Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)
Навчально-науковий інститут комп'ютерних наук та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))
Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до дипломної роботи

перший (бакалаврський)
(рівень вищої освіти)

на тему: Розроблення інтелектуального застосунку для автоматизованого розпізнавання та структурування даних з PDF документів та зображень

Виконала: студентка 4 курсу групи КН-41
спеціальності
122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Максимова Є.П.
(прізвище та ініціали)

Керівник Лукащук Б.С.
(прізвище та ініціали)

Керівник Процик Ю.С.
(прізвище та ініціали)

Рецензент Флуд Л.О.
(прізвище та ініціали)

Львів – 2025

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Спеціальність 122 "Комп'ютерні науки"
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
Борецька І. Б.
"10" червня 20 25 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Максимовій Єлизаветі Павлівні
(прізвище, ім'я, по батькові)

1. Тема роботи *Розроблення інтелектуального застосунку для автоматизованого розпізнавання та структурування даних з PDF документів та зображень*
керівник роботи *Лукащук Богдан Сергійович, асистент*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
керівник роботи *Процик Юрій Степанович, к. ф-м. н. доцент кафедри комп'ютерних наук*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від *15.11.2024 року №С-882*
2. Термін подання студентом роботи *10.06.2025р.*
3. Вихідні дані до роботи: Веб-додаток, що виконує функцію конвертації файлів з розширенням PDF та JPG, JPEG та ін. у текст та/або таблицю
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Стан проблемної області, інформаційне та математичне забезпечення, програмне та технічне забезпечення, висновки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація до диплому.

6. Дата видачі завдання *18.11.2024р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел	*18.11.2024р. – 16.12.2024р.*	Виконано
2	Аналіз проблемної області та постановка задачі	*16.12.2024р. – 16.12.2024р.*	Виконано
3	Проектування інформаційного та математичного забезпечення	*01.01.2025р. – 10.05.2025р.*	Виконано
4	Програмна реалізація проекту	*28.01.2025р. – 10.05.2025р.*	Виконано
5	Тестування та усунення помилок програмної реалізації	*10.05.2025р. – 05.06.2025р.*	Виконано
6	Оформлення записки до дипломного проекту	*05.06.2025 – 10.06.2025*	Виконано

Студент _____ (підпис) **Максимова Є.П.** (прізвище та ініціали)

Керівник роботи _____ (підпис) **Лукащук Б.С.** (прізвище та ініціали)

Керівник роботи _____ (підпис) **Процик Ю.С.** (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота містить 65 сторінок, 18 ілюстрацій, 2 таблиці, 3 додатки, 15 джерел.

У роботі описано етапи розробки та впровадження інтелектуального застосунку для автоматизованого розпізнавання й структурування даних із PDF-документів та зображень. Застосунок використовує попередню обробку зображень за допомогою OCR-рушіїв (Tesseract, PaddleOCR), а також нейромережні моделі через API OpenAI та Anthropic для розуміння даних. Результати перетворюються у формат Excel із збереженням вихідного макету.

Запропоноване рішення дозволяє значно скоротити час ручного введення даних і може застосовуватися в фінансових, юридичних, інженерних компаніях та дослідницьких центрах.

Ключові слова: автоматизоване розпізнавання даних, OCR, PDF2Image, Tesseract, PaddleOCR, OpenAI API, Anthropic API, Python, Flask, layout-parser, pandas, експорт у Excel, структурування даних.

ABSTRACT

The thesis consists of 65 pages, 18 illustrations, 2 tables, 3 appendices, and 15 sources.

This work presents the stages of development and implementation of an intelligent application for automated data recognition and structuring from PDF documents and images. The application employs image preprocessing, OCR engines (Tesseract, PaddleOCR), and neural models accessed via OpenAI and Anthropic APIs to interpret data. The extracted information is exported to Excel while preserving the original layout.

The proposed solution significantly reduces manual data entry time, and can be integrated into financial, legal, engineering firms, and research centers.

Keywords: automated data recognition, OCR, PDF2Image, Tesseract, PaddleOCR, OpenAI API, Anthropic API, Python, Flask, layout-parser, pandas, Excel export, data structuring.

ТЕХНІЧНЕ ЗАВДАННЯ

Розробити програмно-апаратний комплекс Conveer для автоматизованої конвертації PDF-документів та растрових зображень (JPEG, PNG) у структурований табличний формат Excel із збереженням макету.

Використати такі технології:

- мова програмування Python із застосуванням Flask для побудови RESTful API
- бібліотека OpenCV для попередньої обробки зображень
- Tesseract OCR (або PaddleOCR) для розпізнавання тексту
- OpenAI GPT-4o-mini (або Claude) для корекції складних випадків
- бібліотека openpyxl для створення файлів Excel
- HTML/CSS/JavaScript для веб-інтерфейсу.

Дана система повинна виконувати такі основні функції:

- Завантаження PDF-файлів та растрових зображень через веб-інтерфейс.
- Попереднє оброблення зображення: визначення табличних регіонів, деск'юінг та бінаризація.
- Виконання OCR-розпізнавання тексту з отриманого зображення.
- Відновлення структури таблиці за допомогою аналізу координат рядків і колонок.
- Корекція помилок OCR у складних клітинках із використанням LLM.
- Формування проміжного JSON із координатами клітинок, текстом і інформацією про форматування.
- Генерація файлу Excel (.xlsx) із коректним розташуванням тексту, об'єднаними клітинками та стилями.
- Надання результату користувачеві для завантаження через веб-інтерфейс.

Після розробки підготувати детальний звіт, що описуватиме функціональні та технічні аспекти реалізації кожної функції.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

API (Application Programming Interface) – інтерфейс програмування додатків.

CER (Character Error Rate) – відсоток помилок розпізнаних символів.

CSV (Comma-Separated Values) – текстовий формат даних із розділенням комами.

DPI (Dots Per Inch) – кількість крапок на дюйм, показник роздільної здатності зображення.

GPU (Graphics Processing Unit) – графічний процесