наборах даних та порівняти результат з існуючими рішеннями за метриками:

1) швидкість обробки (документів/секунду);
2) Character Error Rate (CER);
3) Word Error Rate (WER);

4) Точністю (Precision);

5) F1- мірою.

є) оцінити ефективність рішення та можливість практичного застосування.

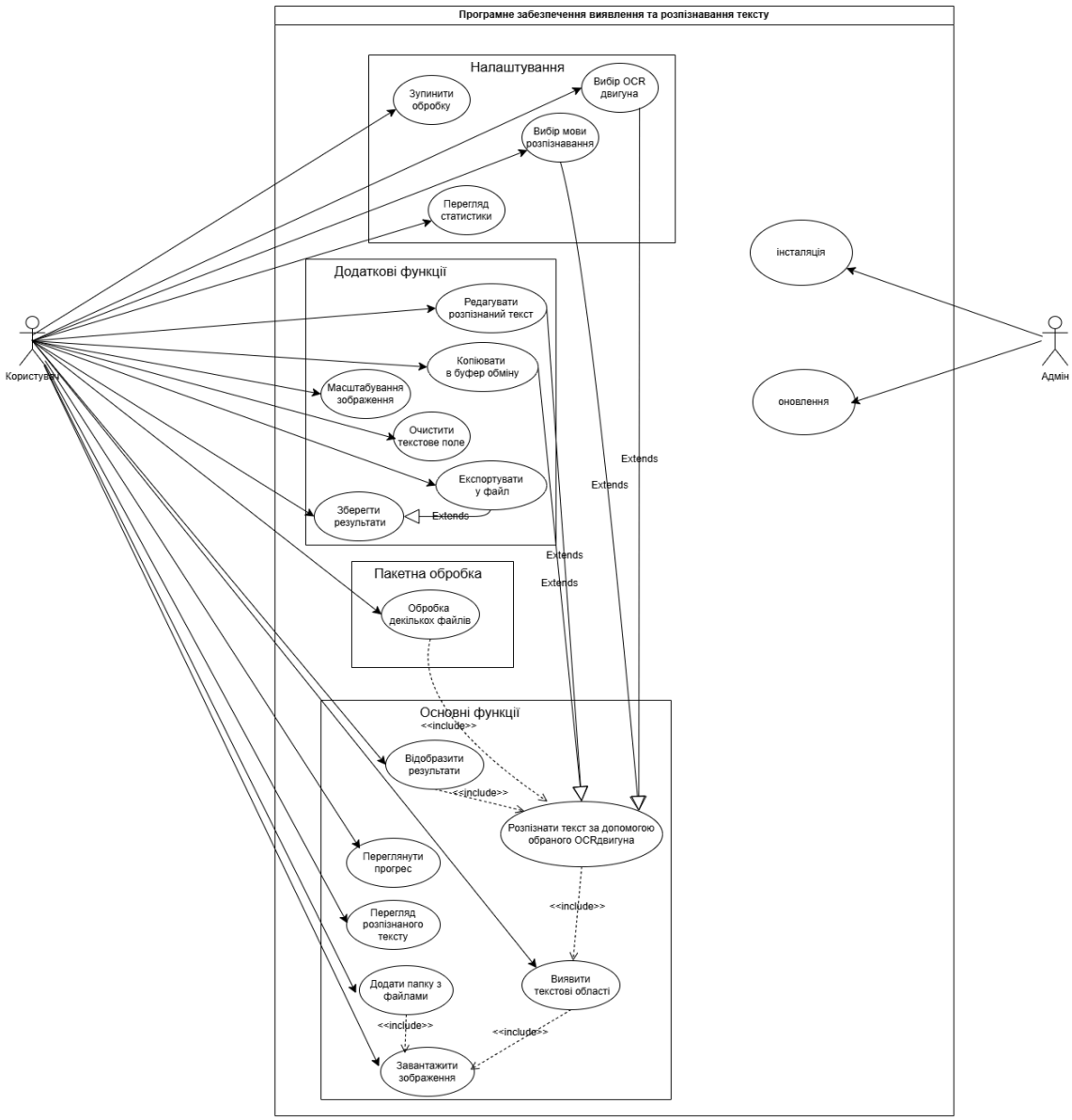


Рисунок 1.5 — Діаграма прецедентів

Діаграма прецедентів (рисунок 1.5) ілюструє взаємозв’язки між акторами системи та відповідними їм прецедентами, формалізуючи функціональні вимоги до програмного забезпечення. Вона систематизує операційні аспекти, асоційовані з кожною роллю користувача, чітко розмежовуючи прецеденти, забезпечуючи структуроване представлення поведінки системи в контексті взаємодії з зовнішніми суб’єктами.

Користувач повинен мати можливість:

- завантажити зображення;
- завантажити декілька файлів, для пакетної обробки;
- вибрати мову розпізнавання;
- вибирати OCR– рушій;
- переглянути відображення оригіналу документа;
- відобразити результат розпізнавання;
- редагувати розпізнаний текст;
- копіювати в буфер обміну текст;
- очистити результат завантаження та результат розпізнавання;
- збільшувати/ зменшувати зображення оригіналу документа;
- зберегти результат розпізнавання/ розпізнавання та редагування;
- експортувати у файл текстового формату;
- переглянути статистику, яка включає дані про OCR–рушій, назву файлу та оцінку точності розпізнавання;
- зупинити розпізнавання.

Адміна задачі:

- інсталяція та налаштування;
- оновлення застосунка.

Висновки до розділу

В розділі розглянуто та проаналізовано предметну область для задачі виявлення та розпізнавання тексті в документах не текстового формату. Зокрема визначено ключові аспекти, такі як різноманітність типів документів, вплив якості зображень, структурні особливості та мовні виклики, а також основні завдання обробки: локалізація, сегментація, ідентифікація символів та постобробка. Окреслено сфери застосування технологій OCR та основні проблеми, що впливають на ефективність розпізнавання, включаючи варіації шрифтів, шум, спотворення та багатомовність. Проведено аналіз існуючих комерційних рішень, таких як ABBYY FineReader, Adobe Acrobat та OmniPage, з оцінкою їх функціональності, переваг: висока точність, підтримка багатомовності, інтеграція з іншими інструментами та недоліків: висока вартість, системні вимоги, обмеження гнучкості. Дані рішення демонструють ефективність для стандартних завдань, але потребують адаптації для специфічних умов.

Досліджено та проаналізовано наукові статті, отримано розуміння та виокремлено сучасні методи виявлення і розпізнавання тексту, особливо рукописного. Даний аналіз показав ефективність глибокого навчання для локалізації тексту, обробки складних документів, а також відповідні метрики для оцінки ефективності. На основі аналізу сформовано постановку задачі: розробку програмного забезпечення з адаптивною попередньою обробкою, гібридною моделлю розпізнавання та модульною архітектурою для підтримки різних форматів і пакетної обробки. Експериментальна оцінка передбачає порівняння з існуючими рішеннями за ключовими метриками, що дозволить підтвердити ефективність та практичне застосування запропонованого підходу. Загалом, проведений аналіз свідчить про необхідність інтеграції сучасних нейромережевих методів для подолання викликів предметної області та створення універсального OCR-рішення.

2 МЕТОДИ ТА ЗАСОБИ ВИЯВЛЕННЯ ТА РОЗПІЗНАВАННЯ ТЕКСТУ

2.1 Вибір архітектури системи та OCR–технологій

2.1.1 Обґрунтування вибору OCR–технологій

На основі проведеного аналізу було обрано комбінований підхід, що використовує декілька OCR–рушіїв для забезпечення максимальної точності та надійності виявлення та розпізнавання. А також розглянуто можливість масштабування та інтеграції гібридної моделі, яка попередньо навчається на даних відповідної специфіки підприємства чи бізнес сфери.

Критерії вибору OCR–технологій:

— точність розпізнавання;

— швидкодія;

— підтримка мов;

— доступність, тобто відкритий вихідний код чи безкоштовна ліцензія;

— можливість інтеграції;

— якість документації.

Основний рушій розпізнавання — Tesseract 4.0. Tesseract OCR, розроблений компанією HP та підтримуваний Google, є одним з найпопулярніших рушіїв, ядром з відкритим кодом. Версія 4.0 показала революційні зміни у порівнянні з попередніми версіями. Гібридна архітектура використовує комбінацію традиційного та нейронного підходів, а режим OEM (OCR Engine Mode) визначає, який підхід використовувати:

— OEM 0, тільки використання Legacy engine;

— OEM 1,тільки Neural Net (LSTM);

— OEM 2, Legacy та LSTM;

— OEM 3, автоматичний вибір оптимального режиму.

LSTM–мережі, дозволяють врахувати контекст при розпізнаванні символів, що значно підвищує точність для з'єднаних літер та складних шрифтів. А використання Bidirectional LSTM — враховувати контекст як зліва, так і справа від розпізнаваного символу.

Переваги для даного проекту:

— висока точність на друкованому тексті;

— відмінна підтримка української мови;

— можливість точного налаштування;

— стабільність та надійність;

— великий досвід використання у промислових системах.

Обмеження:

— нижча швидкість порівняно з деякими сучасними рішеннями;

— потребує якісної попередньої обробки для складних документів;

— обмежена підтримка GPU прискорення.

PaddleOCR — додатковий рушій розпізнавання

PaddleOCR, який є продуктом компанії Baidu на базі фреймворку PaddlePaddle, являється сучасним рішенням, оптимізованим для високої швидкості та точності. Має модульну структуру, яка складається з: Text Detection, Direction Classifier, Text Recognition. Детектор тексту використовує сучасний метод сегментації, що працює швидше та зберігає високу точність. Розпізнавач з своєю архітектурою забезпечує ефективне розпізнавання послідовностей змінної довжини, завдяки CRNN з CTC декодером.

Переваги для проекту:

— висока швидкість обробки;

— вбудована детекція текстових областей;

— відмінні результати на зображеннях з нестандартним розташуванням тексту;

— підтримка повернутого та деформованого тексту;

— оптимізація для роботи без GPU.

Обмеження:

— нижча точність на дуже дрібному тексті;

— менша кількість підтримуваних мов в порівнянні з Tesseract;

— відносно нова технологія з меншою базою досвіду використання.

На основі отриманих результатів дослідження та мети створено гібридну модель в основі якої архітектура, яка поєднує обидві моделі беручи від них основні задачі (для інтеграції після навчання в оновлену версію застосунка):

CRNN — витягує візуальні ознаки;

BiLSTM — моделює послідовності;

Attention — фокусування на важливих частинах.

Переваги над Tesseract та PaddleOCR, що дана модель поєднала обидві архітектури та в результаті отримаємо, що CRNN вчить послідовність, а дана модель вчить, на що дивитися. Після навчання повинна давати максимальний результат розпізнавання тексту на складних документах, а також на специфічної структури.

2.1.2 Стратегія інтеграції OCR-рушіїв

Для максимізації переваг обох технологій було розроблено гібридну стратегію з механізмом автоматичного перемикання (fallback).

Принцип роботи гібридної системи:

- Пріоритетний вибір, тобто користувач може обрати основний рушій через інтерфейс.
- Автоматичний fallback — коли виникає помилка основного рушія відбувається автоматичне перемикання на резервний.
- Аналіз впевненості, що означає порівняння confidence scores для вибору кращого результату.
- Специфічні випадки, тобто використання PaddleOCR для складних макетів, Tesseract для стандартних документів.

2.1.3 PyQt5 для розробки GUI

Для створення графічного інтерфейсу користувача обрано фреймворк PyQt5 — один з найпотужніших інструментів для розробки десктопних застосунків на Python.

Обгрунтуємо вибір саме PyQt5:

Технічні переваги:

— кросплатформеність, тобто підтримка Windows, Linux, macOS;

— багатий набір готових компонент для всіх типів інтерфейсів;

— елегантна система подій для багатопотокових застосунків — Signals and Slots;

— QThread, вбудована підтримка багатопотоковості без блокування GUI;

— сучасний вигляд, підтримка стилів та тем оформлення.

Функціональні можливості для даного проекту:

— QMainWindow, основне вікно з меню та панелями інструментів;

— QFileDialog, діалоги вибору файлів;

— QProgressBar, індикація прогресу обробки;

— QTextEdit, редагування та відображення результатів;

— QTabWidget, відображення списку файлів;

— QLabel та QPixmap, відображення зображень.

PyQt5 використовує модифіковану MVC–архітектуру для організації коду:

— Модель: класи OCREngine, ImageProcessor, ResultManager — бізнес–логіка;

—View, тобто представлення — відображення даних, віджети PyQt5;

— Controller: MainWindow —обробка подій та координація.

Багатопотоковість через QThread, що є ключовою особливістю PyQt5 для даного проекту — можливість виконувати тривалі операції у окремих потоках (рисунок 2.1).

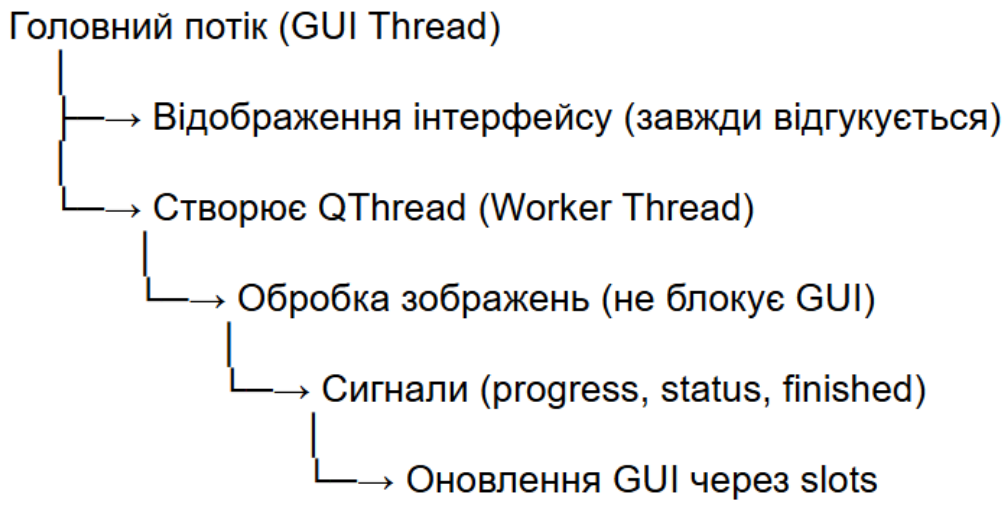


Рисунок 2.1 — Діаграма архітектури Signal/Slots в PyQt5

2.2 Математичне обгрунтування методів розпізнавання тексту

2.2.1 Архітектура LSTM для розпізнавання послідовностей

LSTM – комірка складається з трьох вентилів (gates) та стану комірки, що дозволяє зберігати інформацію на тривалий час.

Вентиль забування, визначає яку інформацію з попереднього стану слід забути:

$$f_t = \sigma(W_f \ [h_{t-1}, x_t] + b_f), \quad (2.1)$$

де:

$f_t \in [0,1]$ — вектор забування;

σ — сигмоїдна функція активації;

W_f — вагова матриця;

h_{t-1} — попередній прихований стан;

x_t — поточний вхід;

b_f — зміщення

Вентиль входу, визначає яку нову інформацію додати до стану:

$$i_t = \sigma(W_i \ [h_{t-1}, x_t] + b_i), \quad (2.2)$$

$$\overline{C}_t = tanh(W_C \ [h_{t-1}, x_t] + b_C), \quad (2.3)$$

де:

i_t — вектор входу;

$\overline{C}_t$ — кандидат на оновлення стану.

Оновлення стану комірки:

$$C_t = f_t \odot C_{t-1} + i_t \odot \overline{C}_t, \tag{2.4}$$

де:

$\odot$ — поелементне множення;

C_t — новий стан комірки.

Вентиль виходу:

$$o_t = \sigma(W_o \quad [h_{t-1}, x_t] + b_o), \tag{2.5}$$

$$h_t = o_t \odot \tanh(C_t), \tag{2.6}$$

Для врахування контексту з обох сторін використовують двонаправлені LSTM:

$$\rightarrow h_t = LSTM_{forward}(x_1, x_2, \ldots, x_t),$$

$$\leftarrow h_t = LSTM_{forward}(x_T, x_{T-1}, \ldots, x_t)$$

$$h_t = [\rightarrow h_t; \leftarrow h_t],$$

де:

$\rightarrow h_t$ — прихований стан прямого напрямку;

$\leftarrow h_t$ — прихований стан зворотного напрямку;

[;] — операція конкатенації.

У Tesseract 4.0 архітектура LSTM використовується для розпізнавання символів.

2.2.2 Архітектура CRNN в PaddleOCR

CRNN поєднує переваги згорткових мереж, а саме вилучення просторових ознак та RNN моделювання послідовностей. Схема структури CRNN на рисунку 2.2 демонструє архітектуру глибокого навчання, яка використовується для оптичного розпізнавання символів у зображеннях та для розпізнавання тексту в нерівних послідовностях.

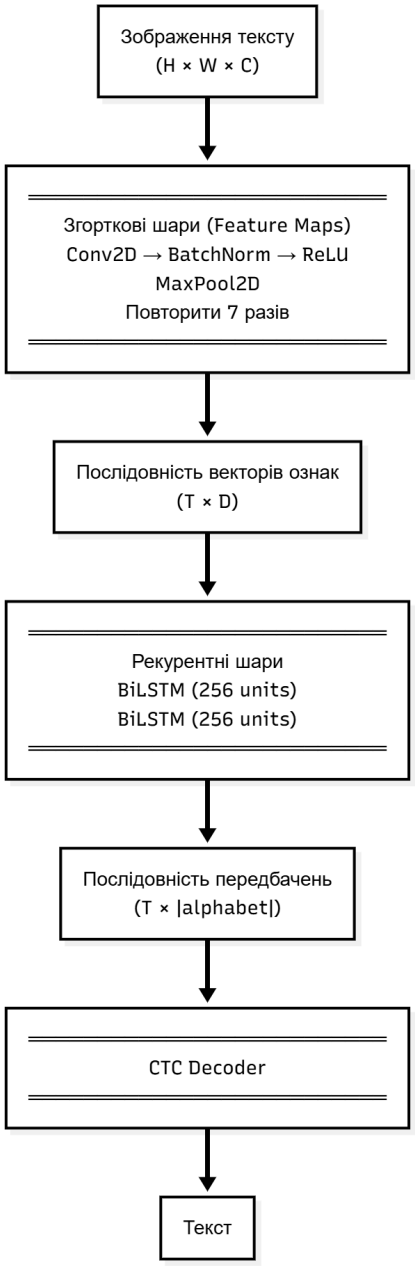


Рисунок 2.2 — Схема CRNN

Згорткові шари Feature Maps, даний блок виконує вилучення просторових ознак із вхідного зображення тексту. На вхід подається зображення тексту з розмірами: висоти, ширини та кількості каналів (1 для чорно-білого, 3 для

кольорового). Шар операцій повторюється 7 разів і містить стандартні компоненти згорткової мережі:

— Conv2D, який виконує згортку для виявлення ознак: ліній, кутів, частин символів;

— BatchNorm, який нормалізує виходи шарів і це прискорює навчання та підвищує стабільність;

— ReLU це функція активації, що додає нелінійності;

— MaxPool2D, який зменшує розмірність просторових даних, зберігаючи найважливіші ознаки та роблячи модель стійкою до невеликих зсувів.

— на вихід надходить послідовність векторів ознак.

Згорткові шари перетворюють 2D-зображення на 1D-послідовність.

Рекурентні шари аналізують послідовності та роблять прогноз на основі витягнутих ознак. На вхід приходить послідовність векторів ознак, які йдуть на блок операцій, що використовують два шари BiLSTM по 256 одиниць кожен. Завдяки двонаправленій структурі мережа обробляє послідовність у двох напрямках, що дає враховувати попередній та наступний контекст під час передбачення символу. На вихід надходить послідовність передбачень. Для кожного часового кроку T, мережа генерує вектор ймовірностей, де |alphabet| є розмір словника, включаючи спеціальний символ “blank”.

CTC Decoder перетворює послідовність ймовірностей на фінальний текстовий рядок.

2.2.3 Метрики якості розпізнавання

Щоб оцінити якість та точність отриманих результатів обов’язково потрібні метрики. Однак оцінка якості зазвичай не зводиться до єдиної простої формули, через те що якість залежить від багатьох факторів: збереження тексту, форматування, структури документа.

Character Error Rate (CER) — відсоток помилок на рівні символів:

$$\mathrm{CER} = \frac{S + D + I}{N}, \tag{2.7}$$

де:

S — кількість підстановок;

D — кількість видалень;

I — кількість вставок;

N — загальна кількість символів у еталоні документа.

Word Error Rate (WER) — відсоток помилок на рівні слів:

$$\mathrm{WER} = \frac{S_w + D_w + I_w}{N_w}, \tag{2.8}$$

де:

індекс w означає операції на рівні слів.

Відстань Левенштейна — мінімальна кількість операцій редагування для перетворення одного рядка в інший.

Confidence Score — середня впевненість розпізнавання.

2.3 Алгоритми та методи попередньої обробки зображень

2.3.1 Корекція геометричних спотворень

Корекція геометричних спотворень включає:

— детекцію та виправлення нахилу (Deskew). Математична модель буде: Нехай I(x,y) — зображення документа з кутом нахилу θ. Задача — знайти θ та застосувати поворот. Реалізація у проекті клас ImageProcessor метод deskew (рисунок 2.3).

```python
def deskew(self, image: np.ndarray) -> np.ndarray:
    """
    Виправлення нахилу зображення

    Args:
        image: Вхідне зображення

    Returns:
        Виправлене зображення
    """
    try:
        # Конвертація в відтінки сірого
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        # Інверсія кольорів
        gray = cv2.bitwise_not(gray)

        # Бінаризація
        thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]

        # Знаходження координат всіх ненульових пікселів
        coords = np.column_stack(np.where(thresh > 0))

        # Обчислення кута нахилу
        angle = cv2.minAreaRect(coords)[-1]

        # Корекція кута
        if angle < -45:
            angle = -(90 + angle)
        else:
            angle = -angle

        # Виправлення нахилу тільки якщо кут значний
        if abs(angle) > 0.5:
            (h, w) = image.shape[:2]
            center = (w // 2, h // 2)
            M = cv2.getRotationMatrix2D(center, angle, 1.0)
            image = cv2.warpAffine(
                image, M, (w, h),
                flags=cv2.INTER_CUBIC,
                borderMode=cv2.BORDER_REPLICATE
            )
            logger.debug(f"Виправлено нахил на {angle:.2f} градусів")

        return image
```

Рисунок 2.3 — Реалізація Deskew

— матриця афінного перетворення, поворот навколо центру (c_x, c_y) на кутθ:

$$M = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & t_x \\ \sin(\theta) & \cos(\theta) & t_y \end{bmatrix}, \tag{2.9}$$

де:

$$t_x = c_x - c_x \cos(\theta) + c_y \sin(\theta), \tag{2.10}$$

$$t_x = c_x - c_x \cos(\theta) + c_y \sin(\theta), \tag{2.11}$$

— корекція перспективних спотворень.

Реалізація у проекті на рисунку 2.4.

```python
def detect_and_correct_perspective(self, image: np.ndarray) -> np.ndarray:
    """
    Виявлення та корекція перспективних спотворень

    Args:
        image: Вхідне зображення

    Returns:
        Зображення з виправленою перспективою
    """
    try:
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        edges = cv2.Canny(gray, 50, 150, apertureSize=3)

        # Знаходження контурів
        contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

        # Знаходження найбільшого прямокутного контуру
        max_area = 0
        best_contour = None

        for contour in contours:
            area = cv2.contourArea(contour)
            if area > max_area:
                peri = cv2.arcLength(contour, True)
                approx = cv2.approxPolyDP(contour, 0.02 * peri, True)

                if len(approx) == 4:
                    max_area = area
                    best_contour = approx

        # Якщо знайдено відповідний контур
        if best_contour is not None and max_area > (image.shape[0] * image.shape[1] * 0.5):
            # Сортування точок
            pts = best_contour.reshape(4, 2)
            rect = np.zeros((4, 2), dtype="float32")
```

Рисунок 2.4 — Реалізація виявлення та корекція перспективних спотворень

2.3.2 Фільтрація шуму та покращення якості

Медіанний фільтр надзвичайно ефективний для видалення випадкових яскравих або темних точок , оскільки шумні викиди при сортуванні переміщуються до країв та не потрапляють у медіану. На відміну від фільтра Гауса медіанний краще зберігає чіткість країв об'єктів на зображенні (рисунок 2.5).

```python
def remove_noise(self, image: np.ndarray) -> np.ndarray:
    """
    Видалення шуму з зображення

    Args:
        image: Вхідне зображення

    Returns:
        Зображення без шуму
    """
    try:
        # Використання медіанного фільтру
        denoised = cv2.medianBlur(image, 3)

        # Морфологічні операції для видалення дрібних артефактів
        kernel = np.ones((2, 2), np.uint8)
        denoised = cv2.morphologyEx(denoised, cv2.MORPH_OPEN, kernel)
        denoised = cv2.morphologyEx(denoised, cv2.MORPH_CLOSE, kernel)

        logger.debug("Шум видалено")
        return denoised

    except Exception as e:
        logger.warning(f"Не вдалося видалити шум: {str(e)}")
        return image
```

Рисунок 2.5 — Реалізація медіанного фільтра

2.3.3 Бінаризація зображення

Метод Otsu — простий та ефективний алгоритм для автоматичного визначення оптимального глобального порогу при бінаризації зображення. Головною метою якого є розділити пікселі на дві групи: фон та об'єкт. Метод шукає поріг, який мінімізує внутрішньокласову дисперсію або максимізує різницю між класами, тому застосувати в проекті гарне рішення (рисунок 2.6).

```python
def otsu_binarization(self, image: np.ndarray) -> np.ndarray:
    """
    Бінаризація методом Otsu

    Args:
        image: Вхідне зображення

    Returns:
        Бінаризоване зображення
    """
    try:
        # Конвертація в відтінки сірого
        if len(image.shape) == 3:
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        else:
            gray = image

        # Застосування фільтру Гауса для згладжування
        blur = cv2.GaussianBlur(gray, (5, 5), 0)

        # Бінаризація методом Otsu
        _, binary = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

        # Конвертація назад в BGR
        binary = cv2.cvtColor(binary, cv2.COLOR_GRAY2BGR)

        logger.debug("Застосовано бінаризацію Otsu")
        return binary

    except Exception as e:
        logger.warning(f"Не вдалося виконати бінаризацію Otsu: {str(e)}")
        return image
```

Рисунок 2.6 — Метод Otsu

Адаптивна бінаризація, даний метод критично важливий для зображень із нерівномірним освітленням, тінями чи градієнтами тому також реалізований в проекті (рисунок 2.7).

```python
def adaptive_binarization(self, image: np.ndarray) -> np.ndarray:
    """
    Адаптивна бінаризація зображення

    Args:
        image: Вхідне зображення

    Returns:
        Бінаризоване зображення
    """
    try:
        # Конвертація в відтінки сірого
        if len(image.shape) == 3:
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        else:
            gray = image

        # Адаптивна бінаризація
        binary = cv2.adaptiveThreshold(
            gray,
            255,
            cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
            cv2.THRESH_BINARY,
            11,
            2
        )

        # Конвертація назад в BGR для сумісності
        binary = cv2.cvtColor(binary, cv2.COLOR_GRAY2BGR)

        logger.debug("Застосовано адаптивну бінаризацію")
        return binary

    except Exception as e:
        logger.warning(f"Не вдалося виконати бінаризацію: {str(e)}")
        return image
```

Рисунок 2.7 — Адаптивна бінаризація

2.3.4 Покращення контрасту

Метод CLAHE використовуватимемо для покращення контрасту зображення, через його специфіку роботи. Суть даного методу полягає в розділені зображення на менші частинки (тайли) і регулює контраст в кожній окремо, що допомагає уникнути надмірного освітлення чи затемнення областей зображення.

Принцип роботи в проекті:

— поділ зображення на тайли розміром 8x8 (рисунок 2.8);

— обмеження контрасту рівним 2.0;

— еквалізація кожного тайла, тобто перерозподіл пікселів, що перевищують поріг, рівномірно по гістограмі;

— білінійна інтерполяція між тайлами, що в результаті усуває артефакти на межах блоків.

```
try:
    # Конвертація в Lab колірний простір
    lab = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)

    # Розділення каналів
    l, a, b = cv2.split(lab)

    # Застосування CLAHE до L каналу
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    l = clahe.apply(l)

    # Об'єднання каналів
    lab = cv2.merge([l, a, b])

    # Конвертація назад в BGR
    enhanced = cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)

    logger.debug("Контраст покращено")
    return enhanced
```

Рисунок 2.8 — Метод CLAHE

2.3.5 Корекція освітлення

Мета ізолювати текстуру, текст від освітлення та шуму і для цього використовуємо корекцію освітлення в якій основою є метод віднімання фону (рисунок 2.9).

```
    try:
        # Конвертація в відтінки сірого
        if len(image.shape) == 3:
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        else:
            gray = image

        # Застосування розмиття для знаходження фону
        dilated_img = cv2.dilate(gray, np.ones((7, 7), np.uint8))
        bg_img = cv2.medianBlur(dilated_img, 21)

        # Корекція освітлення
        diff_img = 255 - cv2.absdiff(gray, bg_img)

        # Нормалізація
        norm_img = cv2.normalize(
            diff_img,
            None,
            alpha=0,
            beta=255,
            norm_type=cv2.NORM_MINMAX,
            dtype=cv2.CV_8UC1
        )

        # Конвертація назад в BGR якщо потрібно
        if len(image.shape) == 3:
            norm_img = cv2.cvtColor(norm_img, cv2.COLOR_GRAY2BGR)

        logger.debug("Освітлення виправлено")
        return norm_img
```

Рисунок 2.9 — Корекція освітлення

Суть роботи складається з декількох кроків:

— оцінка фону, dilate та medianBlur. Операція морфологічної дилатації розширює яскраві ділянки зображення, текст зникає внаслідок дилатації і ділянки заповнюються яскравими значеннями фону. Після чого отримуємо зображення без тексту, але ще з артефактами з яким працює далі медіанний фільтр. Завдяки великому ядру 21x21 фільтр остаточно згладжує результат дилатації. В результаті чого отримуємо карту освітлення, яка є оцінкою цього освітлення;

— віднімання фону від зображення, тобто береться оригінал зображення та віднімається оцінка освітлення, яку отримали на попередньому кроці. Після чого інвертуємо результат. Тепер фон який був близьким до 0 стає близьким до 255, тобто білим, а текст що був білим стає чорним, в результаті отримуємо чорний текст на білому фоні.

— останній крок це нормалізація, тобто покращення контрасту. Робить фон ідеально білим, а текст ідеально чорним.

2.4 Методи постобробки результатів розпізнавання

2.4.1 Базова постобробка тексту

Мета базової постобробки це виправлення поширених помилок форматування, щоб зробити текст чистим, читабельним і стандартизованим, перш ніж передавати його на збереження (рисунок 2.10) . Клас PostProcessor реалізує такий процес як конвеєр, що послідовно застосовує правила очищення. Першим видаляє зайві пробіли, а саме метод remove_extra_spaces вирішує три окремі проблеми пов'язані з пробілами:

— прибирає множинні пробіли;
— усуває пробіли на початку та вкінці рядків, якщо потрібно;
— чистить множинні порожні рядки.

```python
def remove_extra_spaces(self, text: str) -> str:
    """Видалення зайвих пробілів"""
    import re
    # Заміна множинних пробілів на один
    text = re.sub(r' +', ' ', text)
    # Видалення пробілів на початку і в кінці рядків
    text = '\n'.join(line.strip() for line in text.split('\n'))
    # Заміна множинних порожніх рядків
    text = re.sub(r'\n\n+', '\n\n', text)
    return text.strip()

def fix_punctuation(self, text: str) -> str:
    """Корекція розділових знаків"""
    import re
    # Видалення пробілів перед розділовими знаками
    text = re.sub(r'\s+([.,;:!?])', r'\1', text)
    # Додавання пробілу після розділових знаків (якщо його немає)
    text = re.sub(r'([.,;:!?])([^\s\n])', r'\1 \2', text)
    return text
```

Рисунок 2.10 — Базова постобробка тексту

Метод fix_punctuation приводить текст у відповідність до стандартів типографських правил, робить корекцію, тобто видаляє пробіл перед пунктацією чи додає після неї.

2.4.2 Базова автокорекція символів

Наступним кроком постобробки є виправлення помилок, що виникають через візуальну схожість символів. Дана проблема для проекту дуже актуальна, оскільки системи OCR, аналізуючи зображення, часто плутають гомогліфи. Гомогліфи це символи, які виглядають однаково чи схоже, але мають різний цифровий код. Текст з такими помилками стає непрофесійним та ламає аналітику. Метод auto_correct_text — базовий та швидких спосіб вирішення даної проблеми, що заснований на контексті мови (рисунок 2.11).

```
corrections = {
    'ukr': {
        'l': 'I',  # Заміна латинської l на українську I
        '0': 'O',  # Заміна нуля на O (в деяких контекстах)
    },
    'eng': {
        'I': 'I',  # Заміна української I на латинську I
    }
}

# Застосування корекцій залежно від мови
if 'ukr' in language:
    for wrong, right in corrections['ukr'].items():
        text = text.replace(wrong, right)

return text
```

Рисунок 2.11 — Базова автокорекція символів

Висновок до розділу

В розділі розглянуто методи та засоби для виявлення та розпізнавання тексту з акцентом на різні архітектури системи, математичне обгрунтування, алгоритми попередньої обробки зображення та постобробки результатів. Обгрунтовано комбінований підхід з використанням OCR–рушіїв Tesseract 4.0, як основного завдяки точності і підтримки української мови та гібридної архітектури на базі LSTM та PaddleOCR, як додаткового для підвищення швидкості та обробки складних макетів з RCNN–архітектурою.

Математично обґрунтована ключові моделі. LSTM для врахування контексту в послідовностях символів, RCNN для комбінації згорткових і рекурентних шарів, а також метрики якості, такі як CER, WER, відстань Левенштейна для обєктивної оцінки ефективності.

Детально описано алгоритми попередньої обробки та методи, що адаптивно готують зображення до розпізнавання, мінімізуючи вплив шуму, спотворень та нерівномірного освітлення.

Розглянуто базові процедури очищення тексту: видалення зайвих пробілів, корекція пунктуації та автокорекцію символів для усунення помилок пов'язаних з гомогліфами, що підвищує читабельність і точність кінцевого результату.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Опис вимог

Чітко визначені вимоги забезпечують відповідність розробленого програмного забезпечення функціональним і нефункціональним потребам користувача, а також мінімізують ризики переробок та гарантують структурованість та якість кінцевого продукту. Вимоги сформовано з урахуванням аналізу предметної області та цільових сценаріїв використання.

3.1.1 Функціональні вимоги

а) підтримка вхідних форматів таких як: JPEG, PNG, TIFF, PDF (скановані та зображення), пакетна обробка;

б) попередня обробка зображень, а саме: автоматична корекція нахилу, освітлення, шум, бінаризація;

в) виявлення текстових областей, локалізація тексту на складних макетах;

г) OCR–розпізнавання:

1) основний рушій — Tesseract;
2) альтернативний — PaddleOCR;
3) автоматичний fallback.

д) постобробка тексту: очищення пробілів, пунктуації, автокорекція гомогліфів;

е) експорт результатів: TXT, DOCX, PDF, JSON;

є) GUI: вибір файлів, прогрес–бар, попередній перегляд, редагування тексту, налаштування OCR;

ж) пакетна обробка, яка складається з підтримки черги, логування та збереженням статусів.

Функціональними вимогами є реакція системи на дії користувача, а тому опишемо їх відповідно того які можливості матиме користувач працюючи з системою:

— повинен мати можливість завантажити файл відповідного формату;

— спостерігати за прогресом обробки;

— користувач повинен побачити відображення завантаженого ним файлу;

— отримати результат у новому вікні де він зможе також його редагувати додатково самостійно;

— обирати за допомогою якої моделі буде відбуватися виявлення та розпізнавання тексту;

— збереження редагованого файлу у відповідно обраний формат в папку на своєму локальному комп'ютері;

— можливість перегляду статистики з результатами точності вилучення тексту.

3.1.2 Нефункціональні вимоги

— Кросплатформеність, тобто можливість використання на Windows, Linux, macOS.

— Інтуїтивний інтерфейс.

— Масштабованість, можливість додати гібридну модель та використання YOLO в передобробці зображення.

— Безпека, тобто обробка лише локальних файлів.

— Надійність та продуктивність.

3.1.3 Обмеження

— Без підтримки рукописного тексту, але з додаванням у майбутніх версіях.

— Без гібридної моделі, додавання у майбутній версії.

— Без попередньої обробки з використанням YOLO, додавання у майбутній версії.

3.2 Вибір технологій

Вибір технологій це ключовий етап розробки, тому що має визначати не лише технічну реалізацію, а й продуктивність, гнучкість, зручність використання та можливості подальшого розширення системи. Для даного проекту технологічний стек сформовано з урахуванням вимог до високої точності розпізнавання, швидкодії на CPU, кросплатформеності, відкритості коду та простоти розгортання.

Мовою програмування обрано Python, яка є продуктивною та широко підтримуваною з багатою екосистемою бібліотек для комп'ютерного зору, машинного навчання та розробки інтерфейсів. З основних переваг є читабельність коду, швидка прототипізація.

Для OCR–розпізнавання використано два рушія. Основний Tesseract, який побудований на основі LSTM–нейронних мережах. Дозволяє тонке налаштування через pytesseract, а також підтримує понад 120 мов та забезпечує високу точність на

друкованих текстах. Альтернативний рушій PaddleOCR, що гарно працює з складними макетами завдяки своїй архітектурі.

Для обробки зображень використано бібліотеку OpenCV, яка є провідною бібліотекою комп'ютерного зору та забезпечує швидке виконання операцій реального часу: бінаризацію адаптивну та Otsu, фільтрацію шуму за допомогою медіанного фільтру, корекцію нахилу, покращення контрасту CLAHE. А для ефективних векторних обчислень, роботи з масками та афінними матрицями бібліотека Numpy.

Для реалізації інтерфейсу користувача, щоб забезпечити кросплатформенність, функціональність системи та сучасний графічний інтерфейс користувача, було обрано фреймворк PyQt5. Потужний фреймворк, який забезпечує нативний вигляд на Windows, Linux та macOS. Ключовою особливістю є багатопотоковість через QThread і механізм Signal/Slots, що дозволяє виконувати тривалі операції OCR без блокування інтерфейсу. Архітектура коду побудована за модифікованим MVC–патерном, де модель — це логіка обробки, представлення — це віджети, а контролер — це координація подій.

Альтернативою PyQt5 розглядалася бібліотека Tkinter, яка вбудована у стандартну бібліотеку Python. Проста у використанні, не потребує додаткових залежностей та підходить для базових застосунків. Проте її відхилено через:

— застарілий зовнішній вигляд, не нативний на сучасних операційних системах;

— обмежену підтримку багатопотоковості та складних віджетів;

— відсутність сучасних компонентів, таких як прогрес–бар з плавною анімацією, вкладки з drag-and-drop;

— складність створення професійного UI без додаткових тем. PyQt5 перевершує Tkinter за функціональністю, естетикою та можливостями масштабування інтерфейсу.

3.3 Розробка архітектури

Розробка архітектури це вирішальний етап в побудові програмного забезпечення. Вона має чітко відповідати та задовольняти вимоги програми. Враховувати показники, які є нефункціональними вимогами, а саме:

— масштабованість;

— відмовостійкість;

— безпечність;

— продуктивність.

Для цього розглянемо існуючі архітектури програмного забезпечення для десктопних застосунків.

3.3.1 Model—ViewViewModel

Model—ViewViewModel — це архітектурний патерн, який вмкористовується для чіткого розділення логіки, представлення та стану інтерфейсу користувача. Він особливо популярний у додатках з динамічним UI. Розглянемо принцип роботи:

- View відображає дані з ViewModel через привязку;
- Користувач взаємодіє з View, викликає команду з ViewModel;
- ViewModel оновлює Model (наприклад запускає OCR);

— Model змінюється, ViewModel оновлюється, View автоматично перерисовується.

Перевагами є:

— автоматичне оновлення UI;

— легке тестування ViewModel, без GUI;

— реактивність, як у фреймворках React, Vue;

— чисті View, без логіки.

Недоліки:

— складний для простих додатків;

— не нативний у PyQt5

3.3.2 Clean Architecture

Clean Architecture — архітектурний підхід, запропонований Робертом Мартином, який забезпечує максимальну незалежність бізнес–логіки від зовнішніх фреймворків, баз даних, технологій і UI. Мета якого є центральна бізнес–логіка (ядро), а все інше то зовнішні шари, які можна легко замінити.

Перевагами є:

— незалежність від технологій;

— повна тестованість;

— гнучкість;

— довговічність.

Недоліками:

— надмірна складність;

— кожен OCR–модуль повинен мати свій адаптер;

— більше коду ніж потрібно і в підсумку повільна розробка;

— не використовує PyQt5 Signals.

3.3.3 Архітектура застосунка

Проаналізував існуючі архітектури, які можна використати в даному програмному забезпеченні відповідно обраних технологій, було прийнято рішення зосередитися на MVC архітектурі. На верхньому рівні організації системи, забезпечуючи чітке розділення логіки, представлення та взаємодії використаємо обраний варіант (рисунок 3.1). Що дозволить легко підтримувати, тестувати та масштабувати код, а також повторно використовувати компоненти. Система побудована спеціальна під OCR–застосунок на PyQt5 з обробкою зображень, гібридним розпізнаванням та багатопотоковістю. Взаємодія між шарами здійснюється через PyQt5 Signals/Slots, який є нативним механізмом реактивності, що усуває потребу в додаткових бібліотеках прив’язки даних.

Model — повністю незалежний від GUI, містить усю бізнес–логіку: попередню обробку, OCR, постобробку та управління файлами.

View — віджети PyQt5, які лише відображають дані та приймають дії користувача.

Controller — MainWindows, координує потік подій, запускає обробку в QThread, оновлює UI через сигнали.

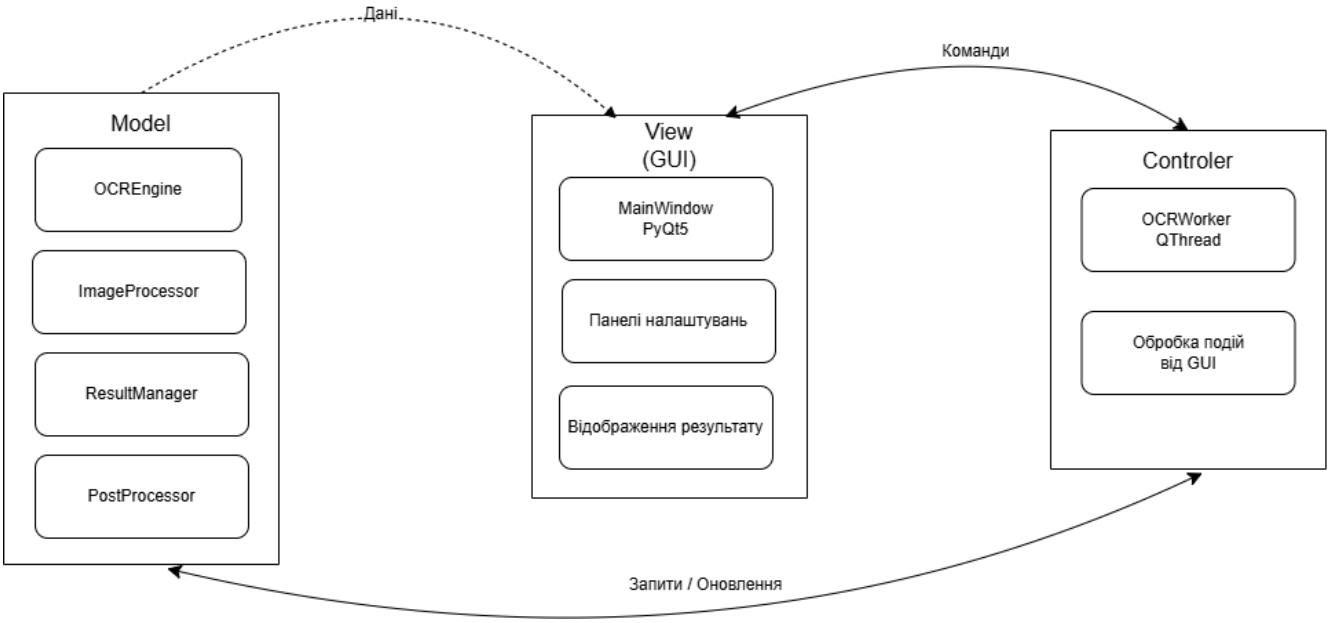


Рисунок 3.1 — Архітектура додатку

Побудуємо та розглянемо діаграму послідовності розпізнавання (рисунок 3.2).

Крок 1 — додавання файлу:

— користувач обирає файл/папку;

— MainWindow додає файли до черги.

Крок 2 — запуск розпізнавання:

— користувач натискає [Розпізнати];

— MainWindow оновлює налаштування.

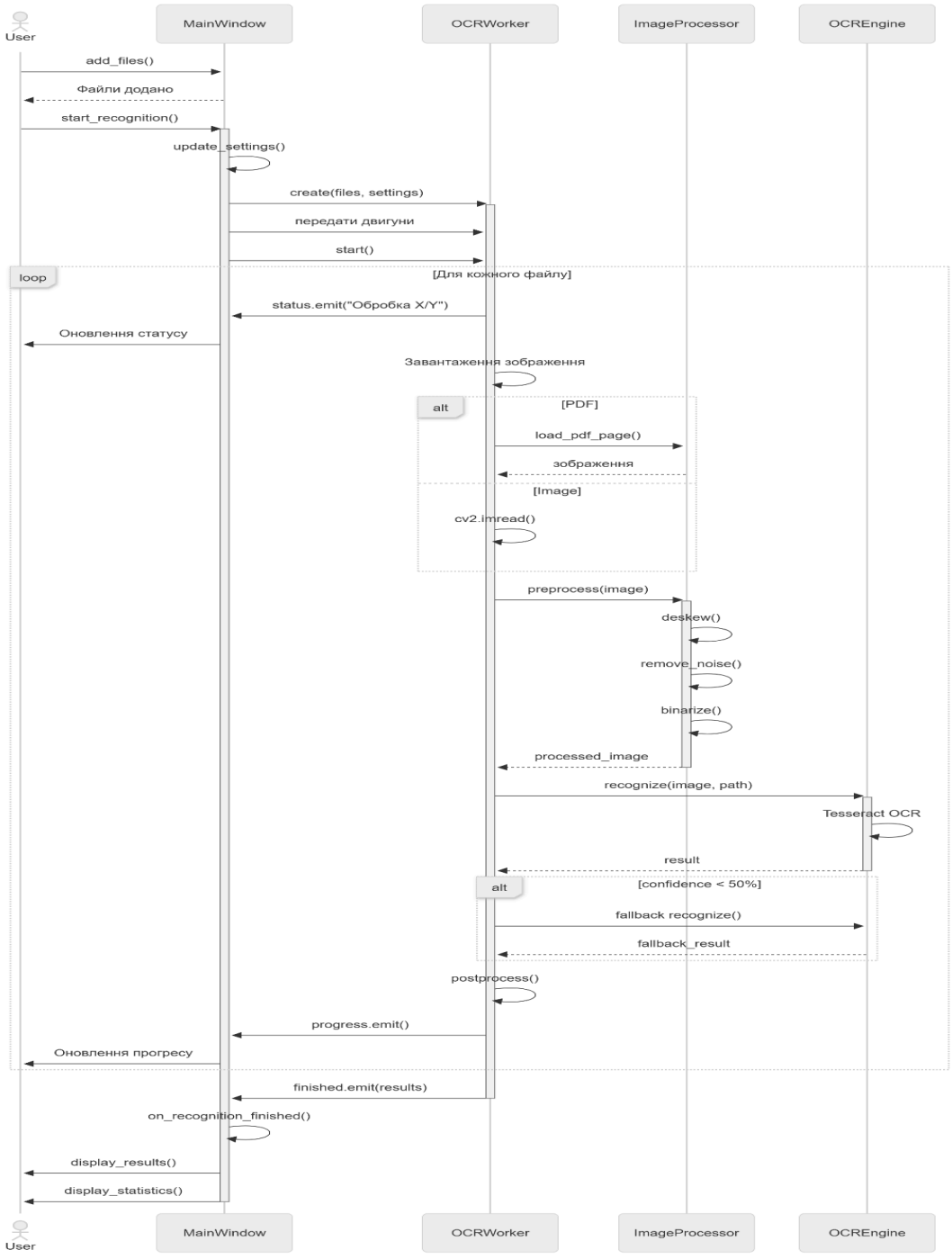


Рисунок 3.2 — Діаграма послідовності розпізнавання

Крок 3 — створення робочого потоку:

— MainWindow створює окремий OCRWorker для кожного файлу;

— передає шлях, налаштування, мову.

Крок 4 — передача рушія: OCRWorker ініціалізує Tesseract/ PaddleOCR.

Крок 5 — запуск обробки: OCRWorker запускається в окремому потоці (QThread).

Крок 6 — оновлення статусу, для кожного файлу: MainWindow оновлює QStatusBar і QProgressBar.

Крок 7 — Завантаження зображення:

— якщо PDF то витягує сторінку через PyPDF2 плюс Pillow;

— якщо зображення то читає через cv2.imread().

Крок 8 — попередня обробка зображення:

— виправлення нахилу застосовує deskew();

— медіанний фільтр, remove_noise();

— адаптивна бінаризація , binarize();

— покращення контрасту, CLAHE.

Крок 9 — основне розпізнавання Tesseract: викликає pytesseract.image_to_string().

Крок 10 — перевірка довіри: якщо довіра <50% йде на fallback/

Крок 11 — Fallback до PaddleOCR:

— використовує PaddleOCR;

— повертається fallback_result.

Крок 12 — постобробка тексту:

— очищення зайвих пробілів, гомогліфи;

— автокорекція.

Крок 13 — оновлення прогресу: MainWindow оновлює QProgressBar.

Крок 14 — завершення обробки одного файлу: results (текст, час, CER, WER).

Крок 15 — відображення результатів:

— додає вкладку в QTabWidget;

— показує оригінал (QLabel);

—показує розпізнаний текст (QTextEdit).

Крок 16 — цикл для наступного файлу:

— повертається до кроку 6;

— прогрес X/Y файлів.

3.4 Інструкція користувача

Головна сторінка користувача розроблена так, щоб користувач максимально швидко та інтуїтивно отримав результат, не потребуючи спеціальних знань.

Інтерфейс побудовано за принципом — від відкриття до збереження розпізнаного документа (рисунок 3.3).

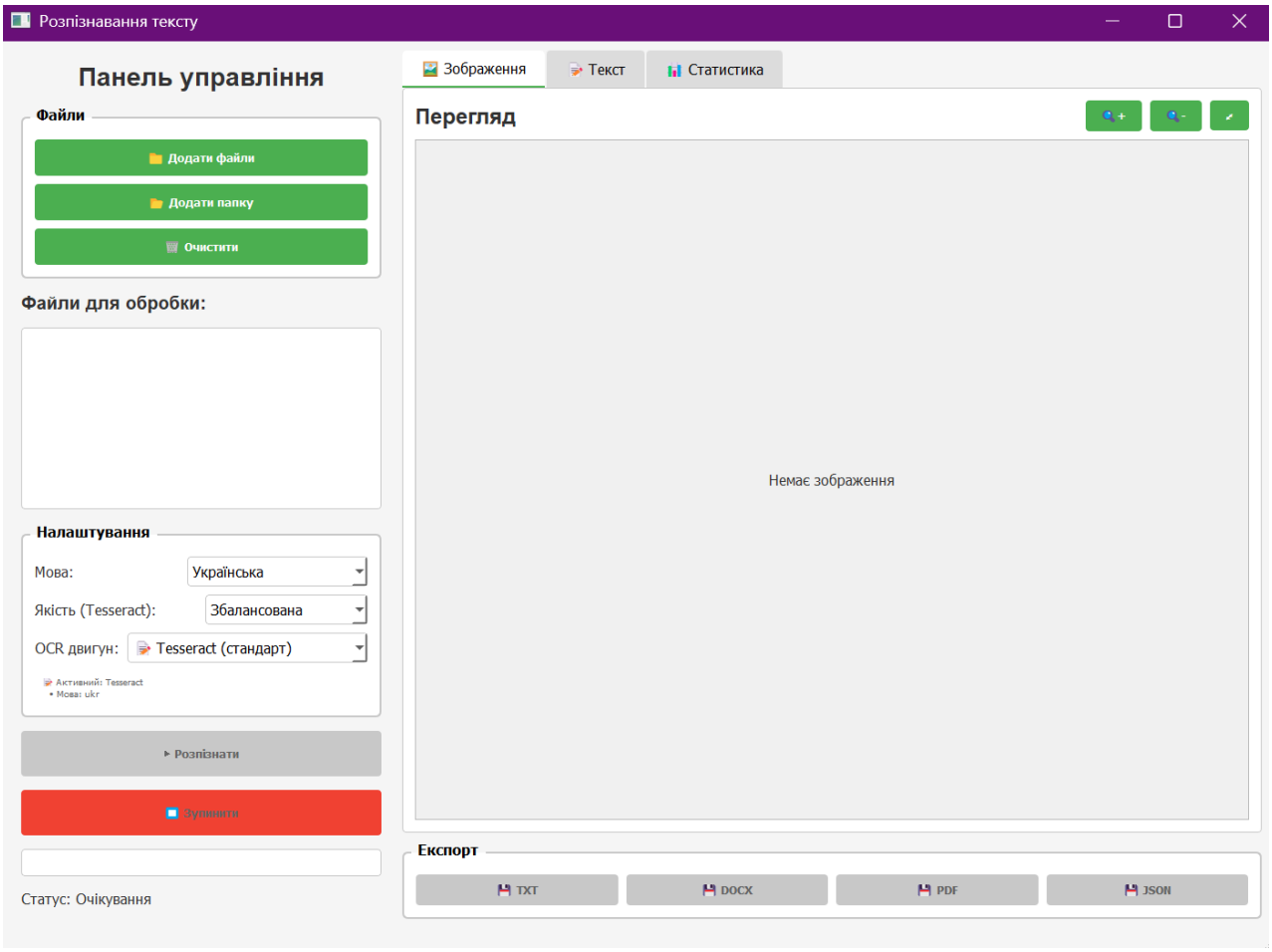


Рисунок 3.3 — Головна сторінка

Дана сторінка для зручності роботи розбита на блоки:

— панель управління;

— робоча зона з відповідними вкладками;

— панель експорту.

Панель управління розташована зліва екрана і містить:

— зону з кнопка для завантаження файлу чи папки;

— вікно в якому користувач бачить назву файлу, який він завантажив;

— кнопку для очистки завантаження;

— зона налаштування та обрання OCR–рушія;

— кнопки для запуску та зупинки процесу розпізнавання.

Першим кроком потрібно додати файл для цього натискаємо на кнопку “додати файл”, після чого з’являється вікно, де обираємо відповідний файл чи папку і в ній вже файл зі свого комп’ютера та натискаємо Відкрити (рисунок 3.4). Аналогічно для завантаження папки пророблюємо такі ж кроки і отримуємо у вікні повний пакет файлів для розпізнавання тексту, що в них міститься.

У зоні налаштування та обрання OCR–рушія користувач обирає мову з випадаючого списку. Для обрання в даній версії доступно: англійська, українська або англійська з українською для документів, що мають відповідно в зміті обидві мови. Можна обрати якість для основного рушія Tesseract:

— збалансована;

— висока точність;

— швидка.

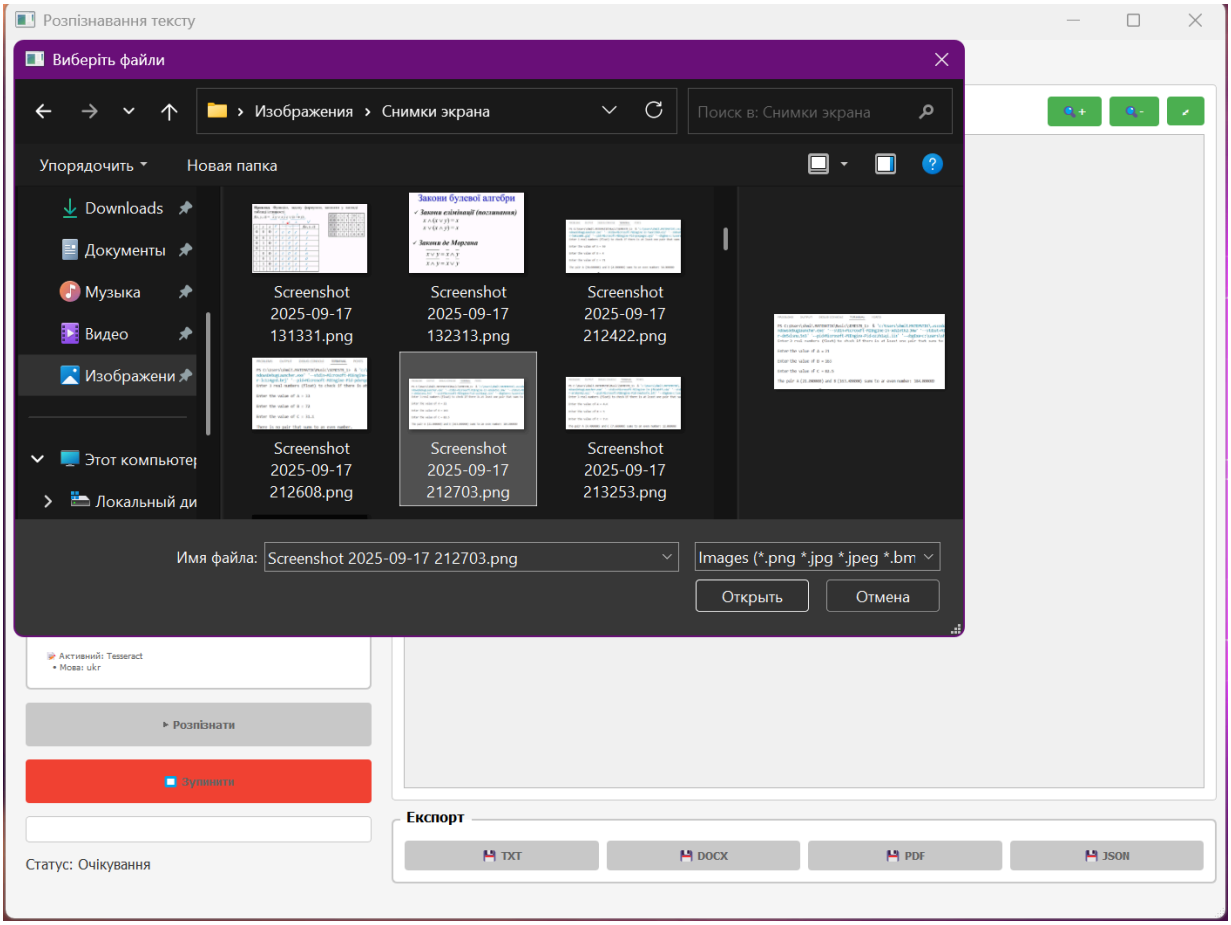


Рисунок 3.4 — Вибір файла

Коли файл завантажено, натисканням кнопки Розпізнати виконується розпізнавання тексту та можна спостерігати за прогресс баром у відсотках та індикації в нижньому лівому куті екрана (рисунок 3.5). Під час розпізнавання якщо натиснути кнопку червоного кольору Зупинити, виходить вікно для підтвердження зупинки і після натискання процес розпізнавання зупиниться і нічого не поверне. Якщо ж не переривали процес розпізнавання, після 100% завершення з’явиться вікно з інформацією результата роботи системи, яке закрити можна натиснувши кнопку зеленого кольору ОК чи на хрестик у верхньому правому кутку віконця (рисунок 3.5).

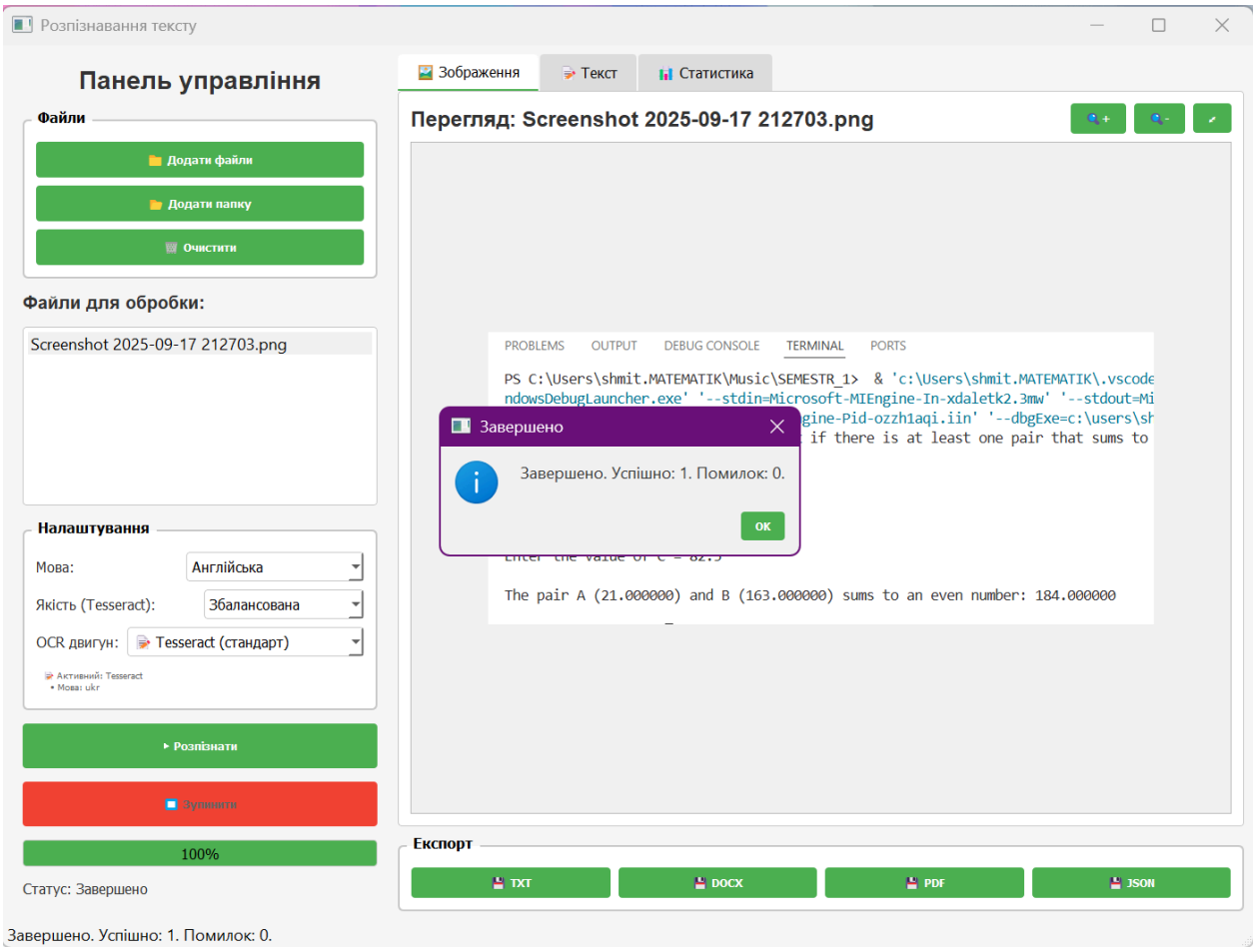


Рисунок 3.5 — Розпізнавання та вивід вікна результату

На вкладці Зображення завантажений оригінал файлу, тобто фото. На вкладці Текст, після розпізнавання з’являється розпізнаний текст та перейшовши на неї з’являються додаткові можливості, такі як: редагування тексту в даному полі вікна; копіювати текст, натиснувши кнопку Копіювати та очистити поле з натисканням кнопки Очистити (рисунок 3.6).

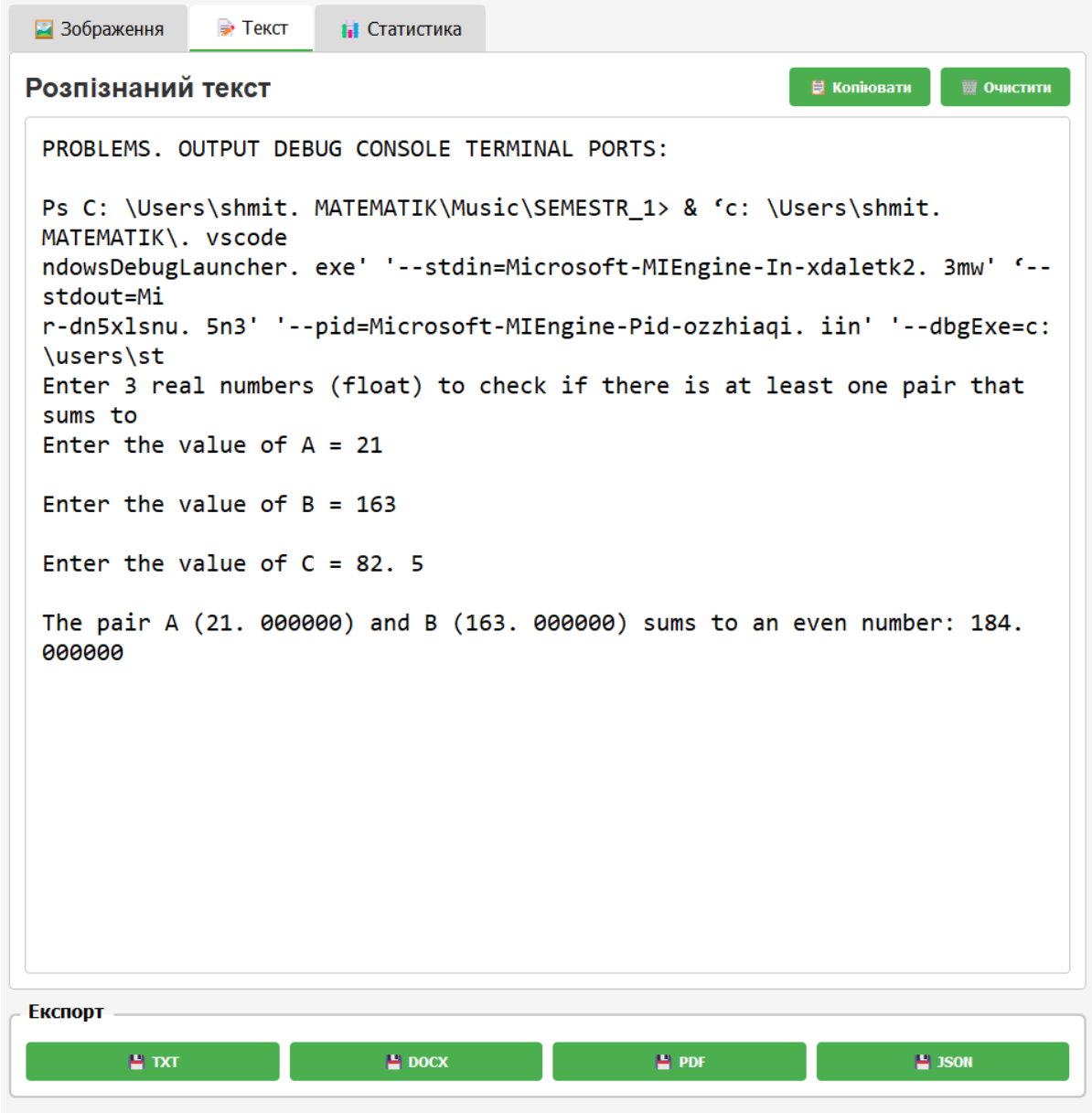


Рисунок 3.6 — Вкладка з розпізнаним текстом

В блоці робочої зони також можна переглянути статистику, перейшовши на відповідну вкладку з назвою Статистика (рисунок 3.7). Вона містить загальні статистичні дані: кількість файлів, інформацію роботи системи про успішність та помилки, кількість символів, кількість слів, обраховує та повертає середній відсоток

впевненості розпізнавання та час загальний який був витрачений на виявлення та розпізнавання тексту; інформацію про налаштування: мови та обраної моделі; деталі, назву файлу, рушій, кількість символів, кількість слів та впевненість.

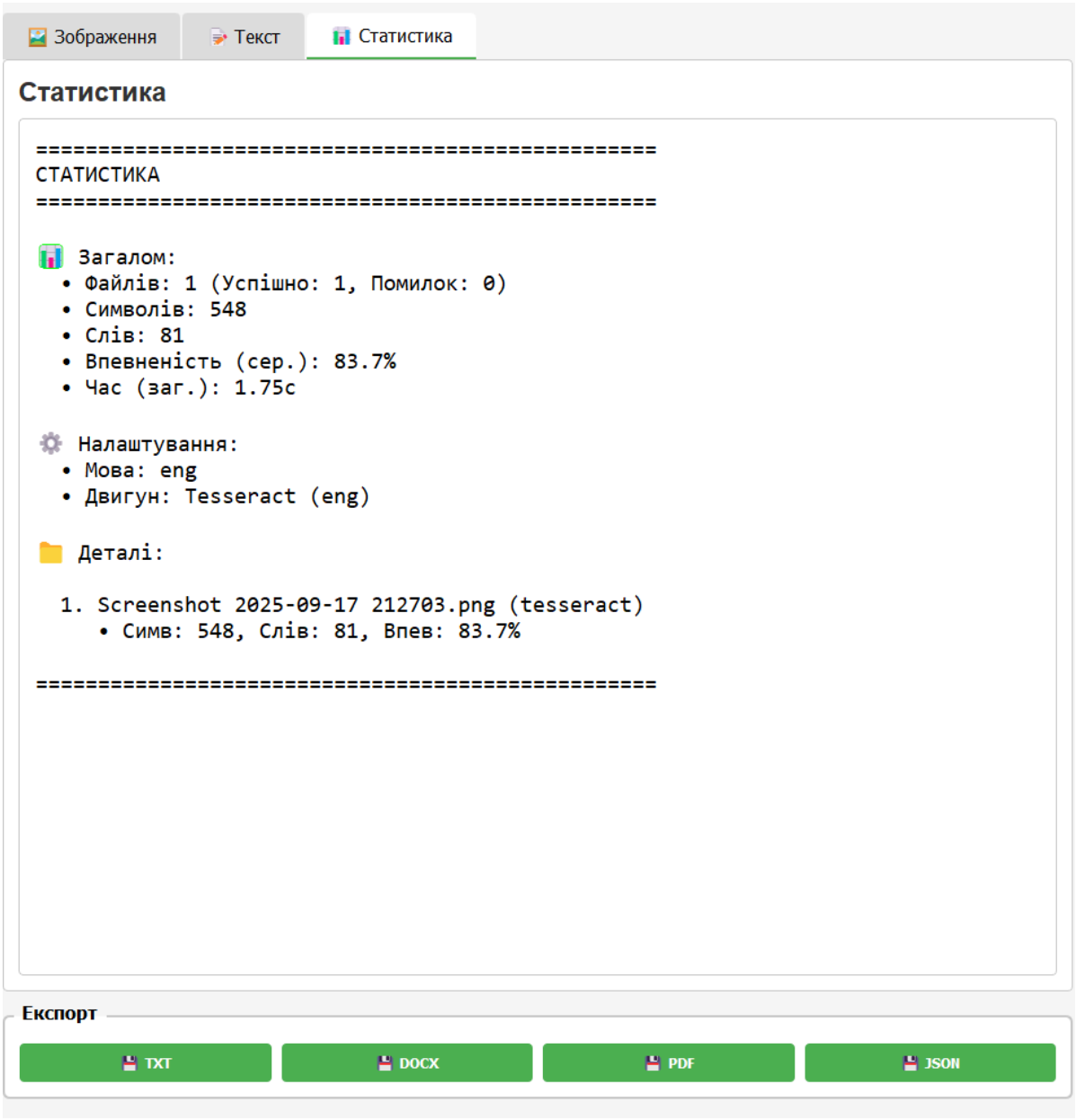


Рисунок 3.7 — Вкладка з статистичними даними

На вкладці Зображення, крім оригіналу документа є кнопки для збільшення та зменшення, натискаючи які відповідно збільшується (рисунок 3.8) або зменшується.

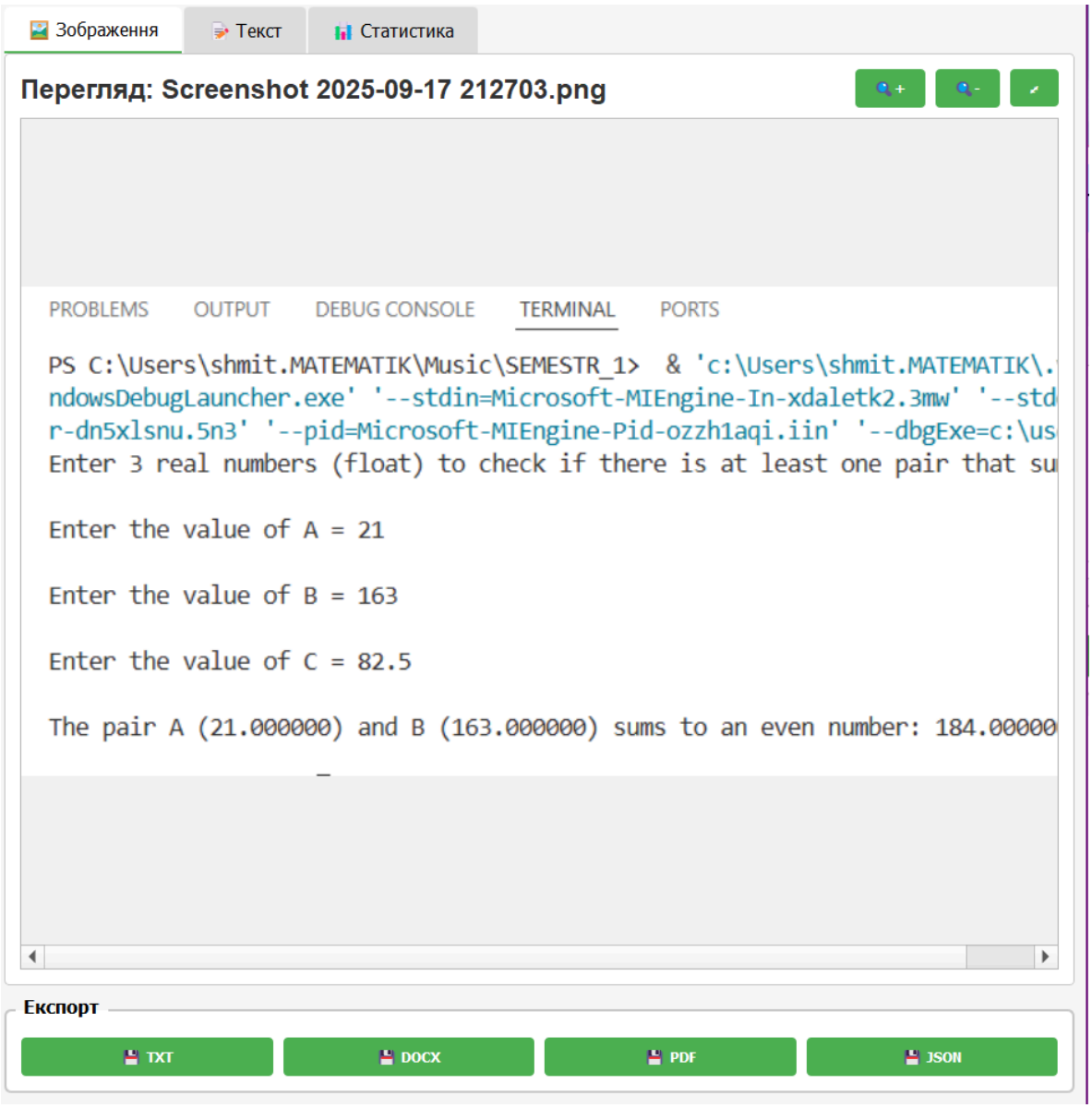


Рисунок 3.8 — Збільшення/зменшення тексту

Реалізовано редагування тексту безпосередньо у тексті, що повернула модель. Можна вписати, змінити текст чи окремі букви/символи за допомогою клавіатури комп’ютера (рисунок 3.9) і якщо експортувати у обраний формат ці зміни залишуться у вашому новому файлі (рисунок 3.13).

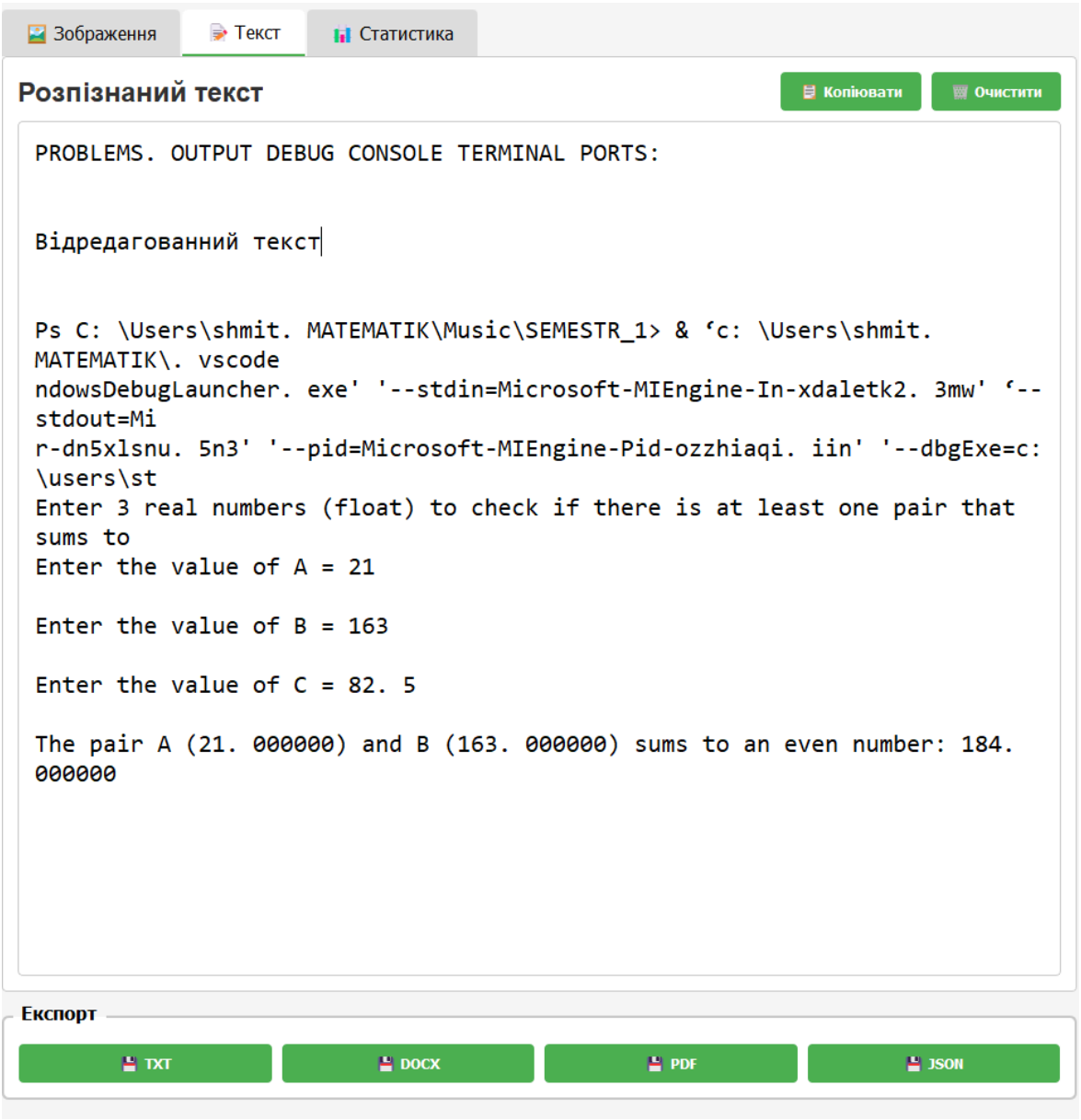


Рисунок 3.9 — Редагування розпізнаного тексту

Якщо натиснути кнопку Очистити то з’являється вікно в якому потрібно підтвердити чи скасувати очистку тексту відповідно натиснувши кнопку Yes чи No (рисунок 3.10).

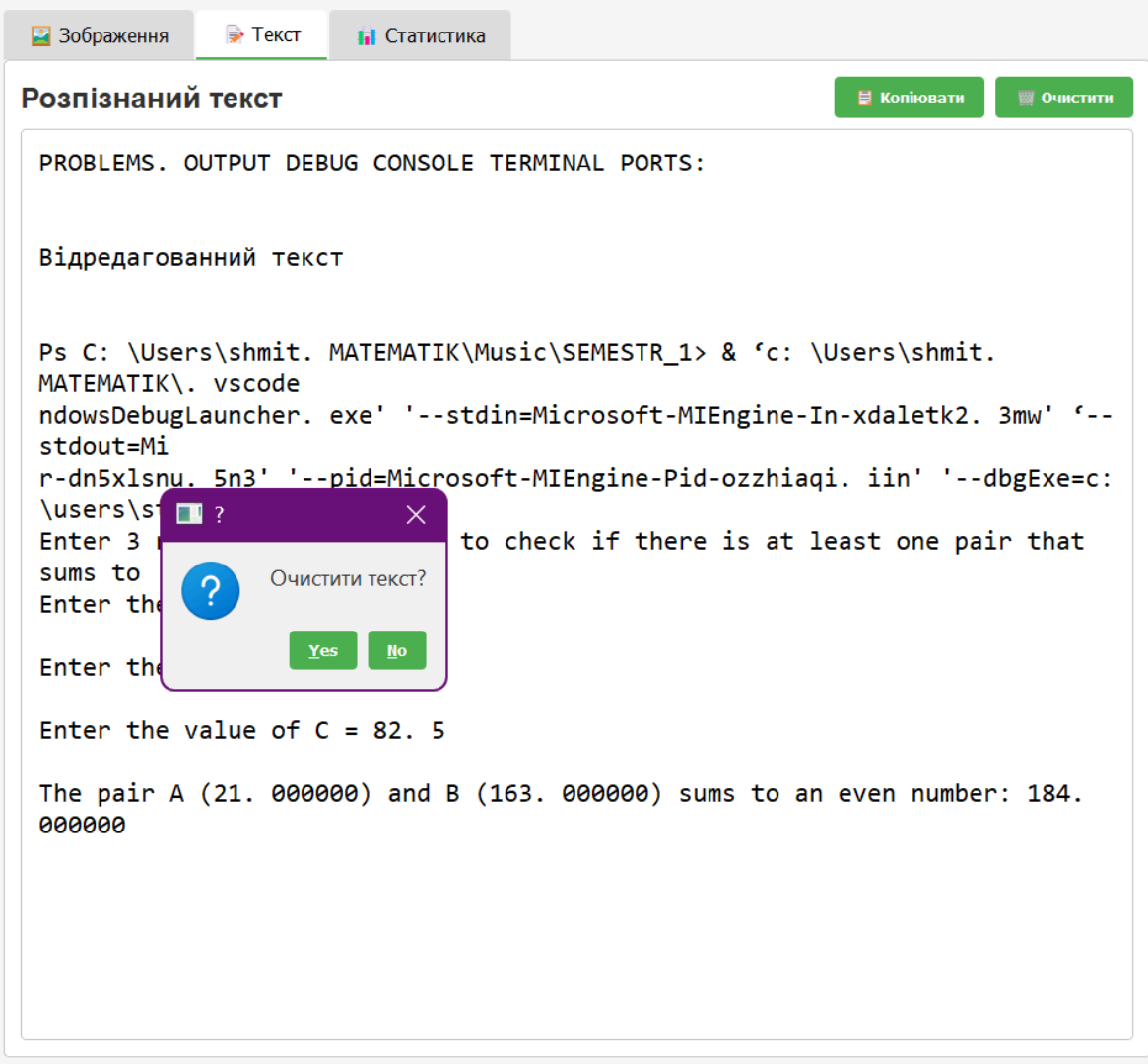


Рисунок 3.10 — Очистка у вікні розпізнаного тексту

Для збереження розпізнаного тексту потрібно обрати варіант формату, що представлені у блоці Експорту і натиснути відповідну кнопку. Для прикладу, якщо потрібно документ отримати у форматі TXT то натискаємо кнопку з назвою TXT, після чого на екрані з'являється вікно де обираємо шлях куди потрібно зберегти, прописуємо назву файла та натискаємо кнопку Зберегти (рисунок 3.11). Якщо натиснути кнопку Відміна відповідно файл не збережеться.

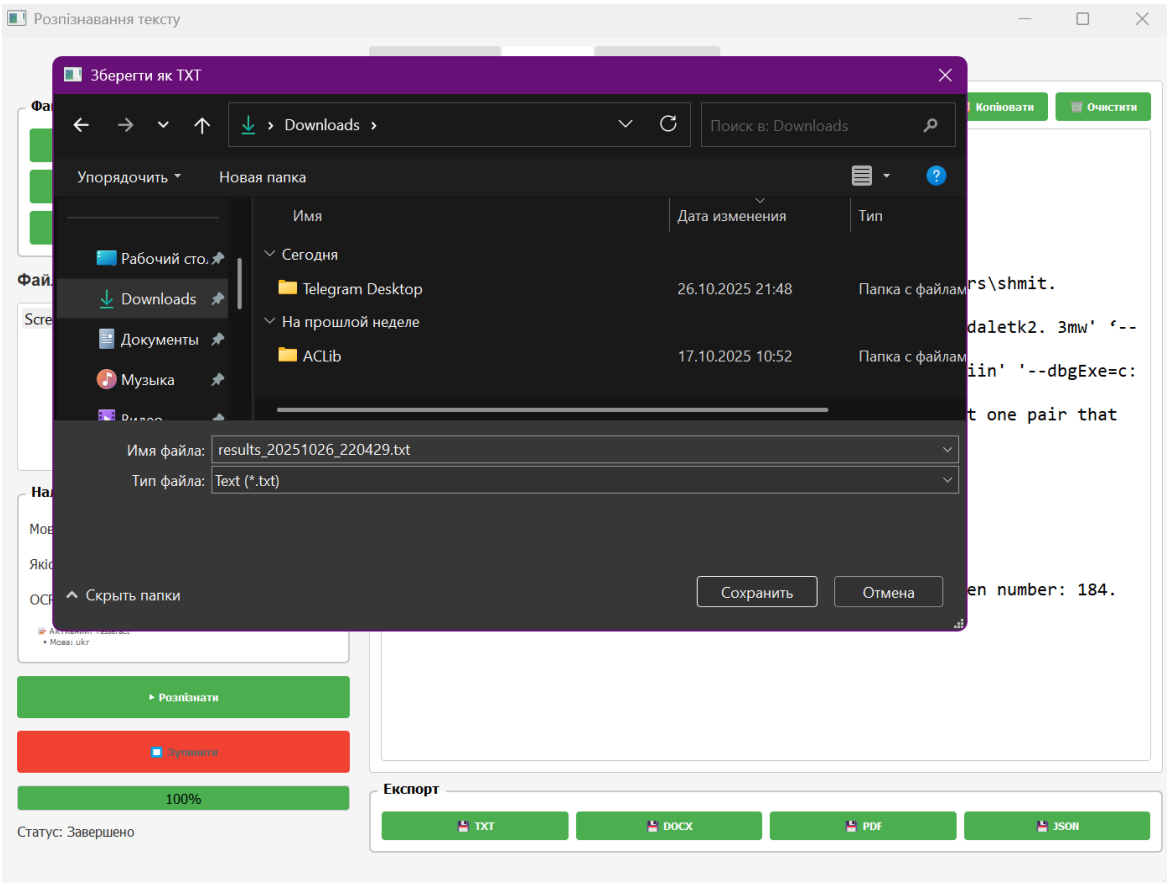


Рисунок 3.11 — Збереження розпізнаного та відредагованого тексту в txt формат

Аналогічні дії для збереження файла у форматі docx (рисунок 3.12). Результатам перевіряємо, перейшовши за шляхом відповідного збереження і відкриваємо у

програмі Word чи Google Docs (рисунок 3.13). В документ також зберігається інформація про результати розпізнавання і все що було додано нв етапі редагування в самій програмі.

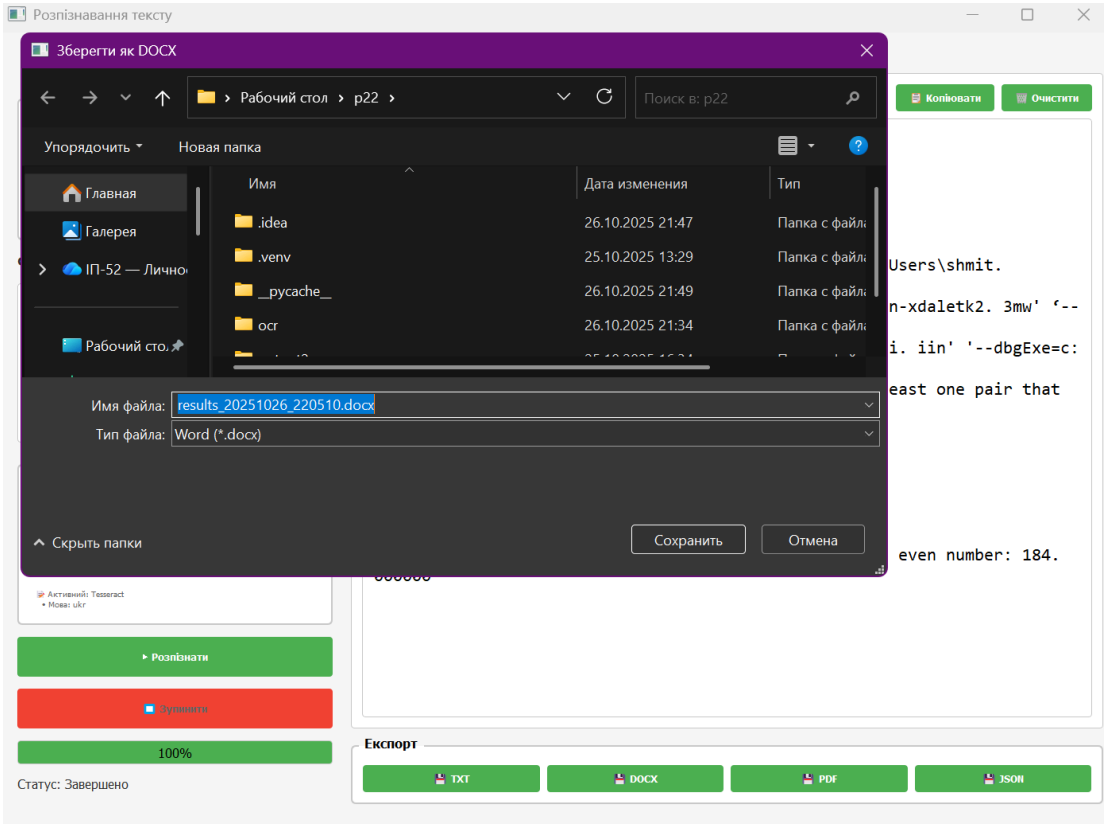


Рисунок 3.12 — Збереження в docx формат

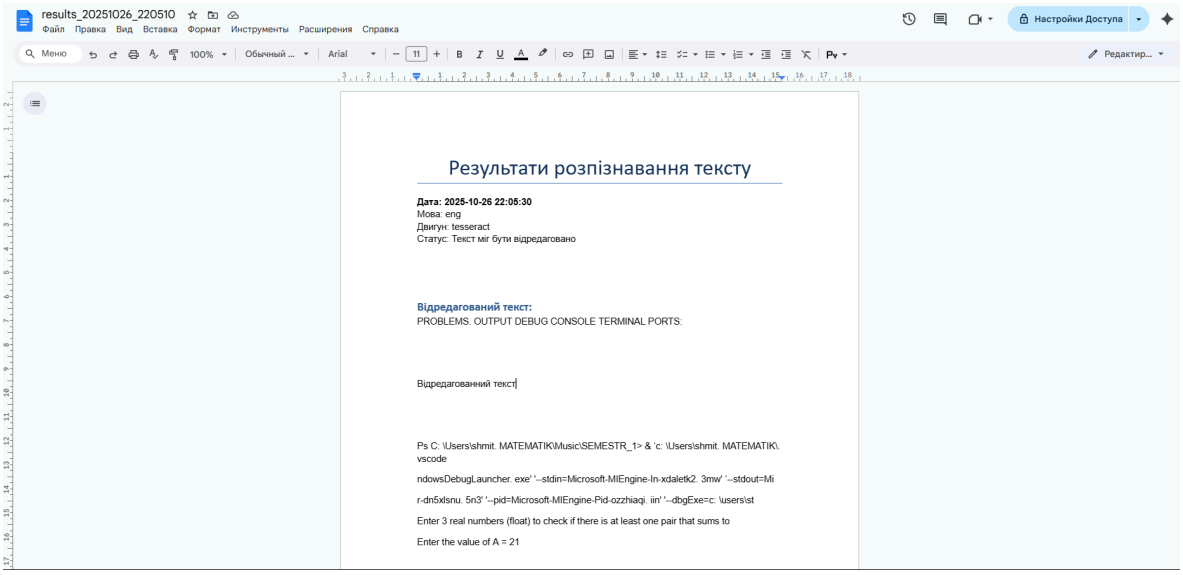


Рисунок 3.13 — Результат збереженого файлу

3.5 Експериментальні дослідження

Проведемо експериментальні дослідження, для цього проведемо декілька експериментів. Протестуємо реалізоване програмне забезпечення на всіх не текстових форматах файлів, на пакетній обробці pdf-файлів, задіюючи кожен рушій, після чого занесемо результати в таблицю.

Для першого експерименту завантажимо pdf-файл, назва якого “Мережі_лаб1”, що складається з 17 сторінок українського тексту та картинок. В налаштуванні оберемо мову “Українська”, “Збалансована” якість розпізнавання та рушій “Tesseract” (рисунок 3.14). В результаті роботи виявлено та розпізнано 3400 слів на 17 сторінках, загальний час розпізнавання склав 57.30 секунд з середньою впевненістю 70% по всім 17 сторінках. А також відповідно кожної сторінки отримали статистику з % впевненості (рисунок 3.15).

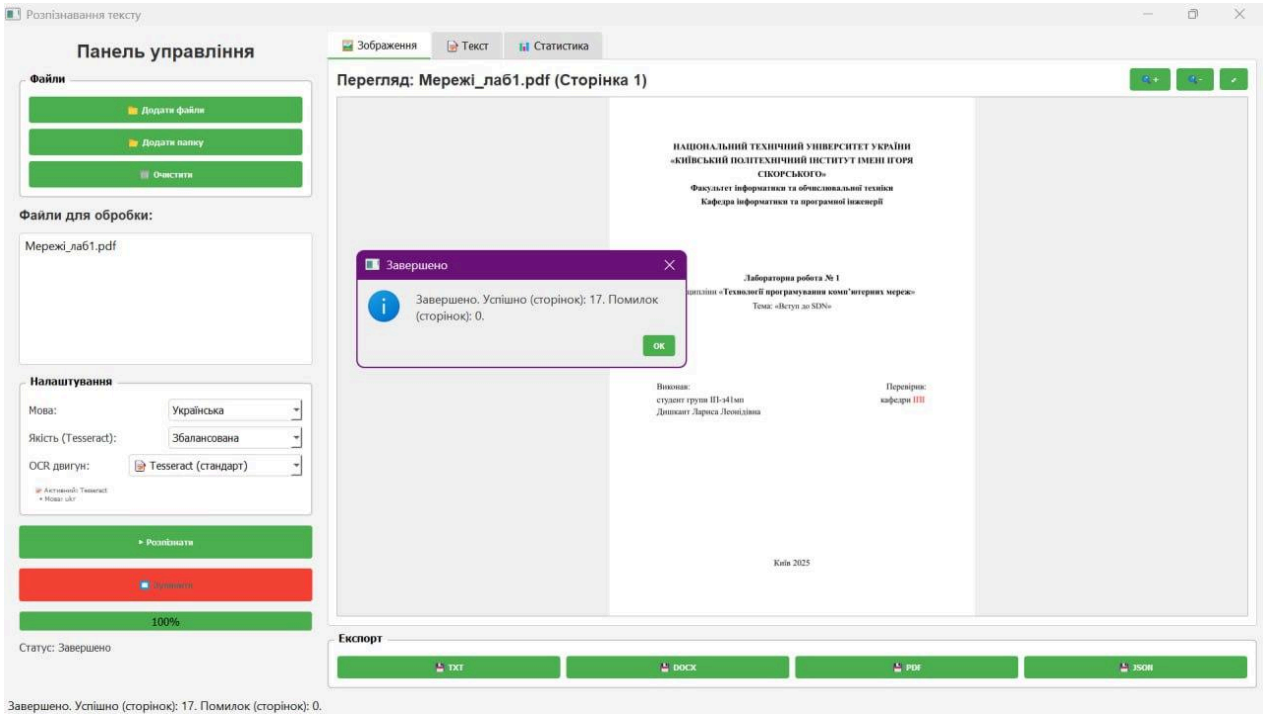


Рисунок 3.14 — Виявлення та розпізнавання тексту в документі pdf

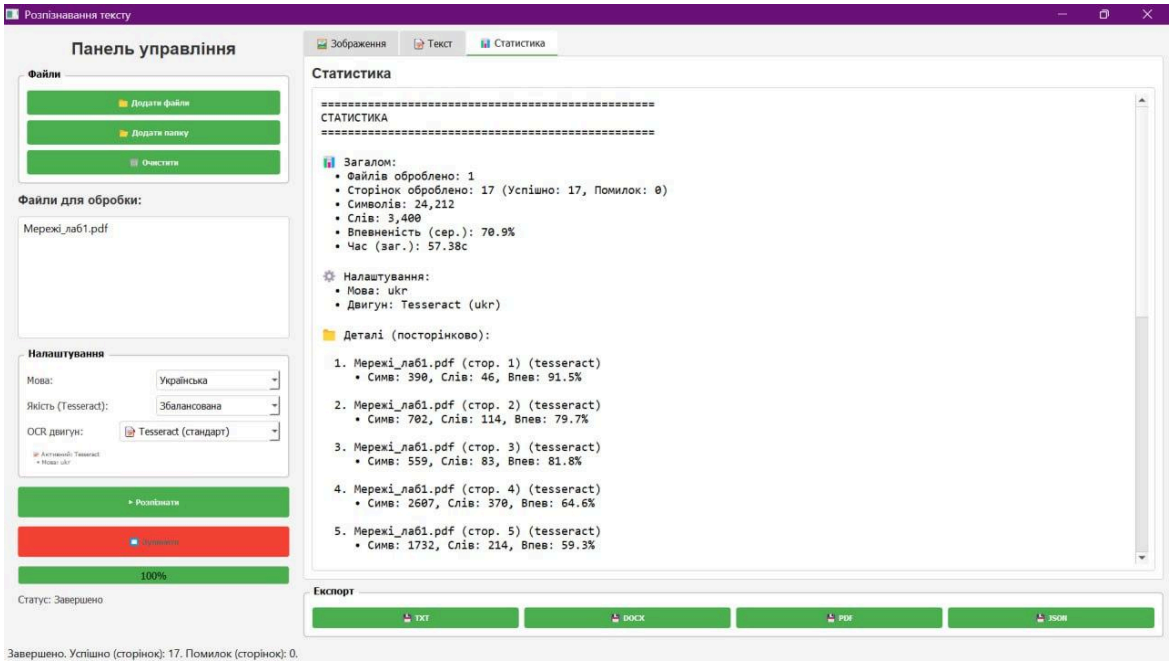


Рисунок 3.15 — Результат розпізнавання багато сторінкового pdf документа

Наступним кроком для виявлення та розпізнавання використаємо ядро PaddleOCR на цьому ж документі (рисунок 3.16). Як бачимо отримали кращі результати, які складають 3563 слова з впевненістю 93.7% (рисунок 3.17), отримані результати занесемо до таблиці.

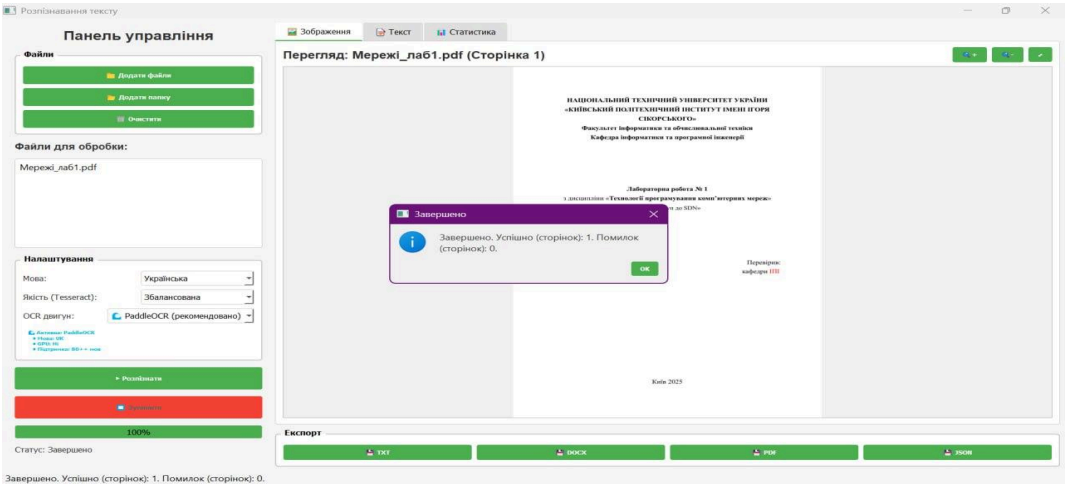


Рисунок 3.16 — Розпізнавання тексту в документі pdf за допомогою PaddleOCR

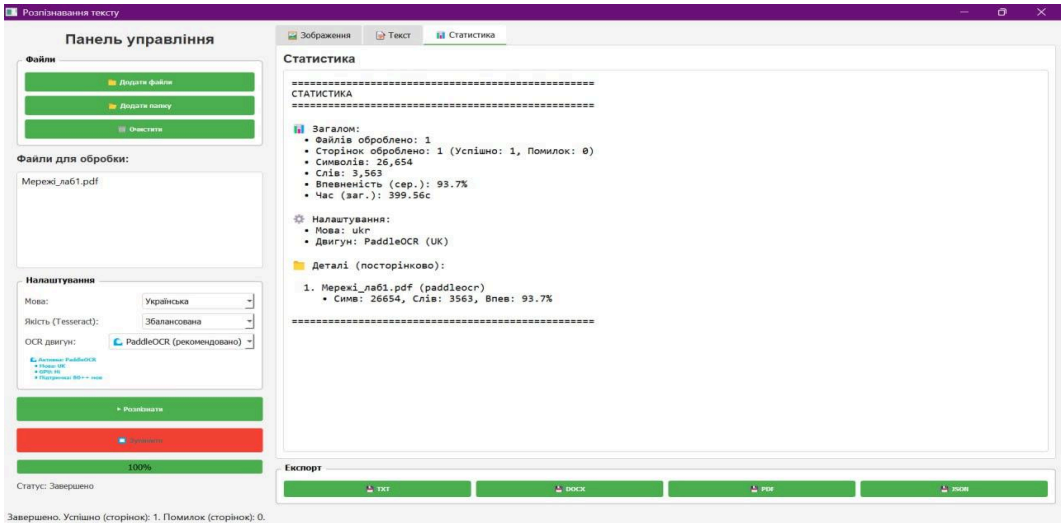


Рисунок 3.17 — Результат роботи PaddleOCR

Завантажуємо ще один файл pdf “Реферат_Дишкант_Л_Л” та запускаємо розпізнавання відразу двох файлів на OCR рушії “Tesseract” (рисунок 3.18) і отриманий результат (рисунок 3.19) вносимо до таблиці для подальшого аналізу.

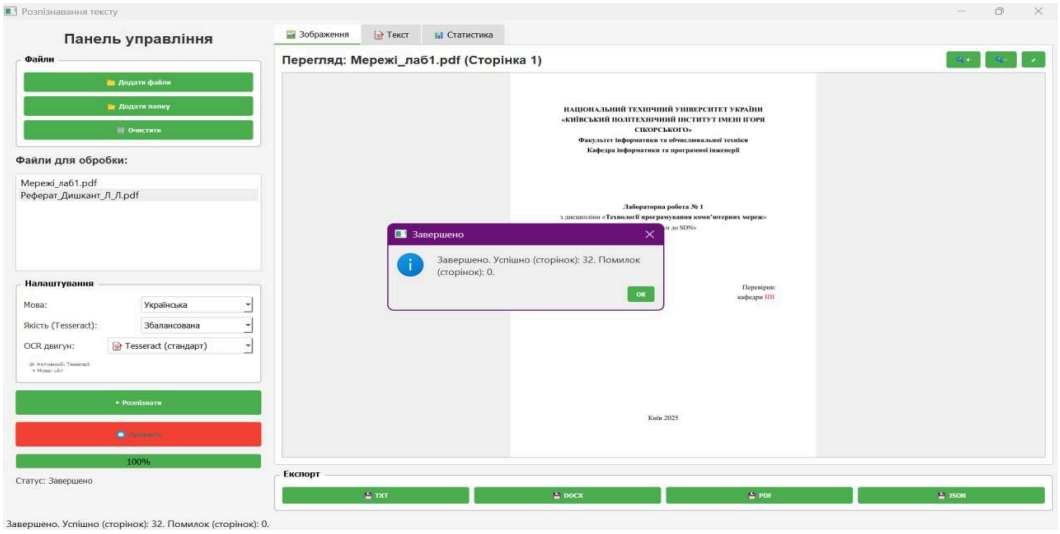


Рисунок 3.18 — Виявлення та розпізнавання тексту в pdf з двох файлів

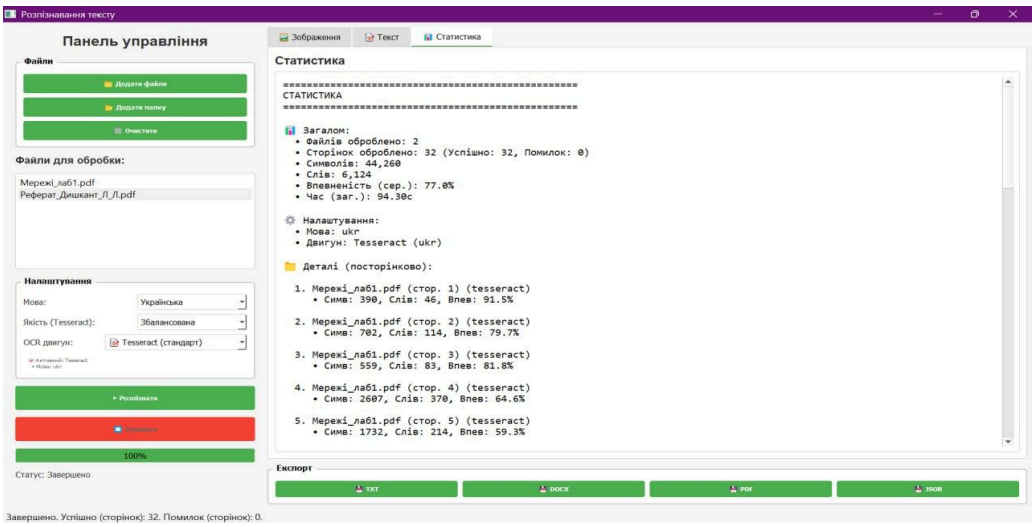
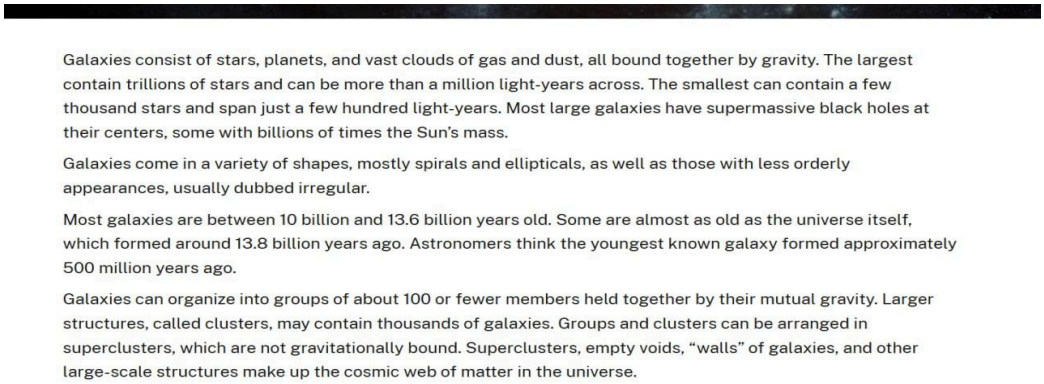


Рисунок 3.19 — Статистичні дані результату розпізнавання двох файлів

Наступним експериментом буде виявлення та розпізнавання тексту на фото формату png, для цього завантажуємо даний файл, обираємо мову “Англійська” та рушій Tesseract (рисунок 3.21) отримані результати (рисунок 3.22) також вносимо в таблицю. А також запускаємо розпізнавання за допомогою OCR рушій PaddleOCR (рисунок 3.23) та отримані результати (рисунок 3.24) показують, що виявлено 185 слова з впевненістю 99,1%.



Galaxies consist of stars, planets, and vast clouds of gas and dust, all bound together by gravity. The largest contain trillions of stars and can be more than a million light-years across. The smallest can contain a few thousand stars and span just a few hundred light-years. Most large galaxies have supermassive black holes at their centers, some with billions of times the Sun's mass.

Galaxies come in a variety of shapes, mostly spirals and ellipticals, as well as those with less orderly appearances, usually dubbed irregular.

Most galaxies are between 10 billion and 13.6 billion years old. Some are almost as old as the universe itself, which formed around 13.8 billion years ago. Astronomers think the youngest known galaxy formed approximately 500 million years ago.

Galaxies can organize into groups of about 100 or fewer members held together by their mutual gravity. Larger structures, called clusters, may contain thousands of galaxies. Groups and clusters can be arranged in superclusters, which are not gravitationally bound. Superclusters, empty voids, “walls” of galaxies, and other large-scale structures make up the cosmic web of matter in the universe.

Рисунок 3.20 — Оригінал файлу з розширенням png

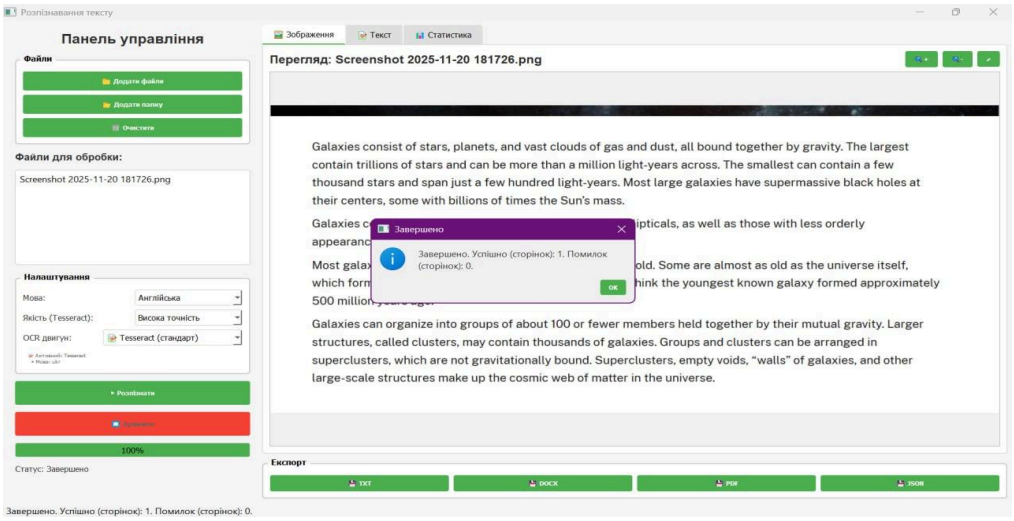


Рисунок 3.21 — Виявлення тексту на фото за допомогою Tesseract

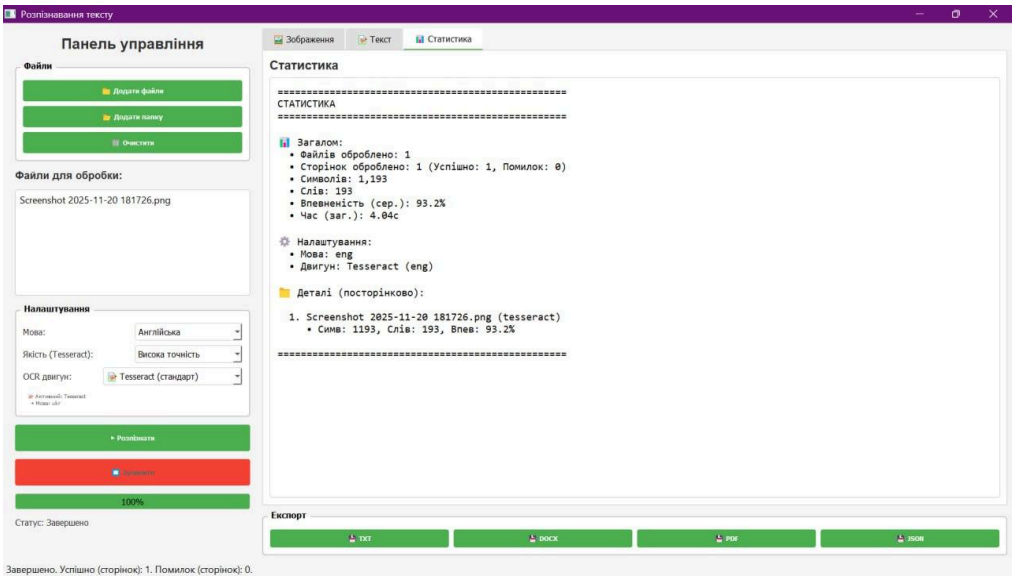


Рисунок 3.22 — Результат роботи Tesseract

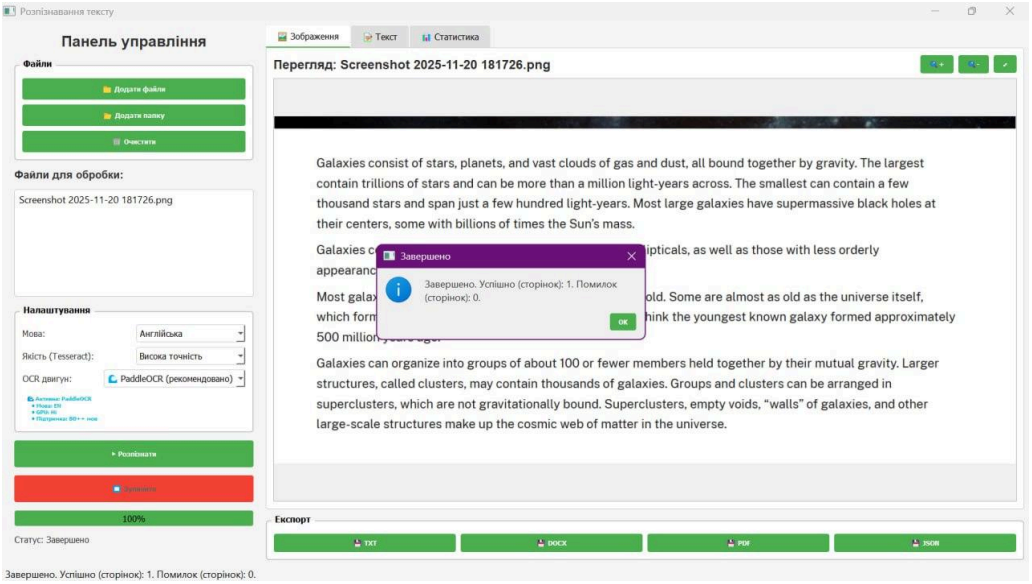


Рисунок 3.23 — Виявлення тексту на фото за допомогою PaddleOCR

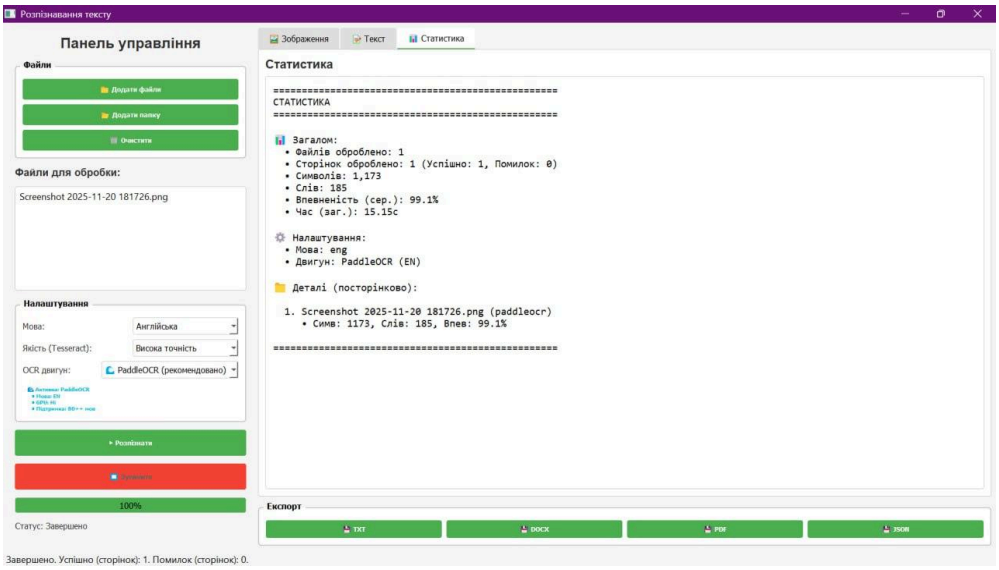


Рисунок 3.24 — Результат роботи PaddleOCR

Також протестуємо розпізнавання на двох pdf-файлах багатосторінкових на рушії PaddleOCR (рисунок 3.25 та рисунок 3.26).

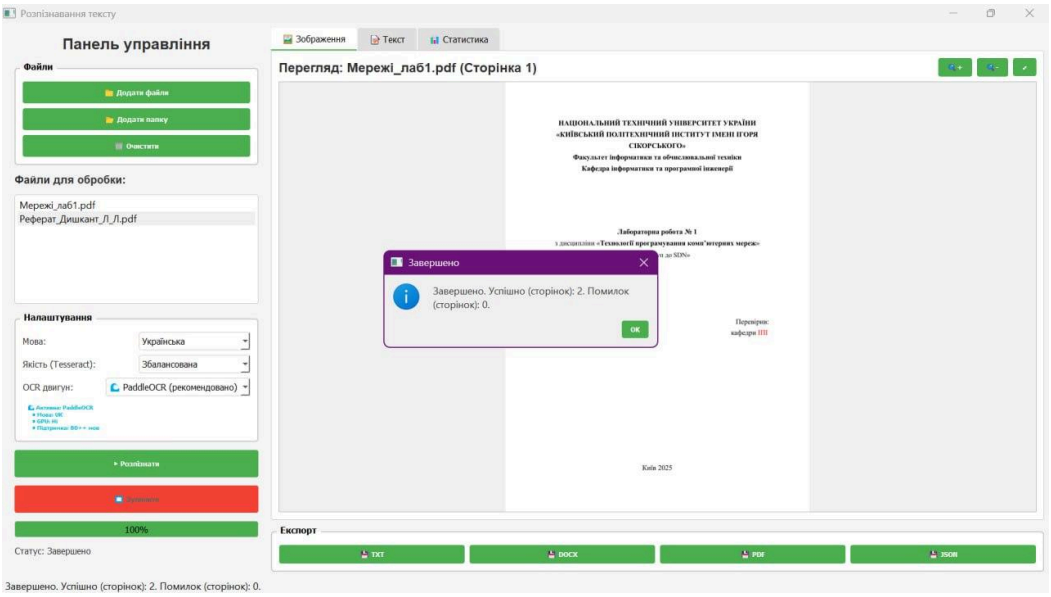


Рисунок 3.25 — Виявлення та розпізнавання тексту в pdf з двох файлів PaddleOCR

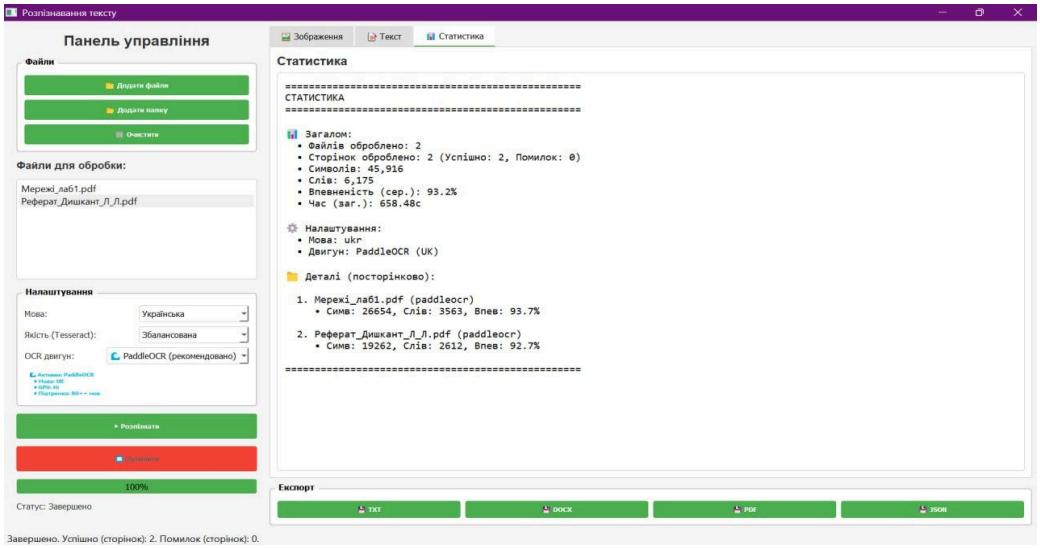


Рисунок 3.26 — Результат розпізнавання ядром PaddleOCR двох pdf-файлів

Для порівняння проведемо тестування на десктопних аналогах OmniPage та Adobe використовуючи ці ж самі файли. Першим завантажимо фото з розширеннням png з рисунку 3.20 та збережемо у текстовий файл і відкриємо у Google docx (рисунок 3.27).

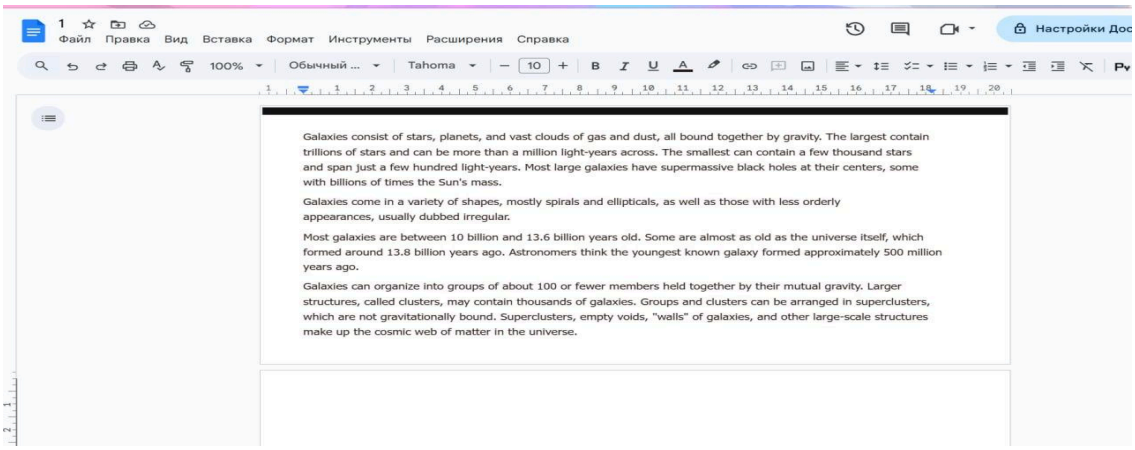
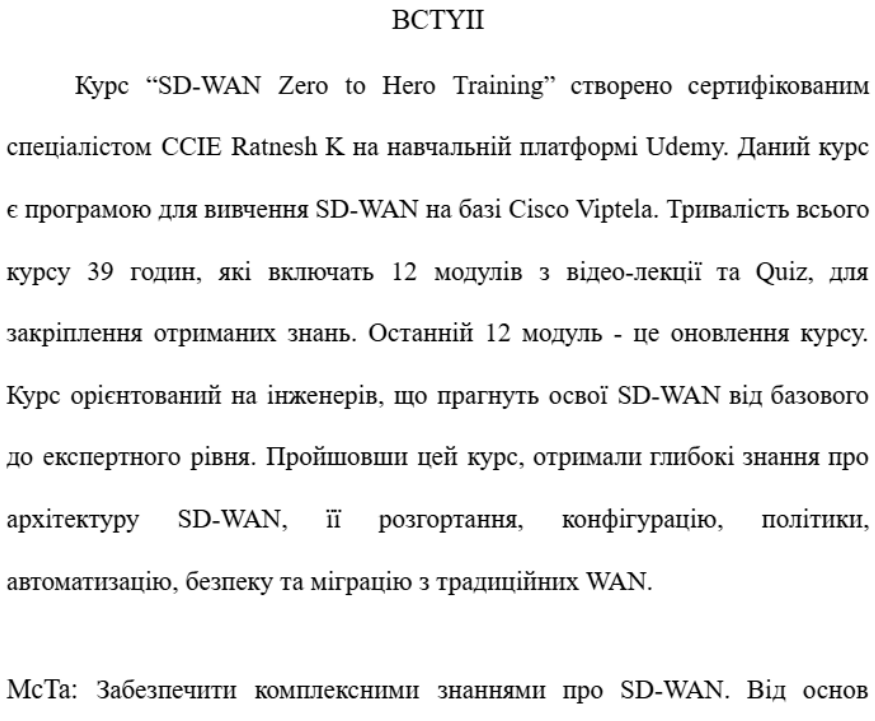


Рисунок 3.27 — Результат конвертації в текстовий документ ПЗ OmniPage



ВСТУП

Курс “SD-WAN Zero to Hero Training” створено сертифікованим спеціалістом CCIE Ratnesh K на навчальній платформі Udemy. Даний курс є програмою для вивчення SD-WAN на базі Cisco Viptela. Тривалість всього курсу 39 годин, які включать 12 модулів з відео-лекції та Quiz, для закріплення отриманих знань. Останній 12 модуль - це оновлення курсу. Курс орієнтований на інженерів, що прагнуть освої SD-WAN від базового до експертного рівня. Пройшовши цей курс, отримали глибокі знання про архітектуру SD-WAN, її розгортання, конфігурацію, політики, автоматизацію, безпеку та міграцію з традиційних WAN.

МсТа: Забезпечити комплексними знаннями про SD-WAN. Від основ

Рисунок 3.28 — Результат конвертації другої сторінки багатосторінкового pdf-файла

Як бачимо з рисунку 3.27 програмне забезпечення порушує розміри текстового документа та передає розміри відповідно фото, які не змінюються. А також при конвертації просто зберігає фото замість того, щоб витягнути та повернути текст з фото. З експерименту з конвертацією pdf-файлу програма справляється з високою точністю, але схожі літери, що є в англійській та українській мові не розпізнає, а замінює (рисунок 3.28). І саме програмне забезпечення дуже незручне. З мінусів можна також виділити: відсутність української мови в меню; не зручна панель керування, тобто користувачу важко буде розібратися як правильно користуватися. Багато зайвих маніпуляцій. Наприклад, щоб завантажити файл спочатку потрібно взагалі обрати тип схожий до вашого документа, наступним файл в який конвертуватимете і третім кроком куди зберігатимете (рисунок 3.29). Після чого

натискаєте на панель з обраним і тільки після з’являється можливість завантажити файл і йде на конвертацію та збереження. Останнім ви повинні використати інше програмне забезпечення, щоб відкрити та вже в ньому працювати, таке як office word.

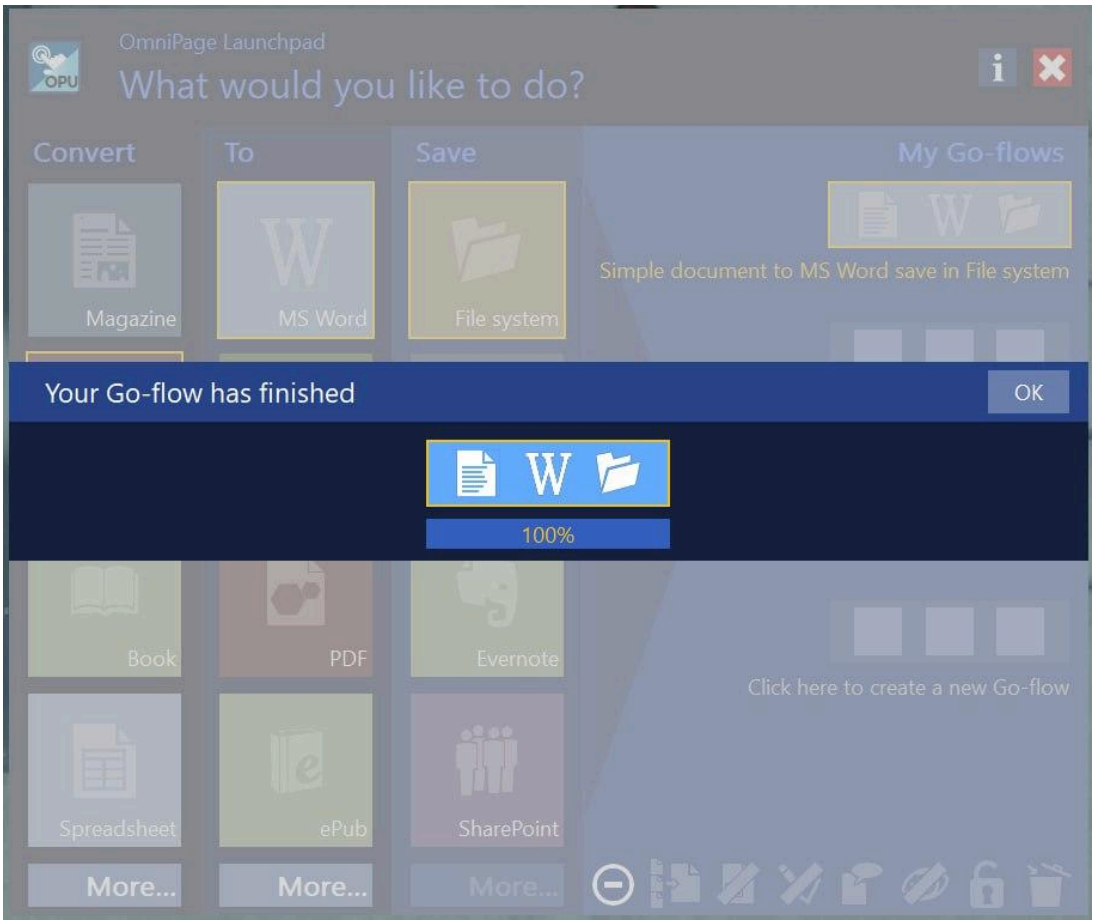


Рисунок 3.29 — Конвертація в ПЗ OmniPage

Наступним протестуємо Acrobat, з фото конвертувати і взагалі завантажувати зображення програма не змогла, тобто вона працює виключно з pdf-документами.

Тому проведемо експерименти тільки на даних файлах. Завантажили файл “Реферат_Дишкант_Л_Л” (рисунок 3.30).

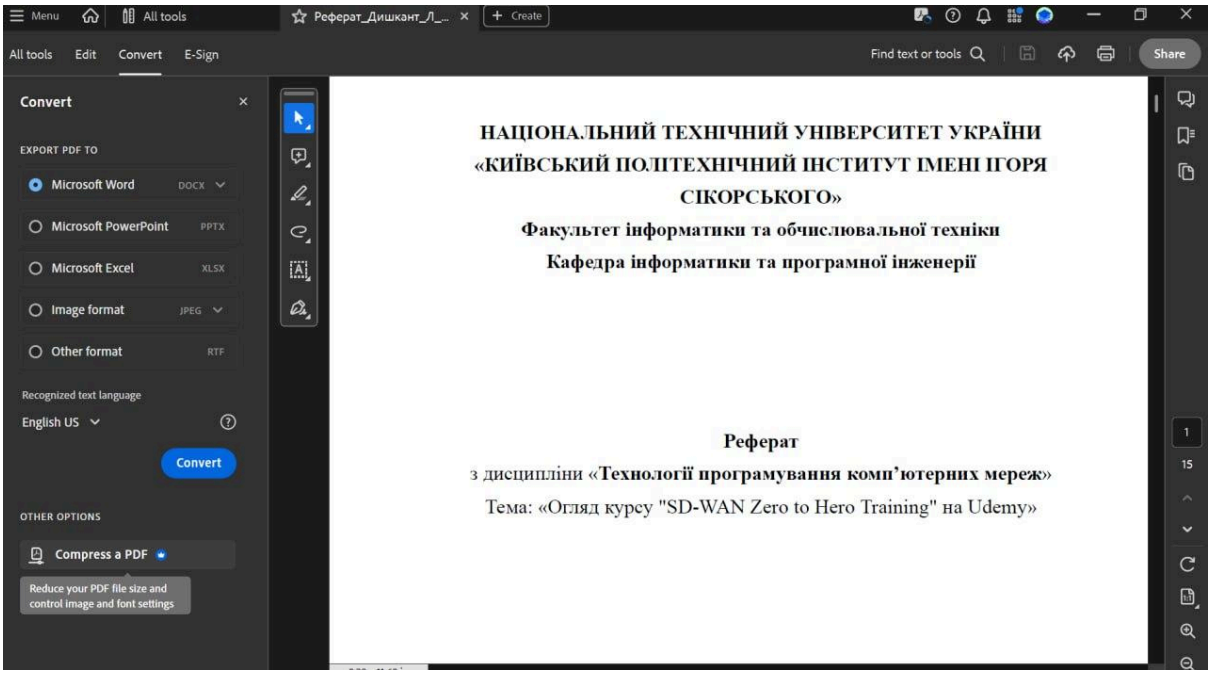


Рисунок 3.30 — Завантаження pdf-файла в Acrobat

Так як використовували пробну версію програмного забезпечення, функції які вказані: конвертація в текстові документи, таблиці чи редагування pdf-файлу завантажуються як розширення і у наданій пробній версії не підвантажилися та протестувати конвертацію та редагування взагалі не вдалося.

3.6 Оцінка ефективності запропонованого рішення

Для того щоб оцінити реалізоване програмне забезпечення, всі отримані результати під час експериментів потрібно внести до таблиці та проаналізувати дані (таблиця 3.1).

Таблиця 3.1 — Оцінка експериментів

Назва рішення	фото	результат % / час	pdf	результат % / час	pdf багатосторінковий	результат % / час
Запропоноване рішення	+	**Tesseract** 93,2% / 4,04 сек	+	**Tesseract** 92,7% / 20,05 сек	+	**Tesseract** 92% в середньому / 94,3 сек
		PaddleOCR 99,9% / 15,15 сек		**PaddleOCR** 98, 8% / 70,04 сек		**PaddleOCR** 93,7% / 458,48 сек
OmniPage	+	повернув фото в текстовий документ та змінив розміри листів відповідно розмірів оригіналу фото	+	98,6% / 40,12 сек	+	92,3% / 212,02 сек
Acrobat	-	-	+	не завантажилося розширення протестувати не вдалося	+	не завантажилося розширення протестувати не вдалося

Проаналізувавши отримані дані результатів експериментів бачимо, що запропоноване рішення має переваги в виявлені та розпізнавання тексту з зображення в досить високому відсотку довіри на обох рушіях. В конвертації

pdf-файлів рушій Tesseract поступається рушію PaddleOCR та існуючим десктопним рішенням, але не сильно. Використовуючи рушій PaddleOCR запропоноване рішення виявляє та розпізнає pdf на 0,2% краще ніж OmniPage, але витрачає більше часу і для пакетної обробки на 1,5% краще та 246,46 секунд більше часу витрачає.

Висновки до розділу

У розділі здійснено повний цикл розробки програмного забезпечення десктопного для виявлення та розпізнавання тексту в не текстових документах, а саме з зображень та pdf-файлів, що повністю відповідає поставленим цілям.

На етапі формування вимог визначено функціональні та нефункціональні вимоги, а також окреслено обмеження поточної версії, які стануть основою для подальшого розвитку. Обрано технологічний стек: Python, PyQt5, OpenCV, Tesseract, PaddleOCR, завдяки чому забезпечено нативний інтерфейс, кросплатформеність та можливість подальшого розширення. Розроблений графічний інтерфейс користувача відповідає принципам мінімалізму та інтуїтивності.

Проведено експериментальні дослідження на реальних документах українською та англійською мовами. Зроблено порівняння з комерційними аналогами, проаналізовано та отримано результати, які показали високу ефективність запропонованого рішення. В порівнянні з аналогами виявлено конкурентні та в окремих сценаріях кращі показники точності, а також значно простіший та зрозуміліший інтерфейс, відсутність залежності від хмарних сервісів.

4 МАРКЕТИНГОВИЙ АНАЛІЗ СТАРТАП–ПРОЄКТУ

4.1 Опис ідеї проекту

Проєкт являє собою програмне забезпечення для десктопних систем, яке призначене для виявлення та розпізнавання тексту в документах нетекстового формату (зображення, скановані документи з розширенням pdf) з використанням гібридних нейромережевих архітектур.

Таблиця 4.1 — Опис ідеї стартап–проєкту

Зміст ідеї	Напрямки застосування	Вигоди для користучача
Розробка десктопного додатка для автоматизованого перетворення сканованих документів та фотографій у редагований текст з використанням комбінації Tesseract та PaddleOCR	Електронний документообіг підприємств	1) Безпека даних (обробка локально). 2) Економія часу на ручний набір тексту. 3) Відсутність щомісячної абонплати. 4) Висока точність розпізнавання завдяки нейронним моделям.
	Оцифрування архівних та бібліотечних фондів	
	Освітня та наукова діяльність	

Таблиця 4.2 — Опис ідеї стартап–проєкту

№	Техніко—економічні характери—тики ідеї	Продукція конкурентів				W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проєкт	ABBYY FineReader	Adobe Acrobat	OmniPage			
1	Вартість ліцензії	S	W	W	W			+
2	Точність розпізнавання	S	S	N (Med)	S			+

Закінчення таблиці 4.2

3	Конфіденційність	S	S	N	S			+
4	Підтримка складних макетів	N	N	N	S		+	
5	Системні вимоги	N	W	W	W		+	
6	Кросплатформеність	S	W	W	W			+

4.2 Технологічний аудит ідеї проєкту

Таблиця 4.3 — Технологічна здійсненність ідеї проєкту

№	Ідея проєкту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Попередня обробка зображення	OpenCV, Numpy (Deskew, Binarization, Denoising)	Так	Вільна
2	Розпізнавання тексту (OCR)	Tesseract (LSTM), PaddleOCR (CRNN)	Так	Вільна
3	Інтерфейс користувача	Python, PyQt5	Так	Вільна
4	Експорт результатів	Python libraries (docx, fpdf)	Так	Вільна
Обрана технологія реалізації ідеї проєкту:Python + PyQt5 + Tesseract/PaddleOCR + OpenCV				

Висновок: технологічна реалізація продукту — можлива, вибрані технології є доступними та мають відкритий вихідний код і сумісні між собою.

4.3 Аналіз ринкових можливостей запуску стартап–проєкту

Таблиця 4.4 — Попередня характеристика потенційного ринку

№	Показники стану ринку	Характеристика
1	Кількість головних гравців, од	3-5 основних (ABBYY, Adobe, Kofax)
2	Загальний обсяг продаж, грн./ум.од	
3	Динаміка ринку	Зростаюча, через цифрову трансформацію

Закінчення таблиці 4.4

4	Наявність обмежень для входу	Середня (потреба у високій технічній експертизі та алгоритмах).
5	Специфічні вимоги до стандартизації та сертифікації	Відсутні жорсткі вимоги, крім захисту персональних даних.
6	Середня норма рентабельності в галузі або по ринку, %	Висока

Висновок: враховуючи кількість головних гравців по ринку, зростаючу динаміку ринку, невелику кількість конкурентів та середню норму рентабельності можна зробити висновок, що на даний момент, ринок для входження стартап–продукту є привабливим для входу з більш доступним та адаптивним продуктом.

Таблиця 4.5 — Характеристика потенційних клієнтів стартап–проєкту

№	Потреба, що формує ринок	Цільова аудиторія	Відмінності у поведінці цільових груп клієнтів	Вимоги споживачів до товару
1	Автоматизація введення даних з паперу	Державні підприємства, малий та середній бізнес	Чутливі до ціни, потребують пакетної обробки, захист витоку конфіденційної інформації	Швидкість, точність, ціна, безпека даних, пакетна обробка. Підтримка багатомовності, експорт у Word. Локальна робота без інтернету, точність розпізнавання.
2	Оцифрування навчальних матеріалів	Студенти, викладачі	Недорогі або безкоштовні рішення	
3	Конфіденційна обробка документів	Юристи, нотаріуси	Високі вимоги до безпеки даних	

Таблиця 4.6. — Фактори загроз

№	Фактор	Зміст загрози	Можлива реакція компанії
1	Конкуренти	Наявність конкурентів котрі надають схожі рішення	Зменшення ціни на поставлену послугу; Розробка унікальних характеристик товару, адаптація під конкретні потреби; Оновлення версії; Надання ліцензій на обслуговування
2	Кошти на розробку та підтримку продукту	Закінчення грошей та недостатнє фінансування	Залучення додаткових інвесторів, мотивація роботи на перспективу; Ітеративна розробка продукту задля покрокового виведення продукту на ринок та отримання відповіді користувачів
3	Вихід аналогу	Вихід аналогу даного товару може призвести до знецінення та безідейності даного товару	Вихід товару на ринок в коротші строки з не повною, але достатньою, функціональністю для зацікавлення усіх цільових аудиторій; Проведення рекламної компанії

Таблиця 4.7. — Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1	Новий продукт	Вихід на ринок, Зменшення монополії, Надання нових рішень у сфері	Розробка нової функціональності; Вихід нової продукції на ринок; Надання різноманітних типів ліцензій в залежності від потреб користувача \ замовника.
2	Вихід аналогу	Надати продукт з певними характеристиками та можливостями що відсутні у компаній конкурентів	Аналіз ринку та користувачів задля задоволення їх потреб та надання функціональності у найкоротші строки за ціну, котра є дешевшою ніж у продуктів–замінників.
3	Зворотній зв’язок від користувачів	Можливість отримання необхідної інформації для вдосконалення продукту	Наявність вхідних даних та реакція на них з боку команди розробників задля задоволення потреб та бажань кінцевих користувачів системи розпізнавання тексту.

Закінчення таблиці 4.7

4	Грошова винагорода за рекламу	При достатньому попиту на систему розпізнавання тексту можлива комерціалізація продукту на основі реклами задля отримання грошової винагороди для подальшого розвиту продукту та оплати заробітної плати працівникам	Точкова комерціалізація продукту; Введення реклами; Ведення додаткових коштів у проєкт задля його подальшого розвитку.

Таблиця 4.8 — Ступеневий аналіз конкуренції на ринку

№	Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1	Тип конкуренції: монополістична	Товар від кожної компанії на ринку, являється недосконалим замінником товару, реалізованого іншими фірмами; На ринку є умови для входу та виходу; Ціна корелює між суперниками;	Розробка продукту з характеристиками, які покривають сфери вживання що не покривають інші товари–замінники; Кореляція цін у відповідності до товарів замінників; Різни типи ліцензій.
2	Рівень конкурентної боротьби: світовий	Всі продукти замінники розроблялись інтернаціональними командами з різних куточків світу, продукти не належать до певної держави, а належать команді розробників	Вихід на ринок збуту продукту з клієнто–необхідною функціональністю; Налагодження маркетингу на основних Інтернет ресурсах задля охоплення великої кількості потенційних користувачів; Надання бета–версій продукту.

Закінчення таблиці 4.8

3	Галузева ознака: внутрішньогалузева	Даний тип продукту може використовуватися тільки у сфері розробки IT додатків \ продуктів	Надання зручного, інтуїтивно зрозумілого інтерфейсу; Підтримка всім відомих методів взаємодії з середовищем розробки; Наявність документації та онлайн підтримки.
4	Конкуренція за видами товарів: товарно–видова	Дана конкуренція — конкуренція між товарами одного виду.	Впровадження функціональності яка відсутня у товарів–замінників; Спрощення інтерфейсів; Надання підтримки.
5	Характер конкурентних переваг: цінова та не цінова	Цінові переваги — точкова комерціалізація; Не цінова — надання функціональності, що відсутня у товарах–замінниках.	Надання платних ліцензій лише на критично важливу функціональність для клієнта з певним строком підтримки, що зазначена у відповідній ліцензії; Впровадження унікальної функціональності.
6	За інтенсивністю: марочна	Наявність унікального знаку що відрізняє даний продукт від продуктів–замінників	Впровадження власної назви та власного знаку.

Таблиця 4.9 — Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари–замінники
	ABBYY FineReader, OmniPage	Розробники мобільних сканерів	Спільноти Open Source	Держустанови, бізнес, фізичні особи	Онлайн-конвертери, ручний набір
Висновки	Сильні позиції, висока ціна	Можуть вийти на десктопний ринок	Низька загроза, технології відкриті	Висока чутливість до ціни та якості	Низька безпека

Проаналізувавши можливості роботи на ринку з огляду на конкурентну ситуацію можна зробити висновок: оскільки кожний з існуючих продуктів не впливає у великій мірі на поточну ситуацію на ринку в цілому, кожний з існуючих продуктів має свою специфічну сферу використання та свої позитивні та негативні

сторони щодо рішення певних типів задач, то робота та вихід на даний ринок є можливою і реалізованою задачею [23].

Для виходу на ринок продукт повинен мати функціонал що відсутній у продуктів–аналогів, повинен задовольняти потреби користувачів, мати необхідний та достатній функціонал з конфігурування, підтримку зі сторони розробників та можливість розробки спеціального функціоналу за відповідною ліцензією.

Таблиця 4.10 — Обґрунтування факторів конкурентоспроможності

№	Фактор конкурентоспроможності	Обґрунтування
1	Точність розпізнавання	Використання гібридного підходу (Tesseract + PaddleOCR) та атоматичний failback
2	Ціна	Використання Open Source дозволяє знизити вартість кінцевого продукту
3	Конфіденційність	Локальна обробка файлів без передачі на сервери
4	Зручність інтерфейсу	Сучасний GUI на PyQt5, інтуїтивне управління
5	Підтримка мов	Підтримка української мови, англійської та змішаної
6	Налаштування під специфічні документи	Оновлення версії з інтеграцією навченої моделі на відповідних документах

Таблиця 4.11 — Порівняльний аналіз сильних та слабких сторін системи розпізнавання тексту

№	Фактор конкурентоспроможності	Бали 1—20	Рейтинг товарів—конкурентів у порівнянні з запропонованим						
			—3	—2	—1	0	+1	+2	+3
1	Ціна продукту	20	Omni Page	Adobe					
2	Якість розпізнавання	18			Adobe		Omni Page	ABBYY	
3	Швидкодія	15						OmniPage	

Закінчення таблиці 4.11

4	Безпека даних	19				OmniPage			
5	Зручність інтерфейсу	12		OmniPage		Adobe			
6	Підтримка мов	10					ABBYY		

Таблиця 4.12 — SWOT аналіз стартап–проєкту

Сильні сторони (S): – Низька собівартість розробки (Python stack); – Висока безпека даних (Local processing); – Гібридна точність (Tesseract +PaddleOCR); – Кросплатформеність.	Слабкі сторони (W): – Відсутність впізнаваного бренду; – Менший функціонал редагування pdf порівняно з Acrobat; – Відсутність підтримки рукописного тексту в поточній версії.
Можливості (O): – Інтеграція в державний сектор; – Додавання модуля розпізнавання рукописного тексту; – Інтегрування попередньонавченої моделі для роботи з спецструктурою документа.	Загрози (T): – Зниження цін конкурентами; – Зміна стандартів форматів файлів.

Таблиця 4.13 — Альтернативи ринкового впровадження стартап–проєкту

№	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Безкоштовне надання певного функціоналу у користування споживачам на обмежений термін	Головний ресурс – люди, даний ресурс –наявний	2–3 місяці
2	Реклама	Залучення власних коштів для реклами товару	1–2 місяці
3	Написання статей та опис товару на відомих ресурсах	Головний ресурс – час, даний ресурс – наявний	2–3 тижні
4	Презентація товару на хакатонах й інших ІТ заходах	Ресурс – час та гроші для участі, наявні	1–3 місяці

4.4 Розроблення ринкової стратегії проєкту

Таблиця 4.14 — Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Державні підприємства	Висока (потребують дешевого інструмента)	Високий	Середня	Висока

Закінчення таблиці 4.14

2	Офісні працівники	Середня (звикли до піратського чи корпоративного ПЗ)	Середній	Висока	Середня
Які цільові групи обрано: Державні підприємства та малий бізнес, оскільки вони найбільш чутливі до ціни та потребують простого рішення для базових задач OCR.					

Відповідно до проведеного аналізу можна зробити висновок, що підходящою цільовою групою для розповсюдження даного програмного продукту є працівники державної сфери, ІТ компанії в цілому та будь–які підприємства котрі використовують програмні продукти побудовані на мові програмування Python. Відповідно до стратегії охоплення ринку збуту товару обрано стратегію масового маркетингу, оскільки для підприємств, офісних працівників та ІТ компаній у цілому надається стандартизований продукт з можливістю розширення функціональності за домовленістю (відповідно до ліцензії).

Таблиця 4.15 — Визначення базової стратегії розвитку

Обрана альтернатива розвитку проєкту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
Надання функціональності що відсутня у товарів–замінників, підтримка клієнтів	Проведення реклами, освітлення унікальної функціональності через інтернет ресурси та інші канали, контакт напряму з споживачами; формування лояльності і прихильності споживачів (ніша офлайн безпечного OCR)	Зниження ступеню замінності товару; Прихильність клієнтів; Відмітні властивості товару; Відмітні характеристики товару; Безпека даних;	Стратегія диференціації (за критерієм безпеки)

Таблиця 4.16 — Визначення базової стратегії конкурентної поведінки

Чи є проєкт «першопрохідцем» на ринку	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, які?	Стратегія конкурентної поведінки
Ні, оскільки є товари–замінники, але дані товари замінники не мають деякого необхідного функціоналу	Так, ціль компанії знайти нових споживачів та, частково, забрати існуючих у конкурентів задля задоволення потреб останніх	Компанія частково копіює характеристики товару конкурента, основна ціль компанії розробка нового унікального функціоналу	Стратегія заняття конкурентної ніші

Таблиця 4.17 — Визначення стратегії позиціонування

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап–проєкту	Вибір асоціацій, які мають сформувати комплексну позицію власного проєкту
1	Безпека	Фокусування	Офлайн обробка	“Приватність понад усе”
2	Точність	Диференціація	Гібридні OCR	“Розумний OCR”
3	Ціна	Лідерство за витрати	Доступність	“Доступний інструмент”

Відповідно до проведеного аналізу можна зробити висновок, що стартап–компанія вибирає як базову стратегію розвитку — стратегію диференціації,

як базову стратегію конкурентної поведінки — стратегію заняття конкурентної ніші “безпечного та доступного OCR”.

4.5 Розроблення маркетингової програми стартап–проєкту

Таблиця 4.18 — Визначення ключових переваг концепції потенційного товару

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1	Робота з фото	Якісний текст навіть з неякісних сканів	Потужна попередня обробка
2	Розпізнавання тексту	Швидке отримання редагованого тексту	Автоматичний вибір кращого рушія (Tesseract/Paddle)
3	Редагування результату	Можливість правити безпосередньо в програмі	Вбудований редактор з порівнянням до оригіналу

Таблиця 4.19 — Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
1. Товар за задумом	Програма для автоматичного перетворення зображень у текст		
2. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх/Тл/Е/Ор
	Висока точність розпізнавання	Нм	Тх
	Конфіденційність даних (локальна обробка файлів без передачі в інтернет чи хмару)	Нм	Ор

Закінчення таблиці 4.19

2. Товар у реальному виконанні	Зручний графічний інтерфейс (GUI на PyQt5, інтуїтивне управління, drag-and-drop)	Нм	Е
	Багатоформатність (PDF, PNG, JPEG, TIFF)	Нм	Тх
	Попередня обробка зображення(видалення шумів, вирівнювання нахилу)	Нм	Тл
	Кросплатформеність	Нм	Тх
	Експорт результатів	Нм	Тх
3. Товар із підкріпленням	До продажу: наявна повна документація, акції на придбання декількох ліцензій, знижки для певних сегментів на покупку товару		
	Після продажу: додаткова підтримка спеціалістів налаштування, підтримка з боку розробника, регулярні оновлення двигунів розпізнавання.		
За рахунок чого потенційний товар буде захищено від копіювання: захист інтелектуальної власності, патент			

Таблиця 4.20 — Визначення меж встановлення ціни

Рівень цін на товари–замінники	Рівень цін на товари–аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
0 грн (онлайн сервіси з рекламою)	100 – 200 у.о.	Середній/Нижче середнього	Нижня: Безкоштовно. Верхня: 10–20 у.о.

Таблиця 4.21 — Формування системи збуту

Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
Шукають ПЗ в інтернеті, читають відгуки	Надання демо-версії, зручне завантаження	Однорівневий (Розробник - Споживач)	Продаж через власний сайт та магазин додатків

Таблиця 4.22 — Концепція маркетингових комунікацій

№	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Пошук вирішення проблеми	Пошукові системи, соціальні мережі та форуми	Швидко, Якісно, Безпечно	Завантаження трафіку на сайт завантаження	Фокус на економії часу та безпеці даних. Патріотичний підхід (підтримка національного продукту як альтернативи конкурентів).
2	Довіра до рекомендацій		Український продукт	Формування довіри та спільноти	

Як результат було створено ринкову (маркетингову) програму, що включає в себе визначення ключових переваг концепції потенційного товару, опис моделі товару, визначення меж встановлення ціни, формування системи збуту та концепцію маркетингових комунікацій.

Висновки до розділу

В четвертому розділі описано стратегії та підходи з розроблення стартап–проєкту, проведено комплексний аналіз. Визначено, що ринок має потенціал для нового гравця, який запропонує доступне, безпечне та точне рішення для українського користувача. Обрано стратегію диференціації, орієнтовану на малий бізнес та державний сектор. Визначено цінову політику, яка є значно привабливішою за конкурентів, та канали просування через інтернет–ресурси. На основі проведеного маркетингового аналізу стартап–проєкту можна сформулювати наступні ключові положення: Проєкт має високий потенціал завдяки використанню гібридних нейромережевих архітектур (Tesseract + PaddleOCR), що забезпечує точність на рівні світових лідерів (ABBYY, Adobe), але за значно нижчою вартістю. Головною технічною перевагою є локальна обробка даних, що вирішує критичне питання конфіденційності для державних та юридичних установ. Визначено, що ринок OCR–рішень перебуває у фазі зростання через глобальну цифровізацію. Попри наявність потужних гравців, існує вільна ніша для доступного десктопного ПЗ, орієнтованого на малий бізнес та державний сектор України, де клієнти чутливі до ціни та вимог безпеки. Обрано **стратегію диференціації** за критерієм безпеки ("приватність понад усе") та **стратегію заняття конкурентної ніші**. Це дозволить уникнути прямої цінової війни з гігантами ринку, фокусуючись на специфічних потребах локальних користувачів та підтримці української мови. Маркетинговий план передбачає поступове впровадження продукту через прямі канали збуту та онлайн–платформи з акцентом на контент–маркетинг (статті, огляди) та участь у профільних ІТ–заходах для формування довіри до бренду.

ВИСНОВКИ

Проведене дослідження та розробка програмного засобу для виявлення й розпізнавання тексту в документах не текстового формату (скановані документи, фотографії, PDF–зображення) повністю досягли поставленої мети – створення ефективного, доступного та кросплатформенного десктопного рішення з підвищеною точністю розпізнавання порівняно з існуючими аналогами.

Основні результати роботи:

а) Виконано ґрунтовний аналіз предметної області та сучасних OCR–технологій, виявлено ключові виклики (різноманітність макетів і шрифтів, низька якість зображень, геометричні спотворення, багатомовність) та проаналізовано комерційні рішення (ABBYY FineReader, Adobe Acrobat, OmniPage) й відкриті наукові підходи;

б) Розроблено комплексний метод підвищення точності розпізнавання, що включає:

1) адаптивну попередню обробку зображень (корекція нахилу й перспективи, видалення шуму, CLAHE, адаптивна бінаризація Otsu/Sauvola, корекція освітлення);
2) гібридну стратегію використання двох найсильніших відкритих OCR–рушіїв: Tesseract 4.0 (LSTM) як основний та PaddleOCR (CRNN–архітектура) як резервний з автоматичним fallback за рівнем впевненості;
3) постобробку результатів (очищення пробілів і пунктуації, автокорекція гомогліфів та типових помилок).

в) Створено модульне кросплатформенне програмне забезпечення на Python 3 з графічним інтерфейсом PyQt5, яке забезпечує:

1) пакетну обробку великих обсягів документів;
2) підтримку форматів JPEG, PNG, TIFF, багатосторінкових PDF;
3) інтуїтивний інтерфейс з попереднім переглядом, редагуванням розпізнаного тексту та статистикою;
4) експорт у TXT, DOCX, PDF, JSON;
5) багатопотокову обробку без блокування інтерфейсу.

г) Проведено серію експериментальних досліджень на реальних україномовних та англомовних документах різної складності. Отримані результати підтвердили високу ефективність розробленого рішення:

1) середня впевненість розпізнавання 93–99 % при використанні PaddleOCR;
2) на складних багатосторінкових PDF та фотографіях точність перевищує або дорівнює комерційним аналогам OmniPage й Adobe Acrobat при значно нижчій вартості та повній незалежності від хмарних сервісів;
3) час обробки одного багатосторінкового документа становить 50–90 секунд на звичайному ноутбуці без GPU.

Запропоноване рішення є повністю безпечним, тобто зберігає конфіденційність даних, не потребує інтернет–з’єднання та ліцензій, що робить його особливо цінним для освітніх і наукових установ, малого та середнього бізнесу, державних архівів та індивідуальних користувачів України.

Подальший розвиток системи передбачає:

— інтеграцію власної гібридної нейронної моделі CRNN + BiLSTM + Attention;

— впровадження детекції текстових блоків на основі YOLOv8/EfficientDet для ще точнішої локалізації;

— підтримку розпізнавання рукописного тексту та математичних формул;

— розширення до веб та мобільних версій.

Розроблений програмний засіб є актуальним, конкурентоспроможним і практично значущим продуктом, який успішно розв’язує задачу якісного та доступного перетворення документів не текстового формату у структурований, редагований текст і може бути рекомендований до широкого впровадження.

Проведено комплексний аналіз описано стратегії та підходи з розроблення стартап–проєкту. Визначено, що ринок має потенціал для нового гравця, який запропонує доступне, безпечне та точне рішення для українського користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Text Detection Design Based on Deep Neural Network Xu Wei, Ding Manna, Wang Weihang_ICASIT 2020: Proceedings of the 2020 International Conference on Aviation Safety and Information Technology Pages 638 – 642

2) An ML–aided Approach to Automatically Generate Schematic Symbols in PCB EDA Tools. Shiyu Gao, Keni Qiu MLCAD '24: Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD Article No.: 6, Pages 1–7 [Електронний ресурс] // Режим доступу: https://doi.org/10.1145/3670474.3685944

3) Implementing the Tesseract Method for Information Extraction from Images Abrar AI Sayem, Asiful Islam Chowdhury, Shahriar Hossain Shojol, ICCTA '23: Proceedings of the 2023 9th International Conference on Computer Technology Applications Pages 97–101 [Електронний ресурс] // Режим доступу: https://doi.org/10.1145/3605423.3605431

4) Automatic identification system of aircraft cockpit indicators based on machine vision Jiaqing Yao, Renwen Chen, Yijun Huang AISS '22: Proceedings of the 4th International Conference on Advanced Information Science and SystemArticle No.: 63, Pages 1–9 [Електронний ресурс] // Режим доступу: https://doi.org/10.1145/3573834.3574549

5) Target Text Location for Vehicle Inspection Based on Template Matching ZiWei Wang, Xinfeng Zhang, Xun Zhao, Hui Li ICCPR '20: Proceedings of the 2020 9th International Conference on Computing and Pattern RecognitionPages 344–349 [Електронний ресурс] // Режим доступу: https://doi.org/10.1145/3436369.3436475

6) Object detection for graphical user interface: old fashioned or deep learning or a combination? Jieshan Chen, Mulong Xie, Zhenchang Xing, Chunyang Chen, Xiwei Xu ESEC/FSE 2020: Proceedings of the 28th ACM Joint Meeting on European Software

Engineering Conference and Symposium on the Foundations of Software EngineeringPages 1202–1214 [Електронний ресурс] // Режим доступу: https://doi.org/10.1145/3368089.3409691

7) Tag Information Recognition Approaches and Algorithms for Cross–Border Products Checking Dunsheng Chen, Yinsheng Li, Xu Liang, ICCSE'19: Proceedings of the 4th International Conference on Crowd Science and EngineeringPages 58–62 [Електронний ресурс] // Режим доступу: https://doi.org/10.1145/3371238.3371248

8) Deep Learning–based End-to-End Address Recognition Solution on Chinese Courier Order Forms Jiayi Zhang, Yue Liu, AI2A '23: Proceedings of the 2023 3rd International Conference on Artificial Intelligence, Automation and Algorithms Pages 158 – 163 [Електронний ресурс] // Режим доступу: https://doi.org/10.1145/3611450.3611473

9) Image Recognition Tools for Blind and Visually Impaired Users: An Emphasis on the Design Considerations Sandra Fernando, Chiemela Ndukwe, Bal Virdee, Ramzi Djemai ACM Transactions on Accessible Computing, Volume 18, Issue 1 Article No.: 1, Pages 1 – 21 [Електронний ресурс] // Режим доступу: https://doi.org/10.1145/370220

10) Scalable and Cost–effective Serverless Architecture for Information Extraction Workflows Dheeraj Chahal, Surya Chaitanya Palepu, Rekha Singhal HiPS '22: Proceedings of the 2nd Workshop on High Performance Serverless Computing Pages 15–23 [Електронний ресурс] // Режим доступу: https://doi.org/10.1145/3526060.3535458

11) A Case Study on Handwritten *Indic* Script Classification: Benchmarking of the Results at Page, Block, Text–line, and Word Levels Pawan Kumar Singh, Ram Sarkar, Ajith Abraham, Mita Nasipuri ACM Transactions on Asian and Low–Resource Language Information Processing (TALLIP), Volume 21, Issue 2 Article No.: 32, Pages 1–36 [Електронний ресурс] // Режим доступу: https://doi.org/10.1145/3476102

12) A Proposal of OCR–based User Positioning Method in Indoor Navigation System Using Unity and Smartphone (INSUS) Evianita Dewi Fajrianti, Nobuo Funabiki, Amma Liesvarastranta Haz ICNCC '23: Proceedings of the 2023 12th International Conference on Networks, Communication and Computing Pages 99–105 [Електронний ресурс] // Режим доступу: https://doi.org/10.1145/3638837.3638852

13) Using YOLO Network for Automatic Processing of Finite Automata Images with Application to Bit–Strings Recognition Daniela S. Costa, Carlos A.B. Mello DocEng '23: Proceedings of the ACM Symposium on Document Engineering 2023 Article No.: 15, Pages 1–9 [Електронний ресурс] // Режим доступу: https://doi.org/10.1145/3573128.3604898

14) Deep learning Arabic printed document knowledge extraction Taghreed Alghamdi, Samia Snoussi, Lobna Hsairi ICFNDS '21: Proceedings of the 5th International Conference on Future Networks and Distributed Systems Pages 201–207 [Електронний ресурс] // Режим доступу: https://doi.org/10.1145/3508072.3508103

15) Text Localization in Scientific Figures using Fully Convolutional Neural Networks on Limited Training Data Morten Jessen, Falk Böschen, Ansgar Scherp DocEng '19: Proceedings of the ACM Symposium on Document Engineering 2019 Article No.: 13, Pages 1–10 [Електронний ресурс] // Режим доступу: https://doi.org/10.1145/3342558.3345396

16) An Automatic Location and Recognition Method for Bank Card Number Yuanxue Xin, Pengfei Shi, Song Han ICAI '19: Proceedings of the 2019 International Conference on Robotics, Intelligent Control and Artificial Intelligence Pages 728–732 [Електронний ресурс] // Режим доступу: https://doi.org/10.1145/3366194.3366325

17) Investigation of Faster–RCNN Inception Resnet V2 on Offline Kanji Handwriting Characters Anthony Adole, Eran Edirisinghe, Baihua Li, Chris Bearchell PRIS '20: Proceedings of the 2020 International Conference on Pattern Recognition and

Intelligent Systems Article No.: 18, Pages 1–5 [Електронний ресурс] // Режим доступу: https://doi.org/10.1145/3415048.3416104

18) What Is the Intended Usage Context of This Model? An Exploratory Study of Pre–Trained Models on Various Model Repositories Lina Gong, Jingxuan Zhang, Mingqiang Wei, Haoxiang Zhang, Zhiqiu Huang ACM Transactions on Software Engineering and Methodology (TOSEM), Volume 32, Issue 3 Article No.: 69, Pages 1–57 [Електронний ресурс] // Режим доступу: https://doi.org/10.1145/3569934

19) Handwriting Text Recognition using CNN and RNN R. Sumathy; S. Narayana Swami;T. Pavan Kumar;V. Lakshmi Narasimha;B. Premalatha2023 2nd International Conference on Applied Artificial Intelligence and Computing (ICAAIC) [Електронний ресурс] // Режим доступу: https://ieeexplore.ieee.org/

20) Review of Artificial Intelligence Methods in Handwriting Identification Using CNN–RNN for Textural Features Chandan Kumar Sonkar;Vinod Kumar 2025 International Conference on Cognitive Computing in Engineering, Communications, Sciences and Biomedical Health Informatics (IC3ECSBHI) Year: 2025, Conference Paper, Publisher: IEEE [Електронний ресурс] // Режим доступу: https://ieeexplore.ieee.org/

21) An English Handwriting Evaluation Algorithm Based on CNNs Yingguo Gao;Runze Yu;Xiaohui Duan 2019 International Conference on Artificial Intelligence and Advanced Manufacturing (AIAM) [Електронний ресурс] // Режим доступу:

22) Handwritten Amharic Word Recognition With Additive Attention Mechanism Ruchika Malhotra;Maru Tesfaye Addis IEEE Access 2024 [Електронний ресурс] // Режим доступу: https://ieeexplore.ieee.org/

23) Classification of Text regions in a Document Image by Analyzing the properties of Connected Components Showmik Bhowmik;Ram Sarkar 2020 IEEE Applied Signal Processing Conference (ASPCON) [Електронний ресурс] // Режим доступу:

24) Comparative Analysis of Text Extraction from Color Images using Tesseract and OpenCV AS Revathi;Nishi A Modi 2021 8th International Conference on Computing for Sustainable Global Development (INDIACom) [Електронний ресурс] // Режим доступу: https://ieeexplore.ieee.org/

25) A Novel Model for Recognising Handwritten Devanagari Numerals using Machine Learning Ch. Prathima;Ramprakash Reddy Arava;K Sevitha;G Manikanth;D Vinay;C Surya 2024 4th International Conference on Pervasive Computing and Social Networking (ICPCSN) [Електронний ресурс] // Режим доступу: https://ieeexplore.ieee.org/

26) A Tool for Extracting Text from Scanned Documents and Convert it into Editable Format S. Sukanya;S. Joseph Gladwin;C. Vinoth Kumar 2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN) [Електронний ресурс] // Режим доступу: https://ieeexplore.ieee.org/

27) Read Textual features in Images and convert to Editable form by extended use of Artificial Neural Networks, Deep learning and Maximally Stable Extremal Region … SJ Gladwin, CV Kumar Journal of Physics: Conference Series, 2021 [Електронний ресурс] // Режим доступу: https://iopscience.iop.org

28) Which OCR toolset is good and why: A comparative study P Jain, K Taneja, H Taneja – Kuwait Journal of Science, 2021 — [Електронний ресурс] // Режим доступу: https://journalskuwait.org

29) Text Extraction from Document Images using CNN and LSTM S Shinde, S Bhosle, G Bomble — 2023 International, 2023 [Електронний ресурс] // Режим доступу: https://ieeexplore.ieee.org

30) A novel method for image to text extraction using tesseract–OCR S Kumar Garai, O Paul, U Dey — American Journal of, 2022 — [Електронний ресурс] // Режим доступу: https://ingentaconnect.com

31) Blind–Aid: Empowering the Visually Impaired using Artificial Intelligence and Image Processing TC Mahalingesh, A Karthik – on Smart Systems, 2023 — [Електронний ресурс] // Режим доступу: https://ieeexplore.ieee.org

32) Zayed Encyclopedia as a Database for an AI–Driven Generative Program Serving Jurisprudential, Usul, and Maqasid Rules K Bioud, H Alshari — 2022 [Електронний ресурс] // Режим доступу: https://alwasl.ac.ae

33) PiExtract: An End-to-End Data Extraction Pipeline for Pie–Charts MS Kanroo, HS Kawoosa, J Dhar, P Goyal — International Conference on, 2025 — Springer [Електронний ресурс] // Режим доступу: https://link.springer.com/

34) A Review on Natural Scene Text Understanding for Computer Vision using Machine Learning MTG Prakash, B Tejaswini, C Varshini, C Harathi — 2022 — [Електронний ресурс] // Режим доступу: https://academia.edu

35) Deep learning approaches for information extraction from visually rich documents: datasets, challenges and methods H Gbada, K Kalti, MA Mahjoub — International Journal on Document , 2024 — Springer [Електронний ресурс] // Режим доступу: https://link.springer.com/

36) Text Removal for Trademark Images Based on Self–Prompting Mechanisms and Multi–Scale Texture Aggregation W Zhou, X Wang, B Zhou, L Li — Applied Sciences, 2025 — [Електронний ресурс] // Режим доступу: https://mdpi.com

37) Benchmarking NAS for article separation in historical newspapers N Girdhar, M Coustaty, A Doucet – International Conference on Asian , 2023 — Springer [Електронний ресурс] // Режим доступу: https://link.springer.com/

38) Natural scene text localization and detection using MSER and its variants: a comprehensive survey K Dutta, R Sarkhel, M Kundu, M Nasipuri… — Multimedia Tools and , 2024 — Springer [Електронний ресурс] // Режим доступу: https://link.springer.com/

39) Layout–aware information extraction from semi–structured medical images K Luo, J Lu, KQ Zhu, W Gao, J Wei, M Zhang — Computers in biology and …, 2019 — Elsevier [Електронний ресурс] // Режим доступу: https://www.sciencedirect.com/

40) Addressing the data gap: building a parallel corpus for Kashmiri language SMU Qumar, M Azim, SMK Quadri — International Journal of Information, 2024 — Springer [Електронний ресурс] // Режим доступу:https://link.springer.com/

41) Unknown–box Approximation to Improve Optical Character Recognition Performance A Randika, N Ray, X Xiao, A Latimer — Recognition—ICDAR 2021: 16th , 2021 — Springer [Електронний ресурс] // Режим доступу: https://ualberta.scholaris.ca/

42) Real–Time Industrial Text Detection with Boundary Awareness and Refined Differentiation Y Yang, M Hu, J Yu, B Jing — Available at SSRN 4959851 — [Електронний ресурс] // Режим доступу: https://papers.ssrn.com

43) Predicting Users' Latent Suicidal Risk in Social Media: An Ensemble Model Based on Social Network Relationships X Meng, C Wang, J Yang, M Li — Materials & Continua, 2024 — [Електронний ресурс] // Режим доступу: https://search.ebscohost.com

44) Content Extraction and Recognition in Scientific Publications Z Wang — 2020 — [Електронний ресурс] // Режим доступу: https://oaktrust.library.tamu.edu

45) Small end-to-end OCR model J Dun — [Електронний ресурс] // Режим доступу: https://repository.tudelft.nl

46) Automatically Interpreting Dutch Tombstone Inscriptions J Bos, C Marocico, AE Tatar — Linguistics in the , 2022 — [Електронний ресурс] // Режим доступу: https://research.rug.nl

47) Advancing ancient arabic manuscript restoration with optimized deep learning and image enhancement techniques of the requirements for the degree of Master of Applied Science K Miloud, ML Abdelmounaim, B Mohammed… — Traitement du , 2024 — [Електронний ресурс] // Режим доступу: https://researchgate.net

48) Avancées en Reconnaissance Optique des Caractères pour les Documents Arabes Historiques B Kiessling — 2021 — [Електронний ресурс] // Режим доступу: https://theses.hal.science

49) Data–Driven Document Unwarping K Ma — 2021 — [Електронний ресурс] // Режим доступу: https://search.proquest.com

50) Facebook Spam Detection Extension Tool for Chromium Browser Architecture LK Azman, NA Abdullah – Applied Information Technology, 2023 — [Електронний ресурс] // Режим доступу: https://penerbit.uthm.edu.my

51) Efficiency and accuracy improvement of document verification: automation and method analysis Proc. SPIE 13574, Fourth International Conference on Electronic Information Engineering and Data Processing (EIEDP 2025), 1357450 (9 May 2025); [Електронний ресурс] // Режим доступу: https://doi.org/10.1117/12.3067142

52) From Tradition to Technology: A Systematic Survey on Navigating Pashto in Modern NLP ZA Khan, Y Xia, F Khaliq, JA Khan — Available at SSRN — 2024 pages 44, [Електронний ресурс] // Режим доступу: https://papers.ssrn.com

53) Which OCR toolset is good and why: A comparative study P Jain, K Taneja, H Taneja – Kuwait Journal of Science, 2021 — [Електронний ресурс] // Режим доступу: https://journalskuwait.org

54) Text Extraction from Document Images using CNN and LSTM Sandip Shinde;Sanket Bhosle;Gaurav Bomble;Shourya Bhosale;Siddhant Bokil 2023 International Conference on Sustainable Communication Networks and Application (ICSCNA) [Електронний ресурс] // Режим доступу: https://ieeexplore.ieee.org/

55) Clinical Natural Language Processing Systems for Information Retrieval from Unstructured Medical Narratives SL Sophie, SS Sathya – Medical Data Analysis and Processing ,2023 — [Електронний ресурс] // Режим доступу: https://taylorfrancis.com

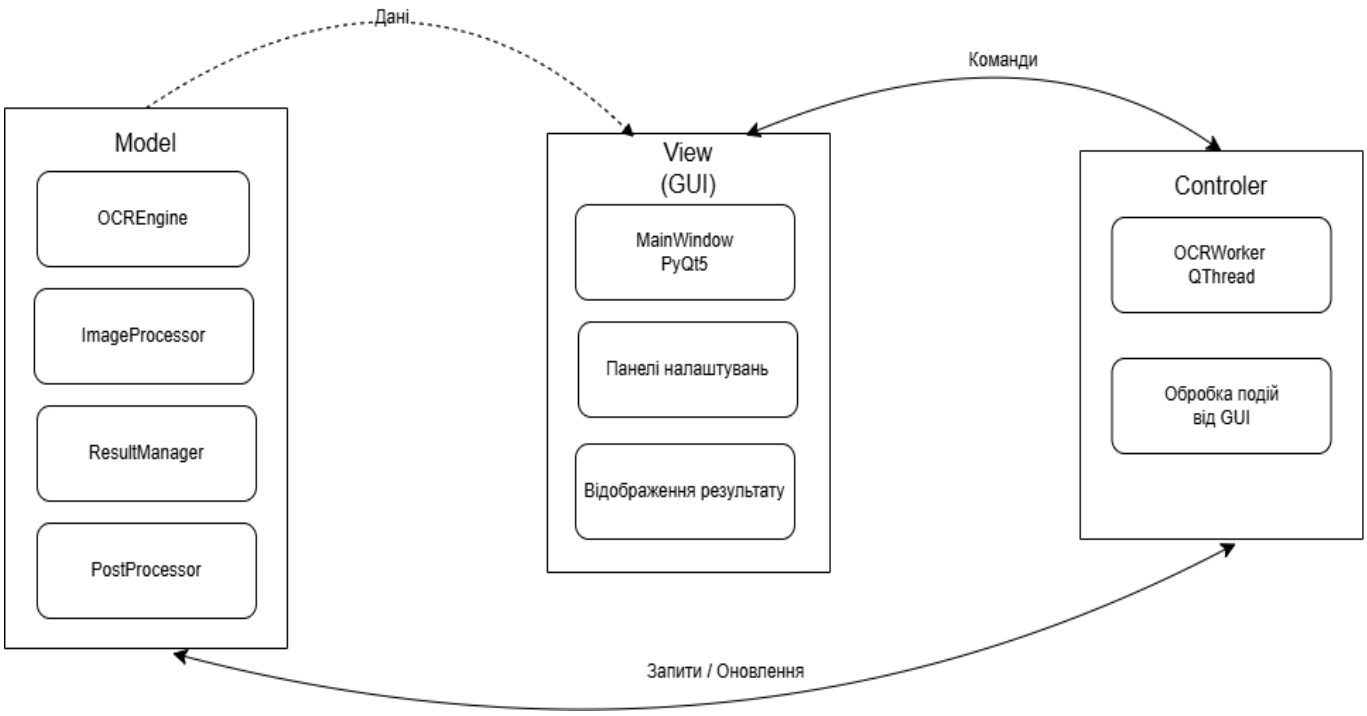
56) This is to certify that I am responsible for the work submitted in this Project, that the original work is mine, except as specified in acknowledgment and references A Musa — 2022 — [Електронний ресурс] // Режим доступу: https://academia.edu

57) PDF Software for Windows | FineReader PDF [Електронний ресурс] // Режим доступу: https://pdf.abbyy.com/finereader-pdf/ Software for Windows | FineReader PDF

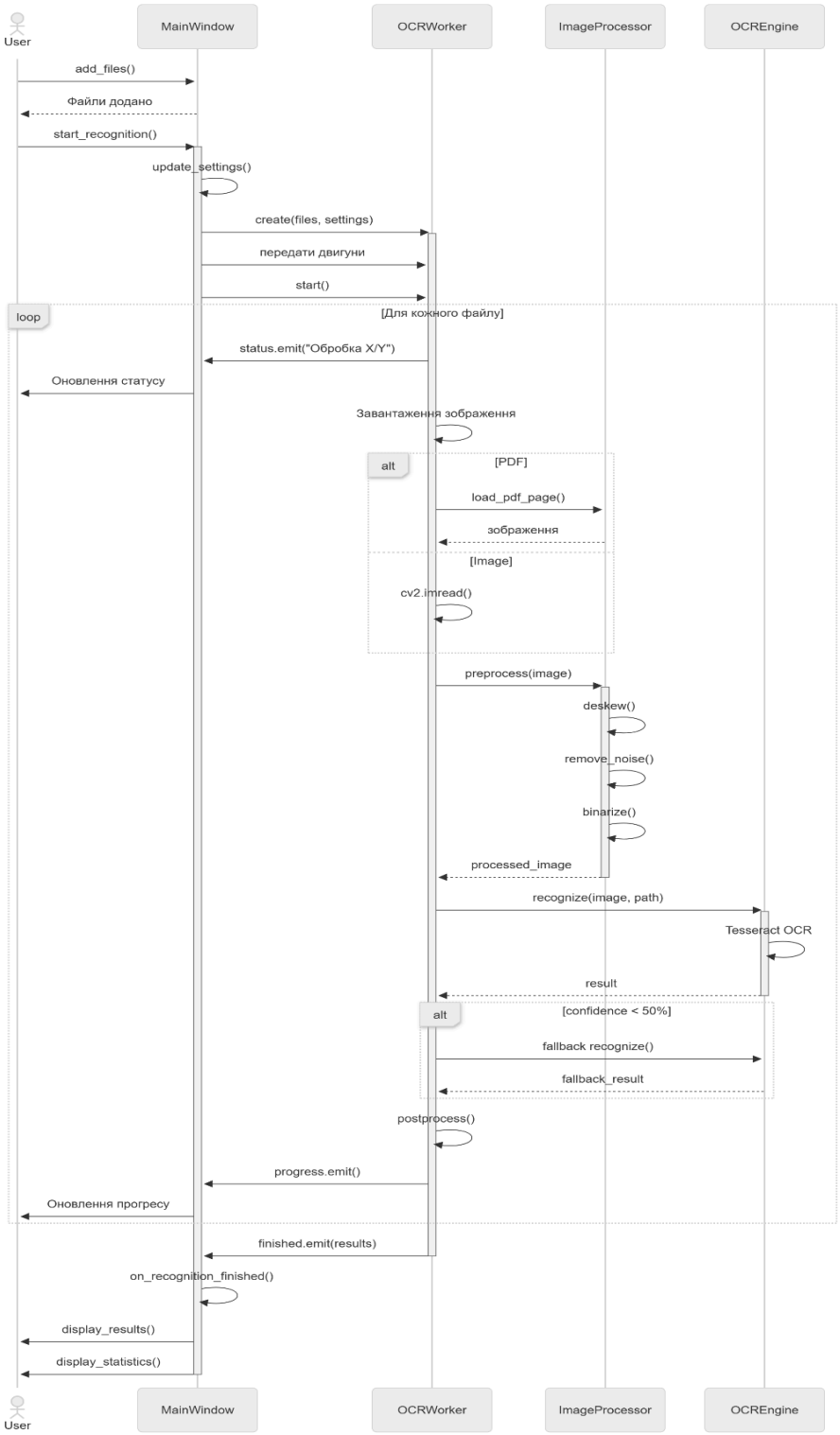
ДОДАТКИ

ДОДАТОК А: Діаграма прецедентів

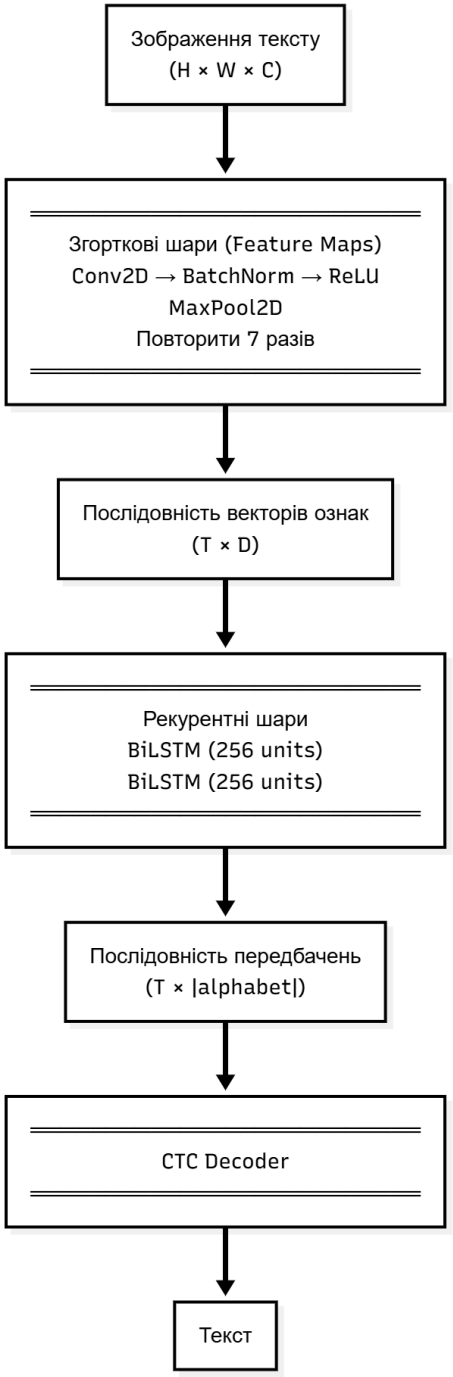
ДОДАТОК Б: Архітектура



ДОДАТОК В: Діаграма послідовності



ДОДАТОК Г: Архітектура CRNN



ДОДАТОК Д: Лістинг коду

image_processor.py

```
class ImageProcessor:

    def __init__(self):
        logger.info("ImageProcessor ініціалізовано")

    def preprocess(self, image_path: str, settings: Dict[str, Any]) -> np.ndarray:

        #Попередня обробка зображення

        try:
            logger.info(f"Початок обробки зображення: {image_path}")

            # Завантаження зображення
            if image_path.lower().endswith('.pdf'):
                image = self.load_pdf_page(image_path)
            else:
                image = cv2.imread(image_path)

            if image is None:
                raise ValueError(f"Не вдалося завантажити зображення: {image_path}")

            # Застосування обробок
            if settings.get('deskew', True):
                image = self.deskew(image)

            if settings.get('noise_removal', True):
                image = self.remove_noise(image)

            if settings.get('binarization') == 'adaptive':
                image = self.adaptive_binarization(image)
            elif settings.get('binarization') == 'otsu':
                image = self.otsu_binarization(image)

            # Покращення контрасту
            image = self.enhance_contrast(image)

            # Корекція освітлення
            image = self.correct_illumination(image)

            logger.info("Обробка зображення завершена")
            return image

        except Exception as e:
            logger.error(f"Помилка обробки зображення: {str(e)}", exc_info=True)
            # Повертаємо оригінальне зображення при помилці
            return cv2.imread(image_path)


    def deskew(self, image: np.ndarray) -> np.ndarray:

        #Виправлення нахилу зображення

        try:
            # Конвертація в відтінки сірого
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

            # Інверсія кольорів
            gray = cv2.bitwise_not(gray)
```

```
        # Бінаризація
        thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]

        # Знаходження координат всіх ненульових пікселів
        coords = np.column_stack(np.where(thresh > 0))

        # Обчислення кута нахилу
        angle = cv2.minAreaRect(coords)[-1]

        # Корекція кута
        if angle < -45:
            angle = -(90 + angle)
        else:
            angle = -angle

        # Виправлення нахилу тільки якщо кут значний
        if abs(angle) > 0.5:
            (h, w) = image.shape[:2]
            center = (w // 2, h // 2)
            M = cv2.getRotationMatrix2D(center, angle, 1.0)
            image = cv2.warpAffine(
                image, M, (w, h),
                flags=cv2.INTER_CUBIC,
                borderMode=cv2.BORDER_REPLICATE
            )
            logger.debug(f"Виправлено нахил на {angle:.2f} градусів")

        return image

    except Exception as e:
        logger.warning(f"Не вдалося виправити нахил: {str(e)}")
        return image

def remove_noise(self, image: np.ndarray) -> np.ndarray:

    #Видалення шуму з зображення

    try:
        # Використання медіанного фільтру
        denoised = cv2.medianBlur(image, 3)

        # Морфологічні операції для видалення дрібних артефактів
        kernel = np.ones((2, 2), np.uint8)
        denoised = cv2.morphologyEx(denoised, cv2.MORPH_OPEN, kernel)
        denoised = cv2.morphologyEx(denoised, cv2.MORPH_CLOSE, kernel)

        logger.debug("Шум видалено")
        return denoised

    except Exception as e:
        logger.warning(f"Не вдалося видалити шум: {str(e)}")
        return image

def adaptive_binarization(self, image: np.ndarray) -> np.ndarray:

    #Адаптивна бінаризація зображення

    try:
        # Конвертація в відтінки сірого
        if len(image.shape) == 3:
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        else:
            gray = image

        # Адаптивна бінаризація
        binary = cv2.adaptiveThreshold(
            gray,
            255,
            cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
            cv2.THRESH_BINARY,
            11,
            2
```

```
            )

            # Конвертація назад в BGR для сумісності
            binary = cv2.cvtColor(binary, cv2.COLOR_GRAY2BGR)

            logger.debug("Застосовано адаптивну бінаризацію")
            return binary

        except Exception as e:
            logger.warning(f"Не вдалося виконати бінаризацію: {str(e)}")
            return image

    def otsu_binarization(self, image: np.ndarray) -> np.ndarray:

        #Бінаризація методом Otsu

        try:
            # Конвертація в відтінки сірого
            if len(image.shape) == 3:
                gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            else:
                gray = image

            # Застосування фільтру Гауса для згладжування
            blur = cv2.GaussianBlur(gray, (5, 5), 0)

            # Бінаризація методом Otsu
            _, binary = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

            # Конвертація назад в BGR
            binary = cv2.cvtColor(binary, cv2.COLOR_GRAY2BGR)

            logger.debug("Застосовано бінаризацію Otsu")
            return binary

        except Exception as e:
            logger.warning(f"Не вдалося виконати бінаризацію Otsu: {str(e)}")
            return image

    def enhance_contrast(self, image: np.ndarray) -> np.ndarray:

        #Покращення контрасту зображення

        try:
            # Конвертація в Lab колірний простір
            lab = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)

            # Розділення каналів
            l, a, b = cv2.split(lab)

            # Застосування CLAHE до L каналу
            clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
            l = clahe.apply(l)

            # Об'єднання каналів
            lab = cv2.merge([l, a, b])

            # Конвертація назад в BGR
            enhanced = cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)

            logger.debug("Контраст покращено")
            return enhanced

        except Exception as e:
            logger.warning(f"Не вдалося покращити контраст: {str(e)}")
            return image

    def correct_illumination(self, image: np.ndarray) -> np.ndarray:

        #Корекція нерівномірного освітлення

        try:
```

```
            # Конвертація в відтінки сірого
            if len(image.shape) == 3:
                gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            else:
                gray = image

            # Застосування розмиття для знаходження фону
            dilated_img = cv2.dilate(gray, np.ones((7, 7), np.uint8))
            bg_img = cv2.medianBlur(dilated_img, 21)

            # Корекція освітлення
            diff_img = 255 - cv2.absdiff(gray, bg_img)

            # Нормалізація
            norm_img = cv2.normalize(
                diff_img,
                None,
                alpha=0,
                beta=255,
                norm_type=cv2.NORM_MINMAX,
                dtype=cv2.CV_8UC1
            )

            # Конвертація назад в BGR якщо потрібно
            if len(image.shape) == 3:
                norm_img = cv2.cvtColor(norm_img, cv2.COLOR_GRAY2BGR)

            logger.debug("Освітлення виправлено")
            return norm_img

        except Exception as e:
            logger.warning(f"Не вдалося виправити освітлення: {str(e)}")
            return image

    def resize_image(self, image: np.ndarray, target_dpi: int = 300) -> np.ndarray:

        #Зміна розміру зображення до цільової роздільної здатності

        try:
            height, width = image.shape[:2]

            # Обчислення коефіцієнта масштабування
            scale = target_dpi / 150.0

            # Обмеження максимального розміру
            if scale > 2.0:
                scale = 2.0

            new_width = int(width * scale)
            new_height = int(height * scale)

            # Зміна розміру
            resized = cv2.resize(
                image,
                (new_width, new_height),
                interpolation=cv2.INTER_CUBIC
            )

            logger.debug(f"Зображення масштабовано з ({width}x{height}) до ({new_width}x{new_height})")
            return resized

        except Exception as e:
            logger.warning(f"Не вдалося змінити розмір: {str(e)}")
            return image

    def detect_and_correct_perspective(self, image: np.ndarray) -> np.ndarray:

        #Виявлення та корекція перспективних спотворень

        try:
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            edges = cv2.Canny(gray, 50, 150, apertureSize=3)
```

```
        # Знаходження контурів
        contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

        # Знаходження найбільшого прямокутного контуру
        max_area = 0
        best_contour = None

        for contour in contours:
            area = cv2.contourArea(contour)
            if area > max_area:
                peri = cv2.arcLength(contour, True)
                approx = cv2.approxPolyDP(contour, 0.02 * peri, True)

                if len(approx) == 4:
                    max_area = area
                    best_contour = approx

        if best_contour is not None and max_area > (image.shape[0] * image.shape[1] * 0.5):
            # Сортування точок
            pts = best_contour.reshape(4, 2)
            rect = np.zeros((4, 2), dtype="float32")

            s = pts.sum(axis=1)
            rect[0] = pts[np.argmin(s)]
            rect[2] = pts[np.argmax(s)]

            diff = np.diff(pts, axis=1)
            rect[1] = pts[np.argmin(diff)]
            rect[3] = pts[np.argmax(diff)]

            # Обчислення розмірів
            (tl, tr, br, bl) = rect
            widthA = np.linalg.norm(br - bl)
            widthB = np.linalg.norm(tr - tl)
            maxWidth = int(max(widthA, widthB))

            heightA = np.linalg.norm(tr - br)
            heightB = np.linalg.norm(tl - bl)
            maxHeight = int(max(heightA, heightB))

            # Цільові точки
            dst = np.array([
                [0, 0],
                [maxWidth - 1, 0],
                [maxWidth - 1, maxHeight - 1],
                [0, maxHeight - 1]
            ], dtype="float32")

            # Обчислення матриці перспективного перетворення
            M = cv2.getPerspectiveTransform(rect, dst)

            # Застосування перетворення
            warped = cv2.warpPerspective(image, M, (maxWidth, maxHeight))

            logger.debug("Перспективу виправлено")
            return warped

        return image

    except Exception as e:
        logger.warning(f"Не вдалося виправити перспективу: {str(e)}")
        return image

def sharpen_image(self, image: np.ndarray) -> np.ndarray:
    try:
        # Ядро для підвищення різкості
        kernel = np.array([[-1, -1, -1],
                           [-1,  9, -1],
                           [-1, -1, -1]])

        sharpened = cv2.filter2D(image, -1, kernel)
```

```
            logger.debug("Різкість підвищено")
            return sharpened

        except Exception as e:
            logger.warning(f"Не вдалося підвищити різкість: {str(e)}")
            return image
```

ocr_engine.py

```
class TextDetector:
    #Виявлення текстових областей

    def __init__(self):
        self.confidence_threshold = 0.5
        logger.info("TextDetector ініціалізовано")

    def detect_text_regions(self, image: np.ndarray) -> List[Dict]:

        try:

            # Конвертація в PIL Image
            if isinstance(image, np.ndarray):
                pil_image = Image.fromarray(image)
            else:
                pil_image = image

            # Отримання даних про регіони
            data = pytesseract.image_to_data(
                pil_image,
                output_type=pytesseract.Output.DICT,
                lang='ukr+eng'
            )

            regions = []
            n_boxes = len(data['text'])

            for i in range(n_boxes):
                # Фільтрація порожніх регіонів
                if int(data['conf'][i]) > 0:
                    (x, y, w, h) = (
                        data['left'][i],
                        data['top'][i],
                        data['width'][i],
                        data['height'][i]
                    )

                    text = data['text'][i]
                    conf = int(data['conf'][i])

                    if text.strip() and conf > self.confidence_threshold * 100:
                        regions.append({
                            'bbox': (x, y, w, h),
                            'text': text,
                            'confidence': conf / 100.0
                        })

            logger.info(f"Виявлено {len(regions)} текстових регіонів")
            return regions

        except Exception as e:
            logger.error(f"Помилка виявлення текстових регіонів: {str(e)}")
            return []


class TextRecognizer:
    #Розпізнавання тексту

    def __init__(self, language='ukr+eng', quality='balanced'):
        self.language = language
```

```
        self.quality = quality
        self.setup_tesseract_config()
        logger.info(f"TextRecognizer ініціалізовано (мова: {language}, якість: {quality})")

    def setup_tesseract_config(self):

        config_parts = []

        # OEM (OCR Engine Mode) - використовуємо комбінацію для найкращих результатів
        config_parts.append('--oem 3')  # Legacy + LSTM (найкраще для цифр і тексту)

        # PSM (Page Segmentation Mode)
        config_parts.append('--psm 6')  # Uniform block of text (краще для документів)

        # ПОКРАЩЕННЯ ДЛЯ ЦИФР І СПЕЦСИМВОЛІВ:
        config_parts.append('-c classify_bln_numeric_mode=1')

        self.tesseract_config = ' '.join(config_parts)

        # Додаткова конфігурація тільки для цифр
        self.digits_config = '--oem 3 --psm 6 -c tessedit_char_whitelist=0123456789.,+-() '

        logger.debug(f"Tesseract config: {self.tesseract_config}")
        logger.debug(f"Digits config: {self.digits_config}")

    def recognize(self, image: np.ndarray) -> Dict[str, Any]:

        #Розпізнавання тексту на зображенні

        try:
            start_time = time.time()

            # Конвертація в PIL Image
            if isinstance(image, np.ndarray):
                pil_image = Image.fromarray(image)
            else:
                pil_image = image

            # Розпізнавання тексту
            text = pytesseract.image_to_string(
                pil_image,
                lang=self.language,
                config=self.tesseract_config
            )

            # Отримання детальних даних
            data = pytesseract.image_to_data(
                pil_image,
                lang=self.language,
                config=self.tesseract_config,
                output_type=pytesseract.Output.DICT
            )

            # Обчислення середньої впевненості
            confidences = [int(c) for c in data['conf'] if int(c) > 0]
            avg_confidence = sum(confidences) / len(confidences) if confidences else 0

            processing_time = time.time() - start_time

            result = {
                'text': text.strip(),
                'confidence': avg_confidence,
                'processing_time': processing_time,
                'word_count': len(text.split()),
                'char_count': len(text),
                'details': data
            }

            logger.info(f"Розпізнано {result['char_count']} символів за {processing_time:.2f}с")
            return result

        except Exception as e:
```

```
            logger.error(f"Помилка розпізнавання тексту: {str(e)}")
            return {
                'text': '',
                'confidence': 0,
                'processing_time': 0,
                'word_count': 0,
                'char_count': 0,
                'error': str(e)
            }


class PostProcessor:
    #Постобробка результатів

    def __init__(self, auto_correct=True):
        self.auto_correct = auto_correct
        logger.info("PostProcessor ініціалізовано")


class OCREngine:

    def __init__(self, settings: Dict[str, Any]):

        self.settings = settings

        # Ініціалізація компонентів
        self.detector = TextDetector()
        self.recognizer = TextRecognizer(
            language=settings.get('language', 'ukr+eng'),
            quality=settings.get('quality', 'balanced')
        )
        self.postprocessor = PostProcessor(
            auto_correct=settings.get('auto_correct', True)
        )

        logger.info("OCREngine ініціалізовано")

    def recognize(self, image: np.ndarray, image_path: str) -> Dict[str, Any]:

        try:
            start_time = time.time()
            logger.info(f"Початок розпізнавання: {image_path}")

            # Виявлення текстових регіонів (опціонально)
            if self.settings.get('detect_regions', False):
                regions = self.detector.detect_text_regions(image)
            else:
                regions = []

            # Розпізнавання тексту
            recognition_result = self.recognizer.recognize(image)

            # Постобробка
            if recognition_result['text']:
                processed_text = self.postprocessor.process(
                    recognition_result['text'],
                    self.settings.get('language', 'ukr+eng')
                )
            else:
                processed_text = ''

            total_time = time.time() - start_time

            result = {
                'image_path': image_path,
                'text': processed_text,
                'raw_text': recognition_result['text'],
                'confidence': recognition_result.get('confidence', 0),
                'processing_time': total_time,
                'word_count': len(processed_text.split()) if processed_text else 0,
                'char_count': len(processed_text) if processed_text else 0,
```

```
            'regions': regions,
            'settings': self.settings.copy()
        }

        logger.info(
            f"Розпізнавання завершено: {result['char_count']} символів, "
            f"впевненість {result['confidence']:.1f}%, час {total_time:.2f}с"
        )

        return result

    except Exception as e:
        logger.error(f"Помилка при розпізнаванні {image_path}: {str(e)}", exc_info=True)
        return {
            'image_path': image_path,
            'text': '',
            'raw_text': '',
            'confidence': 0,
            'processing_time': 0,
            'word_count': 0,
            'char_count': 0,
            'regions': [],
            'error': str(e)
        }


class BatchOCRProcessor:
    #Пакетна обробки документів

    def __init__(self, settings: Dict[str, Any]):
        self.settings = settings
        self.engine = OCREngine(settings)
        logger.info("BatchOCRProcessor ініціалізовано")

    def process_batch(self, image_paths: List[str],
                      progress_callback=None,
                      status_callback=None) -> List[Dict[str, Any]]:

        results = []
        total = len(image_paths)

        for idx, image_path in enumerate(image_paths):
            start_batch_item_time = time.time()
            try:
                # Оновлення статусу
                if status_callback:
                    status_callback(f"Обробка {idx+1}/{total}: {os.path.basename(image_path)}")


                image_np = None
                try:
                    # Спочатку OpenCV
                    image_np = cv2.imread(image_path)
                    if image_np is None:
                        logger.warning(f"cv2.imread failed for {image_path}, trying PIL")
                        # Якщо OpenCV не зміг, використовуємо PIL
                        pil_img = Image.open(image_path)
                        # Конвертуємо в BGR для Tesseract
                        if pil_img.mode == 'RGB':
                            image_np = cv2.cvtColor(np.array(pil_img), cv2.COLOR_RGB2BGR)
                        elif pil_img.mode == 'L':
                            image_np = cv2.cvtColor(np.array(pil_img), cv2.COLOR_GRAY2BGR)
                        elif pil_img.mode == 'RGBA':
                            image_np = cv2.cvtColor(np.array(pil_img), cv2.COLOR_RGBA2BGR)
                        else:
                            image_np = cv2.cvtColor(np.array(pil_img.convert('RGB')), cv2.COLOR_RGB2BGR)
                        if image_np is None: raise ValueError("PIL also failed to load")
                except Exception as load_e:
                    logger.error(f"Failed to load image {image_path} in BatchProcessor: {load_e}")
                    results.append({'image_path': image_path, 'text': '', 'error': f"Load error: {load_e}",
                                    'processing_time': time.time() - start_batch_item_time})
                    if progress_callback: progress_callback(int((idx + 1) / total * 100))
```

```
                continue

            result = self.engine.recognize(image_np, image_path)

            # Розпізнавання
            result = self.engine.recognize(image, image_path)
            results.append(result)

            # Оновлення прогресу
            if progress_callback:
                progress = int((idx + 1) / total * 100)
                progress_callback(progress)

        except Exception as e:
            logger.error(f"Помилка обробки {image_path}: {str(e)}")
            results.append({
                'image_path': image_path,
                'text': '',
                'error': str(e)
            })

    logger.info(f"Пакетна обробка завершена: {len(results)}/{total} файлів")
    return results
```

result_manager.py

```
class ResultManager:

    def __init__(self):
        logger.info("ResultManager ініціалізовано")

    def export(self, results: List[Dict[str, Any]],
               output_path: str,
               format_type: str,
               settings: Dict[str, Any],
               edited_text: str = None) -> bool:

        try:
            logger.info(f"Експорт у формат {format_type}: {output_path}")

            if format_type == 'txt':
                return self.export_txt(results, output_path, settings, edited_text)
            elif format_type == 'docx':
                return self.export_docx(results, output_path, settings, edited_text)
            elif format_type == 'pdf':
                return self.export_pdf(results, output_path, settings, edited_text)
            elif format_type == 'json':
                return self.export_json(results, output_path, settings, edited_text)
            else:
                logger.error(f"Непідтримуваний формат: {format_type}")
                return False
        except Exception as e:
            logger.error(f"Помилка експорту: {str(e)}", exc_info=True)
            return False

    def _generate_statistics_text(self, results: List[Dict[str, Any]]) -> List[str]:
        # Генерує текстові рядки статистики на основі ОРИГІНАЛЬНИХ результатів
        stats = []
        if not results:
            stats.append("Статистика недоступна (немає оригінальних результатів).")
            return stats

        successful_results = [r for r in results if not r.get('error')]


        total_pages = len(results)
        successful_pages = len(successful_results)
```

```
    failed_pages = total_pages - successful_pages

    # Визначаємо кількість унікальних файлів
    total_files = len(set(r.get('image_path') for r in results))


    total_chars = sum(r.get('char_count', 0) for r in successful_results)
    total_words = sum(r.get('word_count', 0) for r in successful_results)
    avg_conf = sum(r.get('confidence', 0) for r in successful_results) / successful_pages if successful_pages else 0
    total_time = sum(r.get('processing_time', 0) for r in results)

    stats.append(f"Всього файлів: {total_files}")
    stats.append(f"Всього сторінок (спроб): {total_pages}")
    stats.append(f"Успішно розпізнано (сторінок): {successful_pages}")
    stats.append(f"Сторінки з помилками: {failed_pages}")
    stats.append(f"Всього символів (успішних): {total_chars:,}")
    stats.append(f"Всього слів (успішних): {total_words:,}")
    stats.append(f"Середня впевненість (успішних): {avg_conf:.1f}%")
    stats.append(f"Загальний час (всіх сторінок): {total_time:.2f}с")
    return stats

def export_txt(self, results: List[Dict[str, Any]],
               output_path: str,
               settings: Dict[str, Any],
               edited_text: str = None) -> bool:
    #Експорт у текстовий файл
    try:
        with open(output_path, 'w', encoding='utf-8') as f:

            f.write("=" * 80 + "\n")
            f.write("РЕЗУЛЬТАТИ РОЗПІЗНАВАННЯ ТЕКСТУ\n")
            f.write(f"Дата: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")
            f.write(f"Мова: {settings.get('language', 'N/A')}\n")
            engine_str = "N/A"
            if results and results[0].get('engine'): engine_str = results[0]['engine'].replace('_fallback',' (F)')
            f.write(f"Двигун: {engine_str}\n")
            if edited_text is not None: f.write("Статус: Текст міг бути відредаговано\n")
            f.write("=" * 80 + "\n\n")


            f.write("--- РОЗПІЗНАНИЙ (АБО ВІДРЕДАГОВАНИЙ) ТЕКСТ ---\n\n")
            if edited_text is not None:
                logger.info("Експорт TXT: Використання відредагованого тексту.")
                f.write(edited_text)
            elif results:
                 logger.info("Експорт TXT: Використання оригінального тексту.")
                 full_text = "\n\n--- Новий документ ---\n\n".join(r.get('text', '') for r in results if not r.get('error'))
                 f.write(full_text if full_text else "[Немає успішно розпізнаного тексту]")
            else:
                 logger.warning("Експорт TXT: Немає тексту.")
                 f.write("[Немає тексту для експорту]")


            f.write(f"\n\n{'='*80}\n")
            f.write("СТАТИСТИКА (на основі оригінального розпізнавання)\n")
            f.write(f"{'='*80}\n")
            stats_lines = self._generate_statistics_text(results)
            for line in stats_lines:
                f.write(line + "\n")



        logger.info(f"Експорт у TXT успішно завершено: {output_path}")
        return True
    except Exception as e:
        logger.error(f"Помилка експорту у TXT: {str(e)}", exc_info=True)
        return False

def export_docx(self, results: List[Dict[str, Any]],
                output_path: str,
                settings: Dict[str, Any],
                edited_text: str = None) -> bool:
```

```
        # Експорт у Word документ
        try:
            from docx import Document
            from docx.shared import Pt, Inches
            from docx.enum.text import WD_ALIGN_PARAGRAPH

            doc = Document()
            style = doc.styles['Normal']; font = style.font; font.name = 'Arial'; font.size = Pt(11)


            title = doc.add_heading('Результати розпізнавання тексту', 0); title.alignment = WD_ALIGN_PARAGRAPH.CENTER
            info = doc.add_paragraph()
            info.add_run(f"Дата: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n").bold = True
            info.add_run(f"Мова: {settings.get('language', 'N/A')}\n")
            engine_str = "N/A"
            if results and results[0].get('engine'): engine_str = results[0]['engine'].replace('_fallback',' (F)')
            info.add_run(f"Двигун: {engine_str}\n")
            if edited_text is not None: info.add_run(f"Статус: Текст міг бути відредаговано\n")
            doc.add_paragraph()


            text_heading = "Відредагований текст:" if edited_text is not None else "Розпізнаний текст:"
            doc.add_heading(text_heading, 1)
            text_to_write = edited_text if edited_text is not None else "\n\n---\n\n".join(r.get('text', '') for r in results if not r.get('error'))
            if text_to_write:
                for para_text in text_to_write.split('\n'): # Зберігаємо абзаци
                    doc.add_paragraph(para_text)
            else:
                doc.add_paragraph("[Немає тексту]")


            doc.add_page_break()
            doc.add_heading('Статистика (на основі оригіналу)', 1)
            stats_lines = self._generate_statistics_text(results)

            for line in stats_lines:
                # Пропускаємо порожні рядки, якщо вони є
                if line.strip():
                    doc.add_paragraph(line)


            doc.save(output_path)
            logger.info(f"Експорт у DOCX успішно завершено: {output_path}")
            return True
        except ImportError:
            logger.error("Помилка експорту DOCX: pip install python-docx")
            raise Exception("Бібліотека python-docx не встановлена.")
        except Exception as e:
            logger.error(f"Помилка експорту у DOCX: {str(e)}", exc_info=True)
            return False
```

main.py

```
class OCRWorker(QThread):
    # Робочий потік для обробки зображень
    progress = pyqtSignal(int)
    status = pyqtSignal(str)
    finished = pyqtSignal(dict)
    error = pyqtSignal(str)

    # Додаємо атрибути для двигунів
    paddle_engine: Any = None
    hybrid_engine: Any = None
    custom_engine: Any = None
    use_paddle_ocr: bool = False
    use_hybrid_model: bool = False
    use_custom_model: bool = False

    def __init__(self, image_paths: List[str], settings: Dict[str, Any]):
```

```
    super().__init__()
    self.image_paths = image_paths
    self.settings = settings
    # Tesseract engine
    self.ocr_engine = OCREngine(settings)
    self.is_cancelled = False
    self.image_processor = ImageProcessor()
    logger.debug("OCRWorker ініціалізовано")

def run(self):

    logger.debug(f"OCRWorker: Початок run() для {len(self.image_paths)} файлів")
    logger.debug(f"OCRWorker: Налаштування: {self.settings}")

    results = []

    try:

        if self.use_paddle_ocr and self.paddle_engine:
            logger.info("Запуск OCRWorker в режимі PaddleOCR (без розбиття PDF)")
            total = len(self.image_paths)
            for idx, image_path in enumerate(self.image_paths):
                if self.is_cancelled:
                    logger.info(f"Обробка скасована (Paddle): {os.path.basename(image_path)}")
                    break

                self.status.emit(f"Обробка (Paddle) {idx + 1}/{total}: {os.path.basename(image_path)}")
                engine_start_time = time.time()
                try:

                    paddle_result = self.paddle_engine.recognize(image_path, return_confidence=True)
                    result = {
                        'text': paddle_result.get('text', ''),
                        'confidence': paddle_result.get('confidence', 0.0),
                        'image_path': image_path, 'engine': 'paddleocr',
                        'error': paddle_result.get('error'),
                        'processing_time': paddle_result.get('processing_time', 0.0)
                    }
                    if not result['error']:
                        logger.info(f"🌊 PaddleOCR OK: '{result['text'][:50]}...'")
                    else:
                        # Якщо Paddle сам повернув помилку, ми не робимо fallback
                        logger.warning(f"🌊 PaddleOCR Error: {result['error']}")

                except Exception as e:
                    logger.error(f"❌ Критична помилка PaddleOCR під час recognize(): {e}", exc_info=True)
                    result = {'image_path': image_path, 'text': '', 'error': str(e), 'engine': 'paddleocr_crash',
                              'processing_time': time.time() - engine_start_time}

                # Додаємо поля
                result.setdefault('char_count', len(result.get('text', '')))
                result.setdefault('word_count', len(result.get('text', '').split()))
                results.append(result)
                self.progress.emit(int((idx + 1) / total * 100))


        else:
            logger.info("Запуск OCRWorker в режимі Tesseract/Hybrid (з розбиттям PDF)")
            tasks = []
            self.status.emit("Підготовка файлів... (це може зайняти час для PDF)")


            for image_path in self.image_paths:
                if self.is_cancelled: break
                try:
                    display_image_path = image_path
                    if image_path.lower().endswith('.pdf'):
                        all_pages_images = self.image_processor.load_pdf_all_pages(image_path)
                        if not all_pages_images: raise ValueError("load_pdf_all_pages повернув порожній список")
                        for i, page_image in enumerate(all_pages_images):
                            tasks.append((page_image, image_path, i + 1))
                    else:
```

```
            loaded_image = cv2.imread(image_path)
            if loaded_image is None:
                pil_img = Image.open(image_path)
                if pil_img.mode == 'RGB':
                    loaded_image = cv2.cvtColor(np.array(pil_img), cv2.COLOR_RGB2BGR)
                elif pil_img.mode == 'L':
                    loaded_image = cv2.cvtColor(np.array(pil_img), cv2.COLOR_GRAY2BGR)
                elif pil_img.mode == 'RGBA':
                    loaded_image = cv2.cvtColor(np.array(pil_img), cv2.COLOR_RGBA2BGR)
                else:
                    loaded_image = cv2.cvtColor(np.array(pil_img.convert('RGB')), cv2.COLOR_RGB2BGR)
            if loaded_image is not None:
                tasks.append((loaded_image, image_path, 0))
            else:
                raise ValueError("cv2.imread та PIL не змогли завантажити зображення")
        except ImportError as imp_err:
            results.append({'image_path': image_path, 'text': '',
                            'error': f"Помилка PDF: pdf2image/Poppler не встановлено.",
                            'engine': 'loading_error', 'char_count': 0, 'word_count': 0, 'confidence': 0.0,
                            'processing_time': 0})
        except Exception as load_err:
            results.append(
                {'image_path': image_path, 'text': '', 'error': f"Помилка завантаження: {load_err}",
                 'engine': 'loading_error', 'char_count': 0, 'word_count': 0, 'confidence': 0.0,
                 'processing_time': 0})

    if self.is_cancelled:
        self.finished.emit({'results': results, 'success': True})
        return


    logger.info(f"Підготовка завершена. Початок обробки {len(tasks)} завдань.")
    total = len(tasks)
    for idx, (loaded_image, original_path, page_num) in enumerate(tasks):
        if self.is_cancelled:
            logger.info(f"Обробка скасована на завданні {idx + 1}/{total}")
            break

        if page_num > 0:
            display_name = f"{os.path.basename(original_path)} (стор. {page_num})"
        else:
            display_name = os.path.basename(original_path)
        self.status.emit(f"Обробка {idx + 1}/{total}: {display_name}")
        engine_start_time = time.time()
        result = None



        if self.use_hybrid_model and self.hybrid_engine:
            try:
                self.status.emit(f"🚀 Hybrid...")
                text, confidence = self.hybrid_engine.recognize(loaded_image, return_confidence=True)
                result = {'text': text, 'confidence': confidence, 'image_path': original_path,
                          'engine': 'hybrid'}
            except Exception as e:
                logger.error(f" Помилка hybrid: {e}, fallback...", exc_info=True)
                self.status.emit(f"Fallback Tesseract...")
                try:
                    result = self.ocr_engine.recognize(loaded_image, original_path)
                    result['engine'] = 'tesseract_fallback'
                except Exception as te:
                    result = {'image_path': original_path, 'text': '', 'error': str(te),
                              'engine': 'tesseract_fallback'}

        elif self.use_custom_model and self.custom_engine:
            try:
                self.status.emit(f" CRNN...")
                text, confidence = self.custom_engine.recognize(loaded_image, return_confidence=True)
                result = {'text': text, 'confidence': confidence, 'image_path': original_path,
                          'engine': 'crnn'}
            except Exception as e:
                logger.error(f" Помилка CRNN: {e}, fallback...", exc_info=True)
```

```
                    self.status.emit(f" Fallback Tesseract...")
                    try:
                        result = self.ocr_engine.recognize(loaded_image, original_path)
                        result['engine'] = 'tesseract_fallback'
                    except Exception as te:
                        result = {'image_path': original_path, 'text': '', 'error': str(te),
                                  'engine': 'tesseract_fallback'}

                else:
                    try:
                        self.status.emit(f"Tesseract...")
                        result = self.ocr_engine.recognize(loaded_image, original_path)
                        result['engine'] = 'tesseract'
                    except Exception as e:
                        logger.error(f"Помилка Tesseract: {e}", exc_info=True)
                        result = {'image_path': original_path, 'text': '', 'error': str(e), 'engine': 'tesseract'}


                if result:
                    result.setdefault('text', '')
                    result.setdefault('confidence', 0.0)
                    result.setdefault('engine', 'unknown')
                    result.setdefault('error', None)
                    result['char_count'] = len(result['text'])
                    result['word_count'] = len(result['text'].split())
                    result.setdefault('processing_time', time.time() - engine_start_time)
                    if page_num > 0: result['page'] = page_num
                    results.append(result)
                else:
                    logger.error(f"Не вдалося отримати 'result' для {display_name}")
                    results.append({'image_path': original_path, 'text': '', 'error': 'Unknown processing error',
                                    'engine': 'unknown', 'char_count': 0, 'word_count': 0, 'confidence': 0.0,
                                    'processing_time': 0, 'page': page_num if page_num > 0 else 1})

                self.progress.emit(int((idx + 1) / total * 100))


            if not self.is_cancelled:
                logger.info("Цикл обробки завдань завершено, надсилання сигналу finished.")
                self.finished.emit({'results': results, 'success': True})
            else:
                logger.info("Цикл обробки перервано через скасування.")

        except Exception as e:
            logger.error(f"Критична помилка в OCRWorker run(): {str(e)}", exc_info=True)
            if not self.is_cancelled:
                self.error.emit(f"Критична помилка потоку: {str(e)}")

    def cancel(self):

        logger.info("OCRWorker: Cancel requested.")
        self.is_cancelled = True


class MainWindow(QMainWindow):


    def __init__(self):
        super().__init__()

        global PADDLEOCR_AVAILABLE, CUSTOM_MODEL_AVAILABLE, HYBRID_MODEL_AVAILABLE
        self.image_paths = []
        self.current_results = []
        self.settings = self.get_default_settings()
        self.worker: OCRWorker = None
        self.result_manager = ResultManager()
        self.image_processor = ImageProcessor()


        self.use_paddle_ocr = False
        self.use_hybrid_model = False
        self.use_custom_model = False
```

```
        self.custom_engine = None
        if CUSTOM_MODEL_AVAILABLE:
            try:
                logger.info("Спроба завантаження CRNN...")
                self.custom_engine = CustomOCREngine('best_ocr_model.pth')
                logger.info("CRNN завантажено")
            except Exception as e:
                logger.warning(f"Помилка завантаження CRNN (ігнорується): {e}", exc_info=False)
                CUSTOM_MODEL_AVAILABLE = False

        self.hybrid_engine = None
        if HYBRID_MODEL_AVAILABLE:
            try:
                logger.info("Спроба завантаження Hybrid...")
                self.hybrid_engine = HybridOCREngine('best_hybrid_model.pth')
                logger.info(" Hybrid завантажено")
            except Exception as e:
                logger.warning(f"Помилка завантаження Hybrid (ігнорується): {e}", exc_info=False)
                HYBRID_MODEL_AVAILABLE = False

        self.paddle_engine = None
        if PADDLEOCR_AVAILABLE:
            try:
                logger.info("Спроба ініціалізації PaddleOCR...")
                self.paddle_engine = PaddleOCREngine(lang='uk')
                logger.info(" PaddleOCR ініціалізовано")
            except Exception as e:
                logger.error(f"!! Критична помилка ініціалізації PaddleOCR: {e}", exc_info=True)
                PADDLEOCR_AVAILABLE = False
                QMessageBox.critical(self, "Помилка PaddleOCR", f"Не вдалося ініціалізувати PaddleOCR:\n{e}\n\nДвигун буде недоступний.")
                self.image_processor = ImageProcessor()


        self.export_panel = None
        self.zoom_factor = 1.15
        self.min_zoom = 0.1
        self.max_zoom = 10.0
        self.init_ui()
        self.apply_styles()


        self.change_ocr_engine("Tesseract (стандарт)")

        logger.info("Застосунок MainWindow повністю ініціалізовано")

    def eventFilter(self, obj, event):

        if obj is self.image_scroll_area.viewport() and event.type() == QEvent.Wheel:
            if self.current_pixmap:

                delta = event.angleDelta().y()
                if delta > 0:
                    logger.debug("Wheel Up: Zoom In")
                    self.zoom_in()
                elif delta < 0:
                    logger.debug("Wheel Down: Zoom Out")
                    self.zoom_out()
                return True

        return super().eventFilter(obj, event)

    def init_ui(self):
        self.setWindowTitle("Розпізнавання тексту")
        self.setGeometry(100, 100, 1400, 900)
        central_widget = QWidget(); self.setCentralWidget(central_widget)
        main_layout = QHBoxLayout(central_widget); main_layout.setSpacing(10); main_layout.setContentsMargins(10, 10, 10, 10)
        left_panel = self.create_left_panel(); main_layout.addWidget(left_panel, stretch=1)
        right_panel = self.create_right_panel(); main_layout.addWidget(right_panel, stretch=3)
        self.statusBar().showMessage("Готово")
        self.image_scroll_area.viewport().installEventFilter(self)
```

```
        logger.debug("UI ініціалізовано")

    def create_left_panel(self):
        panel = QWidget(); layout = QVBoxLayout(panel); layout.setSpacing(15)
        title = QLabel("Панель управління"); title.setFont(QFont("Arial", 14, QFont.Bold)); title.setAlignment(Qt.AlignCenter); layout.addWidget(title)

        file_group = QGroupBox("Файли"); file_layout = QVBoxLayout()
        self.btn_add_files = QPushButton("📁 Додати файли"); self.btn_add_files.setMinimumHeight(40); self.btn_add_files.clicked.connect(self.add_files);
file_layout.addWidget(self.btn_add_files)
        self.btn_add_folder = QPushButton("📂 Додати папку"); self.btn_add_folder.setMinimumHeight(40);
self.btn_add_folder.clicked.connect(self.add_folder); file_layout.addWidget(self.btn_add_folder)
        self.btn_clear = QPushButton("🗑️ Очистити"); self.btn_clear.setMinimumHeight(40); self.btn_clear.clicked.connect(self.clear_files);
file_layout.addWidget(self.btn_clear)
        file_group.setLayout(file_layout); layout.addWidget(file_group)

        files_label = QLabel("Файли для обробки:"); files_label.setFont(QFont("Arial", 10, QFont.Bold)); layout.addWidget(files_label)
        self.file_list = QListWidget(); self.file_list.setMinimumHeight(200); layout.addWidget(self.file_list)

        settings_group = QGroupBox("Налаштування"); settings_layout = QVBoxLayout()
        lang_layout = QHBoxLayout(); lang_layout.addWidget(QLabel("Мова:")); self.lang_combo = QComboBox(); self.lang_combo.addItems(["Українська",
"Англійська", "Українська+Англійська"]); self.lang_combo.setCurrentIndex(0); self.lang_combo.currentTextChanged.connect(self.update_settings);
lang_layout.addWidget(self.lang_combo); settings_layout.addLayout(lang_layout)
        quality_layout = QHBoxLayout(); quality_layout.addWidget(QLabel("Якість (Tesseract):")); self.quality_combo = QComboBox();
self.quality_combo.addItems(["Швидка", "Збалансована", "Висока точність"]); self.quality_combo.setCurrentIndex(1);
self.quality_combo.currentTextChanged.connect(self.update_settings); quality_layout.addWidget(self.quality_combo);
settings_layout.addLayout(quality_layout)

        engine_layout = QHBoxLayout(); engine_layout.addWidget(QLabel("OCR двигун:")); self.engine_combo = QComboBox()
        options = ["📝 Tesseract (стандарт)"]

        if PADDLEOCR_AVAILABLE: options.append("🌊 PaddleOCR (рекомендовано)")
        if CUSTOM_MODEL_AVAILABLE: options.append("🤖 CRNN модель")
        if HYBRID_MODEL_AVAILABLE: options.append("🚀 Гібридна модель")
        self.engine_combo.addItems(options)
        self.engine_combo.setCurrentIndex(0); self.engine_combo.setItemData(0, "Tesseract OCR", Qt.ToolTipRole)
        idx = 1
        if PADDLEOCR_AVAILABLE: self.engine_combo.setItemData(idx, "PaddleOCR - висока точність", Qt.ToolTipRole); idx += 1
        if CUSTOM_MODEL_AVAILABLE: self.engine_combo.setItemData(idx, "CRNN модель", Qt.ToolTipRole); idx += 1
        if HYBRID_MODEL_AVAILABLE: self.engine_combo.setItemData(idx, "Гібридна модель (CNN+LSTM+Att)", Qt.ToolTipRole)
        self.engine_combo.currentTextChanged.connect(self.change_ocr_engine); engine_layout.addWidget(self.engine_combo);
settings_layout.addLayout(engine_layout)

        self.model_info_label = QLabel(""); self.model_info_label.setStyleSheet("color: #666; font-size: 10px; padding: 5px;");
self.model_info_label.setWordWrap(True); settings_layout.addWidget(self.model_info_label)
        settings_group.setLayout(settings_layout); layout.addWidget(settings_group)

        self.btn_start = QPushButton("▶️ Розпізнати"); self.btn_start.setMinimumHeight(50); self.btn_start.setEnabled(False);
self.btn_start.clicked.connect(self.start_recognition); layout.addWidget(self.btn_start)
        self.btn_cancel = QPushButton("⏹️ Зупинити"); self.btn_cancel.setMinimumHeight(50); self.btn_cancel.setEnabled(False);
self.btn_cancel.clicked.connect(self.cancel_recognition); layout.addWidget(self.btn_cancel)
        self.progress_bar = QProgressBar(); self.progress_bar.setMinimumHeight(30); layout.addWidget(self.progress_bar)
        self.status_label = QLabel("Статус: Очікування"); self.status_label.setWordWrap(True); layout.addWidget(self.status_label)
        layout.addStretch()
        logger.debug("Ліва панель створена")
        return panel

    def create_right_panel(self):
        panel = QWidget(); layout = QVBoxLayout(panel); layout.setContentsMargins(0,0,0,0)
        self.tabs = QTabWidget(); self.image_tab = self.create_image_tab(); self.tabs.addTab(self.image_tab, "🖼️ Зображення")
        self.results_tab = self.create_results_tab(); self.tabs.addTab(self.results_tab, "📝 Текст")
        self.stats_tab = self.create_stats_tab(); self.tabs.addTab(self.stats_tab, "📊 Статистика")
        layout.addWidget(self.tabs); self.export_panel = self.create_export_panel(); layout.addWidget(self.export_panel)
        logger.debug("Права панель створена")
        return panel

    def create_image_tab(self):
        tab = QWidget(); layout = QVBoxLayout(tab); header = QHBoxLayout()
        self.image_title = QLabel("Перегляд"); self.image_title.setFont(QFont("Arial", 12, QFont.Bold)); header.addWidget(self.image_title);
header.addStretch()
        self.btn_zoom_in = QPushButton("🔍+"); self.btn_zoom_in.clicked.connect(self.zoom_in); header.addWidget(self.btn_zoom_in)
        self.btn_zoom_out = QPushButton("🔍-"); self.btn_zoom_out.clicked.connect(self.zoom_out); header.addWidget(self.btn_zoom_out)
        self.btn_zoom_fit = QPushButton("⤢"); self.btn_zoom_fit.clicked.connect(self.zoom_fit); header.addWidget(self.btn_zoom_fit)
```

```
        layout.addLayout(header)
        self.image_scroll_area = QScrollArea()
        scroll = self.image_scroll_area
        scroll.setWidgetResizable(True); scroll.setAlignment(Qt.AlignCenter)
        self.image_label = PanningLabel(scroll_area=scroll)
        self.image_label.setAlignment(Qt.AlignCenter)
        self.image_label.setStyleSheet("QLabel { background-color: #eee; border: 1px solid #ccc; min-height: 300px; min-width: 300px; }")
        self.image_label.setStyleSheet("QLabel { background-color: #f0f0f0; }"); self.image_label.setText("Немає зображення")
        scroll.setWidget(self.image_label); layout.addWidget(scroll); self.current_zoom = 1.0; self.current_pixmap = None
        return tab

    def create_results_tab(self):
        tab = QWidget(); layout = QVBoxLayout(tab); header = QHBoxLayout()
        results_title = QLabel("Розпізнаний текст"); results_title.setFont(QFont("Arial", 12, QFont.Bold)); header.addWidget(results_title); header.addStretch()
        self.btn_copy = QPushButton("📋 Копіювати"); self.btn_copy.clicked.connect(self.copy_text); header.addWidget(self.btn_copy)
        self.btn_clear_text = QPushButton("🗑️ Очистити"); self.btn_clear_text.clicked.connect(self.clear_text); header.addWidget(self.btn_clear_text)
        layout.addLayout(header); self.result_text = QTextEdit(); self.result_text.setFont(QFont("Consolas", 11)); self.result_text.setPlaceholderText("Тут буде
текст...")
        layout.addWidget(self.result_text)
        return tab

    def create_stats_tab(self):
        tab = QWidget(); layout = QVBoxLayout(tab); stats_title = QLabel("Статистика"); stats_title.setFont(QFont("Arial", 12, QFont.Bold));
layout.addWidget(stats_title)
        self.stats_text = QTextEdit(); self.stats_text.setFont(QFont("Arial", 10)); self.stats_text.setReadOnly(True);
self.stats_text.setPlaceholderText("Статистика з'явиться тут...")
        layout.addWidget(self.stats_text)
        return tab

    def create_export_panel(self):
        panel = QGroupBox("Експорт"); layout = QHBoxLayout(panel)
        self.btn_export_txt = QPushButton("💾 TXT"); self.btn_export_txt.clicked.connect(lambda: self.export_results('txt'));
layout.addWidget(self.btn_export_txt)
        self.btn_export_docx = QPushButton("💾 DOCX"); self.btn_export_docx.clicked.connect(lambda: self.export_results('docx'));
layout.addWidget(self.btn_export_docx)
        self.btn_export_pdf = QPushButton("💾 PDF"); self.btn_export_pdf.clicked.connect(lambda: self.export_results('pdf'));
layout.addWidget(self.btn_export_pdf)
        self.btn_export_json = QPushButton("💾 JSON"); self.btn_export_json.clicked.connect(lambda: self.export_results('json'));
layout.addWidget(self.btn_export_json)
        for i in range(layout.count()): layout.itemAt(i).widget().setEnabled(False) # Вимикаємо кнопки спочатку
        return panel

    def apply_styles(self):
        self.setStyleSheet("""
            QMainWindow { background-color: #f5f5f5; }
            QPushButton { background-color: #4CAF50; color: white; border: none; padding: 8px 16px; border-radius: 4px; font-size: 13px; font-weight: bold; }
            QPushButton:hover { background-color: #45a049; } QPushButton:pressed { background-color: #3d8b40; }
            QPushButton:disabled { background-color: #cccccc; color: #666666; }
            QPushButton#btn_cancel { background-color: #f44336; } QPushButton#btn_cancel:hover { background-color: #da190b; }
            QGroupBox { font-weight: bold; border: 2px solid #cccccc; border-radius: 6px; margin-top: 12px; padding-top: 10px; background-color: white; }
            QGroupBox::title { subcontrol-origin: margin; left: 10px; padding: 0 5px; }
            QListWidget { border: 1px solid #cccccc; border-radius: 4px; background-color: white; padding: 5px; }
            QTextEdit { border: 1px solid #cccccc; border-radius: 4px; background-color: white; padding: 10px; font-family: Consolas, monospace; } /* Змінено
шрифт */
            QProgressBar { border: 1px solid #cccccc; border-radius: 4px; text-align: center; background-color: white; }
            QProgressBar::chunk { background-color: #4CAF50; border-radius: 3px; }
            QComboBox { border: 1px solid #cccccc; border-radius: 4px; padding: 5px; background-color: white; }
            QTabWidget::pane { border: 1px solid #cccccc; border-radius: 4px; background-color: white; }
            QTabBar::tab { background-color: #e0e0e0; padding: 10px 20px; margin-right: 2px; border-top-left-radius: 4px; border-top-right-radius: 4px; }
            QTabBar::tab:selected { background-color: white; border-bottom: 2px solid #4CAF50; }
            QLabel { color: #333333; }
        """)
        self.btn_cancel.setObjectName("btn_cancel")
        logger.debug("Стилі застосовано")

    def get_default_settings(self) -> Dict[str, Any]:
        return {'language': 'ukr', 'quality': 'balanced', 'auto_correct': True, 'preserve_layout': True, 'dpi': 300 }

    def update_settings(self):
        """Оновлення налаштувань"""
        lang_map = {"Українська": "ukr", "Англійська": "eng", "Українська+Англійська": "ukr+eng"}
        self.settings['language'] = lang_map.get(self.lang_combo.currentText(), 'ukr')
```

```
        quality_map = {"Швидка": "fast", "Збалансована": "balanced", "Висока точність": "high"}
        self.settings['quality'] = quality_map.get(self.quality_combo.currentText(), 'balanced')

        logger.info(f"Налаштування оновлено: Мова={self.settings['language']}, Якість Tesseract={self.settings['quality']}")


        if not (self.use_paddle_ocr or self.use_hybrid_model or self.use_custom_model):
            if self.worker and hasattr(self.worker, 'ocr_engine') and self.worker.ocr_engine:
                try:
                    logger.debug("Спроба оновлення Tesseract config у активному worker...")
                    self.worker.ocr_engine.recognizer.language = self.settings['language']
                    self.worker.ocr_engine.recognizer.quality = self.settings['quality']
                    self.worker.ocr_engine.recognizer.setup_tesseract_config()
                    logger.info(f"Tesseract config у worker оновлено: lang={self.settings['language']}, quality={self.settings['quality']}")
                except Exception as e:
                    logger.error(f"Помилка оновлення Tesseract config у worker: {e}")
            else:
                logger.debug("Tesseract активний, але worker не запущено, налаштування будуть застосовані при старті.")


    def change_ocr_engine(self, engine_text: str):
        # Зміна OCR двигуна
        global PADDLEOCR_AVAILABLE, CUSTOM_MODEL_AVAILABLE, HYBRID_MODEL_AVAILABLE
        logger.info(f"Зміна двигуна UI на: '{engine_text}'")


        self.use_paddle_ocr = False
        self.use_hybrid_model = False
        self.use_custom_model = False

        # Визначення активного двигуна
        if ("PaddleOCR" in engine_text or "🌊" in engine_text) and PADDLEOCR_AVAILABLE:
            self.use_paddle_ocr = True
            status_msg = "🌊 Використовується PaddleOCR"
        elif ("Гібридна модель" in engine_text or "🚀" in engine_text) and HYBRID_MODEL_AVAILABLE:
            self.use_hybrid_model = True
            status_msg = "🚀 Використовується Гібридна модель"
        elif ("CRNN модель" in engine_text or "🤖" in engine_text) and CUSTOM_MODEL_AVAILABLE:
            self.use_custom_model = True
            status_msg = "🤖 Використовується CRNN модель"
        else:
            # Якщо обраний двигун недоступний або це Tesseract
            if not ("Tesseract" in engine_text or "📝" in engine_text):
                logger.warning(f"Двигун '{engine_text}' недоступний, активовано Tesseract.")

            status_msg = "📝 Використовується Tesseract"

        logger.info(status_msg)
        self.statusBar().showMessage(status_msg.split('(')[0].strip(), 3000) # Коротке повідомлення в статус бар
        self.update_model_info()


    def update_model_info(self):

        if not hasattr(self, 'model_info_label'): return
        logger.debug("Оновлення model_info_label")

        text = "Інформація про модель недоступна"
        style = "color: red; font-size: 10px; padding: 5px;"

        try:
            if self.use_paddle_ocr and self.paddle_engine:
                info = self.paddle_engine.get_model_info()
                current_lang = info.get('language', 'N/A').upper()
                text = f"🌊 Активна: PaddleOCR\n  • Мова: {current_lang}\n  • GPU: {'Так' if info.get('gpu_enabled') else 'Ні'}\n  • Підтримка:
{info.get('supported_languages', 'N/A')}+ мов"
                style = "color: #00BCD4; font-size: 10px; padding: 5px; font-weight: bold;"
            elif self.use_hybrid_model and self.hybrid_engine:
                info = self.hybrid_engine.get_model_info()
                text = f"🚀 Активна: Гібридна\n  • Словник: {info.get('vocabulary_size', 'N/A')} симв.\n  • Пристрій: {info.get('device', 'N/A').upper()}"
                style = "color: #4CAF50; font-size: 10px; padding: 5px; font-weight: bold;"
```

```
        elif self.use_custom_model and self.custom_engine:
            info = self.custom_engine.get_model_info()
            text = f"🤖 Активна: CRNN\n   • Словник: {info.get('vocabulary_size', 'N/A')} симв.\n   • Пристрій: {info.get('device', 'N/A').upper()}"
            style = "color: #2196F3; font-size: 10px; padding: 5px; font-weight: bold;"
        else: # Tesseract
            tesseract_lang = self.settings.get('language', 'ukr')
            text = f"📝 Активний: Tesseract\n   • Мова: {tesseract_lang}"
            style = "color: #666; font-size: 10px; padding: 5px;"
    except Exception as e:
        logger.error(f"Помилка отримання інформації про модель: {e}", exc_info=True)

    self.model_info_label.setText(text)
    self.model_info_label.setStyleSheet(style)


def add_files(self):

    files, _ = QFileDialog.getOpenFileNames(self, "Виберіть файли", "", "Images (*.png *.jpg *.jpeg *.bmp *.tiff *.tif);;PDF (*.pdf);;All (*.*)")
    if files:
        count = 0
        for file in files:
            if file not in self.image_paths:
                self.image_paths.append(file); self.file_list.addItem(os.path.basename(file)); count += 1
        if count > 0: self.btn_start.setEnabled(True); self.statusBar().showMessage(f"Додано {count} файлів"); logger.info(f"Додано файли: {files}")

def add_folder(self):

    folder = QFileDialog.getExistingDirectory(self, "Виберіть папку")
    if folder:
        added = 0; extensions = ('.png', '.jpg', '.jpeg', '.bmp', '.tiff', '.tif', '.pdf')
        logger.info(f"Сканування папки: {folder}")
        try:
            for file in os.listdir(folder):
                if file.lower().endswith(extensions):
                    fp = os.path.join(folder, file)
                    if fp not in self.image_paths:
                        self.image_paths.append(fp); self.file_list.addItem(file); added += 1
        except Exception as e:
            logger.error(f"Помилка читання папки {folder}: {e}")
            QMessageBox.warning(self, "Помилка", f"Не вдалося прочитати папку:\n{e}")
            return
        if added > 0:
            self.btn_start.setEnabled(True); self.statusBar().showMessage(f"Додано {added} файлів з папки"); logger.info(f"Додано {added} файлів")
        else: QMessageBox.information(self, "Інфо", "Підтримувані файли не знайдено")

def clear_files(self):

    self.image_paths.clear(); self.file_list.clear(); self.btn_start.setEnabled(False)
    self.statusBar().showMessage("Список очищено"); logger.info("Список файлів очищено")

def start_recognition(self):
    if not self.image_paths: QMessageBox.warning(self, "Увага", "Додайте файли"); return
    logger.info("Натиснуто кнопку 'Розпізнати'")

    self.update_settings()


    if self.use_paddle_ocr and self.paddle_engine:
        lang_map_paddle = {'ukr': 'uk', 'eng': 'en', 'ukr+eng': 'en'}
        target_lang = lang_map_paddle.get(self.settings.get('language', 'ukr'), 'uk')
        logger.info(f"Перевірка/встановлення мови PaddleOCR на '{target_lang}'...")
        try:

            self.paddle_engine.change_language(target_lang)
            # Перевіряємо, чи мова дійсно встановлена
            if self.paddle_engine.lang == target_lang:
                logger.info(f"Мову PaddleOCR успішно встановлено/підтверджено: '{target_lang}'")
                self.update_model_info() # Оновлюємо UI
            else:
                raise Exception(f"Не вдалося встановити мову {target_lang}, поточна: {self.paddle_engine.lang}")
        except Exception as e:
            logger.error(f"Критична помилка встановлення мови PaddleOCR '{target_lang}': {e}", exc_info=True)
```

```
            QMessageBox.critical(self, "Помилка PaddleOCR", f"Не вдалося завантажити мовну модель '{target_lang}':\n{e}\n\nСпробуйте іншу мову або двигун.")
            return

        elif self.use_paddle_ocr and not self.paddle_engine:
            logger.error("Критично: Прапорець use_paddle_ocr встановлено, але self.paddle_engine is None!")
            QMessageBox.critical(self, "Помилка конфігурації", "Помилка ініціалізації PaddleOCR. Двигун недоступний.")
            return


        # Блокування UI
        logger.debug("Блокування UI перед запуском worker")
        self.btn_start.setEnabled(False); self.btn_cancel.setEnabled(True)
        self.progress_bar.setValue(0); self.result_text.clear(); self.stats_text.clear()
        if self.export_panel:
           for btn in self.export_panel.findChildren(QPushButton): btn.setEnabled(False)

        # Створення та запуск Worker'a
        logger.debug("Створення екземпляру OCRWorker")
        self.worker = OCRWorker(self.image_paths.copy(), self.settings.copy())

        # Передача інстансів та прапорців двигунів
        logger.debug("Передача двигунів та прапорців у worker")
        self.worker.use_paddle_ocr = self.use_paddle_ocr
        self.worker.paddle_engine = self.paddle_engine if self.use_paddle_ocr else None

        self.worker.use_hybrid_model = self.use_hybrid_model
        self.worker.hybrid_engine = self.hybrid_engine if self.use_hybrid_model else None

        self.worker.use_custom_model = self.use_custom_model
        self.worker.custom_engine = self.custom_engine if self.use_custom_model else None



        logger.debug("Підключення сигналів worker")
        try:

            self.worker.progress.disconnect()
            self.worker.status.disconnect()
            self.worker.finished.disconnect()
            self.worker.error.disconnect()
            logger.debug("Попередні з'єднання сигналів відключено")
        except TypeError:
            logger.debug("Попередніх з'єднань сигналів не було")
        except Exception as e:
            logger.warning(f"Помилка при відключенні сигналів: {e}")

        self.worker.progress.connect(self.update_progress)
        self.worker.status.connect(self.update_status)
        self.worker.finished.connect(self.recognition_finished)
        self.worker.error.connect(self.recognition_error)
        logger.debug("Сигнали worker підключено")

        # Повідомлення про старт
        active_engine_str = self._get_active_engine_string()
        self.statusBar().showMessage(f"Запуск ({active_engine_str})...")
        logger.info(f"Запуск worker з двигуном: {active_engine_str}")

        self.worker.start()
        logger.debug("Worker запущено")

    def _get_active_engine_string(self) -> str:
        if self.use_paddle_ocr and self.paddle_engine: return f"PaddleOCR ({self.paddle_engine.lang.upper()})"
        elif self.use_hybrid_model: return "Гібридна модель"
        elif self.use_custom_model: return "CRNN модель"
        else: return f"Tesseract ({self.settings.get('language', 'N/A')})"


    def cancel_recognition(self):
        if self.worker and self.worker.isRunning():
            reply = QMessageBox.question(self, "Підтвердження", "Зупинити обробку?", QMessageBox.Yes | QMessageBox.No)
            if reply == QMessageBox.Yes:
```

```
            logger.info("Користувач підтвердив скасування")
            self.worker.cancel() # Посилаємо сигнал скасування потоку
            self.btn_start.setEnabled(True) # Дозволяємо запустити знову
            self.btn_cancel.setEnabled(False) # Блокуємо кнопку скасування
            self.statusBar().showMessage("Обробка скасовується...")
            self.status_label.setText("Статус: Скасування...")
        else:
            logger.warning("Спроба скасування, коли worker не запущено або вже зупинено")

    def update_progress(self, value: int):
        self.progress_bar.setValue(value)

    def update_status(self, status: str):
        self.status_label.setText(f"Статус: {status}")
        self.statusBar().showMessage(status, 2000)

    def recognition_finished(self, data: Dict[str, Any]):
        logger.info("Отримано сигнал finished від OCRWorker")
        was_cancelled = not self.btn_cancel.isEnabled()

        self.btn_start.setEnabled(True);
        self.btn_cancel.setEnabled(False);
        self.progress_bar.setValue(100)

        if was_cancelled:
            self.statusBar().showMessage("Обробка скасована");
            self.status_label.setText("Статус: Скасовано")
            logger.info("Обробка була скасована, результати не відображаються");
            return

        results = data.get('results', [])
        self.current_results = results  # Зберігаємо результати

        if results:
            successful_results = [r for r in results if not r.get('error')]
            errors_count = len(results) - len(successful_results)

            if successful_results:
                doc_texts = defaultdict(list)

                for r in successful_results:
                    page = r.get('page', 0)
                    doc_texts[r['image_path']].append((page, r.get('text', '')))

                all_doc_strings = []
                separator = "\n\n==========================================\n\n"

                # Обробляємо кожен документ
                for image_path, pages in doc_texts.items():
                    # Сортуємо сторінки за їх номером
                    pages.sort(key=lambda x: x[0])

                    doc_string_parts = []
                    is_multi_page = len(pages) > 1 or (
                            pages and pages[0][0] > 0)  # Перевіряємо, чи це багатосторінковий PDF

                    for page, text in pages:
                        if is_multi_page:
                            # Додаємо роздільник сторінок для PDF
                            doc_string_parts.append(f"--- (Сторінка {page}) ---\n\n{text}")
                        else:
                            # Не додаємо для звичайних зображень
                            doc_string_parts.append(text)

                    # З'єднуємо частини одного документа
                    doc_text = "\n\n".join(doc_string_parts)
                    all_doc_strings.append(f"--- Документ: {os.path.basename(image_path)} ---\n\n{doc_text}")

                # 3. З'єднуємо всі документи
                all_text = separator.join(all_doc_strings)
```

```
            self.result_text.setText(all_text)
            if successful_results[0].get('image_path'): self.display_image(successful_results[0]['image_path'])
            stats = self.generate_statistics(results);
            self.stats_text.setText(stats)
            if self.export_panel:
                for btn in self.export_panel.findChildren(QPushButton): btn.setEnabled(True)

            status_msg = f"Завершено. Успішно (сторінок): {len(successful_results)}. Помилок (сторінок): {errors_count}."
            self.statusBar().showMessage(status_msg);
            self.status_label.setText("Статус: Завершено")
            QMessageBox.information(self, "Завершено", status_msg);
            logger.info(status_msg)
        else:
            self.statusBar().showMessage(f"Завершено з помилками ({errors_count} файлів).");
            self.status_label.setText("Статус: Завершено з помилками")
            stats = self.generate_statistics(results);
            self.stats_text.setText(stats)
            QMessageBox.warning(self, "Помилка", f"Не вдалося розпізнати жодного файлу.\nДив. 'Статистика'.");
            logger.warning(f"Розпізнавання завершено, але всі {errors_count} файли мали помилки.")
    else:
        self.statusBar().showMessage("Завершено. Немає результатів.");
        self.status_label.setText("Статус: Завершено (немає результатів)")
        QMessageBox.warning(self, "Увага", "Результати відсутні.");
        logger.warning("Розпізнавання завершено, список результатів порожній.")

def recognition_error(self, error_msg: str):
    logger.error(f"Отримано сигнал error від OCRWorker: {error_msg}")
    self.btn_start.setEnabled(True); self.btn_cancel.setEnabled(False)
    self.status_label.setText(f"Статус: Критична помилка!"); self.statusBar().showMessage("Критична помилка!")
    QMessageBox.critical(self, "Критична помилка", f"Помилка в потоці розпізнавання:\n{error_msg}\n\nДив. ocr_app.log.")

def display_image(self, image_path: str):
    # Відображення зображення чи першої сторінки PDF
    logger.debug(f"Attempting to display: {image_path}")
    pixmap_to_display = None
    display_text = f"Перегляд: {os.path.basename(image_path)}"
    error_text = None
    if not image_path or not os.path.exists(image_path):
        error_text = "Файл не знайдено"
        logger.warning(f"Display error: File not found: {image_path}")
    else:
        try:
            if image_path.lower().endswith('.pdf'):

                logger.debug("File is PDF, attempting to render page 0")
```

paddle_ocr_engine.py

```
class PaddleOCREngine:


    def __init__(self, lang='uk', use_angle_cls=True, use_gpu=False, **kwargs):

        if not PADDLEOCR_AVAILABLE:
            logger.error("Спроба ініціалізувати PaddleOCREngine, коли PADDLEOCR_AVAILABLE=False")
            raise ImportError("PaddleOCR не встановлено!")

        self.lang = lang
        self.use_gpu = use_gpu
        self.use_angle_cls = use_angle_cls
        self.ocr = None

        try:
            self._initialize_paddle(lang)
        except Exception as e:
            logger.error(f"Помилка під час початкової ініціалізації PaddleOCR: {e}", exc_info=False)
```

```
    def _initialize_paddle(self, lang_code: str):

        start_time = time.time()
        logger.info(f"Спроба ініціалізації/переініціалізації PaddleOCR: мова={lang_code}, GPU setting={self.use_gpu}, Angle Cls
setting={self.use_angle_cls}")
        try:

            self.ocr = PaddleOCR(
                use_angle_cls=self.use_angle_cls,
                lang=lang_code,

            )
            self.lang = lang_code
            elapsed_time = time.time() - start_time
            logger.info(f"PaddleOCR ({lang_code}) ініціалізовано успішно за {elapsed_time:.2f} сек")
        except Exception as e:
            logger.error(f"Помилка ініціалізації PaddleOCR ({lang_code}): {e}", exc_info=False)
            self.ocr = None
            raise

    def recognize(self, image: Union[str, np.ndarray, Image.Image],
                  return_confidence=False) -> Dict[str, Any]:
        # Розпізнавання тексту на зображенні
        if self.ocr is None:
            logger.error(" PaddleOCR model not initialized!")
            return {'text': "", 'confidence': 0.0, 'error': "PaddleOCR model not initialized", 'processing_time': 0.0}

        start_time = time.time()
        logger.debug(f"-> recognize: lang={self.lang}, type={type(image)}")
        img_input, input_description = self._prepare_input(image)
        if img_input is None:
            elapsed_time = time.time() - start_time
            return {'text': "", 'confidence': 0.0, 'error': f"Invalid input", 'processing_time': elapsed_time}

        logger.debug(f"Input for ocr(): {input_description}")

        try:

            result_pages = self.ocr.ocr(img_input)

            logger.debug(f"Raw result (pages): {str(result_pages)[:500]}...")

            if not result_pages:
                logger.warning("PaddleOCR returned None or empty list")
                elapsed_time = time.time() - start_time
                return {'text': "", 'confidence': 0.0, 'processing_time': elapsed_time}



            all_page_texts = []
            all_confidences = []


            for page_result_list in result_pages:


                texts, confidences = self._parse_ocr_result_new_format(page_result_list)

                all_page_texts.append('\n'.join(texts))
                all_confidences.extend(confidences)

            # З'єднуємо текст з усіх сторінок
            full_text = "\n\n--- (Нова сторінка) ---\n\n".join(all_page_texts)
            avg_confidence = (sum(all_confidences) / len(all_confidences) * 100) if all_confidences else 0.0

            elapsed_time = time.time() - start_time
            logger.info(f"<- recognize OK ({self.lang}): {len(texts)} lines, {elapsed_time:.2f}s")
            return {'text': full_text, 'confidence': avg_confidence, 'processing_time': elapsed_time}

        except Exception as e:
            elapsed_time = time.time() - start_time
```

```
            logger.error(f"Error in recognize ({self.lang}, {elapsed_time:.2f}s): {e}", exc_info=True)
            return {'text': "", 'confidence': 0.0, 'error': str(e), 'processing_time': elapsed_time}


    def recognize_with_boxes(self, image: Union[str, np.ndarray, Image.Image]) -> List[Dict]:

        if self.ocr is None:
             logger.error("recognize_with_boxes: PaddleOCR model not initialized!")
             return []

        start_time = time.time()
        logger.debug(f"-> recognize_with_boxes: lang={self.lang}")
        img_input, _ = self._prepare_input(image)
        if img_input is None: return []

        try:

            result_pages = self.ocr.ocr(img_input)

            logger.debug(f"Raw result (boxes): {str(result_pages)[:500]}...")

            if not result_pages:
                logger.warning("recognize_with_boxes: PaddleOCR returned None or empty list")
                return []


            formatted_results = []
            for page_result_list in result_pages:

                page_boxes = self._parse_ocr_result_new_format_with_boxes(page_result_list)
                formatted_results.extend(page_boxes)


            elapsed_time = time.time() - start_time
            logger.info(f"<- recognize_with_boxes OK ({self.lang}): {len(formatted_results)} boxes, {elapsed_time:.2f}s")
            return formatted_results

        except Exception as e:
            elapsed_time = time.time() - start_time
            logger.error(f" Error in recognize_with_boxes ({self.lang}, {elapsed_time:.2f}s): {e}", exc_info=True)
            return []


    def _prepare_input(self, image: Union[str, np.ndarray, Image.Image]) -> Tuple[Union[str, np.ndarray, None], str]:

        img_input = None; input_description = ""
        try:
            if isinstance(image, str):
                if not os.path.exists(image): logger.error(f"File not found: {image}"); return None, "File not found"
                img_input = image; input_description = os.path.basename(image)
            elif isinstance(image, np.ndarray):
                if len(image.shape) == 3 and image.shape[2] == 3: img_input = image; input_description = f"Numpy BGR {image.shape}"
                elif len(image.shape) == 2: img_input = image; input_description = f"Numpy Gray {image.shape}"
                else: raise ValueError(f"Unsupported numpy shape: {image.shape}")
            elif isinstance(image, Image.Image):
                if image.mode == 'RGB': img_input = cv2.cvtColor(np.array(image), cv2.COLOR_RGB2BGR); input_description = f"PIL RGB {image.size} ->
BGR"
                elif image.mode == 'L': img_input = np.array(image); input_description = f"PIL L {image.size} -> Gray"
                elif image.mode == 'RGBA': img_input = cv2.cvtColor(np.array(image), cv2.COLOR_RGBA2BGR); input_description = f"PIL RGBA
{image.size} -> BGR"
                else: logger.warning(f"Converting PIL {image.mode} -> RGB -> BGR"); img_input = cv2.cvtColor(np.array(image.convert("RGB")),
cv2.COLOR_RGB2BGR); input_description = f"PIL {image.mode} {image.size} -> BGR"
            else: raise ValueError(f"Unsupported image type: {type(image)}")
        except Exception as e: logger.error(f"Input preparation error: {e}", exc_info=False); return None, f"Input prep error: {e}"
        return img_input, input_description


    def _parse_ocr_result_old_format(self, result: list) -> Tuple[List[str], List[float]]:
        texts = []; confidences = []

         return texts, confidences
```

```
def _parse_ocr_result_new_format(self, page_data: dict) -> Tuple[List[str], List[float]]:

    texts = []
    confidences = []
    try:

        if isinstance(page_data, dict):
            rec_texts = page_data.get('rec_texts', [])
            rec_scores = page_data.get('rec_scores', [])

            if len(rec_texts) != len(rec_scores):
                logger.warning(
                    f"Кількість текстів ({len(rec_texts)}) не співпадає з кількістю оцінок ({len(rec_scores)})!")

                num_items = min(len(rec_texts), len(rec_scores))
            else:
                num_items = len(rec_texts)

            for i in range(num_items):
                text = str(rec_texts[i]).strip() if rec_texts[i] else ""
                score = rec_scores[i]
                if text:
                    texts.append(text)

                    conf = max(0.0, min(1.0, float(score))) if isinstance(score, (float, int)) else 0.9
                    confidences.append(conf)
        else:

            logger.warning(f"Результат не схожий на словник, парсинг провалено. Result: {str(page_data)[:100]}...")

    except Exception as e:
        logger.error(f"Помилка парсингу нового формату (словника): {e}", exc_info=False)

    return texts, confidences


def _parse_ocr_result_new_format_with_boxes(self, page_data: dict) -> List[Dict]:

    formatted_results = []
    try:

        if isinstance(page_data, dict):
            rec_texts = page_data.get('rec_texts', [])
            rec_scores = page_data.get('rec_scores', [])
            dt_polys = page_data.get('dt_polys', [])

            # Перевіряємо довжини
            num_items = len(rec_texts)
            if len(rec_scores) != num_items or len(dt_polys) != num_items:
                 logger.warning(f"Неспівпадіння довжин у новому форматі: texts={len(rec_texts)}, scores={len(rec_scores)}, polys={len(dt_polys)}")
                 num_items = min(len(rec_texts), len(rec_scores), len(dt_polys))
            for i in range(num_items):
                text = str(rec_texts[i]).strip() if rec_texts[i] else ""
                score = rec_scores[i]
                poly = dt_polys[i]

                if text:
                    conf = max(0.0, min(1.0, float(score))) if isinstance(score, (float, int)) else 0.9
                    try:

                        if isinstance(poly, np.ndarray) and poly.shape == (4, 2):
                            np_box = poly.astype(int)
                            x_min, y_min = np.min(np_box, axis=0)
                            x_max, y_max = np.max(np_box, axis=0)
                            formatted_results.append({
                                'text': text,
                                'confidence': conf * 100, # У відсотки
                                'box': [int(x_min), int(y_min), int(x_max), int(y_max)]
                            })
                        else: logger.warning(f"Неправильний формат полігону: {poly}")
                    except Exception as box_err:
```

```
                logger.error(f"Помилка обробки полігону {poly}: {box_err}")

        else:
            logger.warning(f"Результат (boxes) не схожий на словник: {str(page_data)[:100]}...")


    except Exception as e:
        logger.error(f"Помилка парсингу нового формату результату (з boxes): {e}", exc_info=False)

    return formatted_results



def change_language(self, lang: str):
    # Зміна мови розпізнавання
    if lang != self.lang or self.ocr is None:
        logger.info(f"Attempting language change: {self.lang} -> {lang}")
        try:
            self._initialize_paddle(lang)
            logger.info(f"Language successfully changed to {self.lang}")
        except Exception as e:
            logger.error(f"Failed to change language to {lang}: {e}. Keeping language: {self.lang}", exc_info=False)
            raise RuntimeError(f"Failed to switch PaddleOCR language to {lang}: {e}") from e
    else:
        logger.debug(f"Language already set to {lang}, no change needed.")


def get_model_info(self) -> Dict[str, Any]:

    return {
        'model_type': 'PaddleOCR',
        'language': self.lang if self.ocr else 'N/A',
        'gpu_enabled': self.use_gpu,
        'angle_cls_enabled': self.use_angle_cls,
        'supported_languages': '80+',
        'is_loaded': self.ocr is not None,
        'features': ['Detection', 'Recognition', 'Angle Correction', '80+ Languages']
    }
```

ДОДАТОК Е: Результати перевірки роботи на співпадіння

StrikePlagiarism.com

Дата звіту 12/4/2025
Дата редагування 12/5/2025

Документ прийнятий

Звіт подібності

Метадані

Назва організації
National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute
Заголовок
ІП-з41мп_Дишкант_ПЗ
Автор Науковий керівник / Експерт
ДишкантКрамар Ю.М.
підрозділ
ФІОТ, К-а інформатики та програмної інженерії

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

9533
Кількість слів

75923
Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

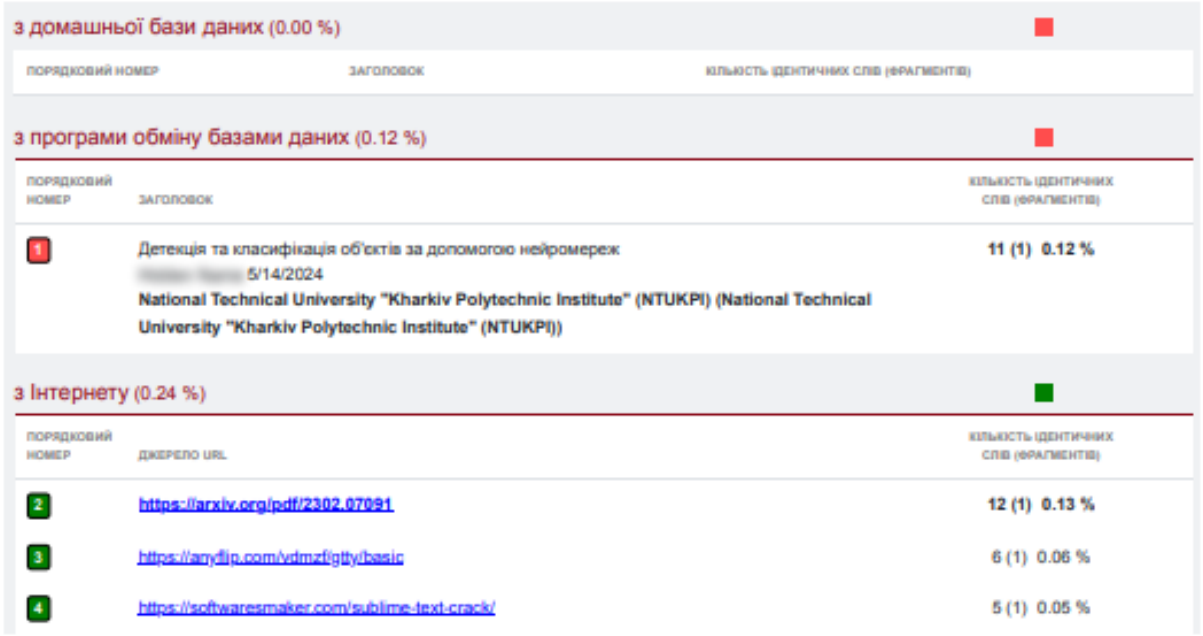
Заміна букв		1
Інтервали		0
Мікропробіли		0
Білі знаки		0
Парафрази (SmartMarks)		2

Джерела

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізуаати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://arxiv.org/pdf/2302.07091	12 0.13 %
2	Детекція та класифікація об'єктів за допомогою нейромереж [redacted] 5/14/2024 National Technical University "Kharkiv Polytechnic Institute" (NTUKPI) (National Technical University "Kharkiv Polytechnic Institute" (NTUKPI))	11 0.12 %
3	https://anyflip.com/vdmzf/gtty/basic	6 0.06 %
4	https://softwaresmaker.com/sublime-text-crack/	5 0.05 %



з домашньої бази даних (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з програми обміну базами даних (0.12 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Детекція та класифікація об'єктів за допомогою нейромереж [redacted] 5/14/2024 **National Technical University "Kharkiv Polytechnic Institute" (NTUKPI) (National Technical University "Kharkiv Polytechnic Institute" (NTUKPI))**	**11 (1) 0.12 %**

з Інтернету (0.24 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
2	https://arxiv.org/pdf/2302.07091	**12 (1) 0.13 %**
3	https://anydip.com/vdmzfigtty/basic	6 (1) 0.06 %
4	https://softwaresmaker.com/sublime-text-crack/	5 (1) 0.05 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)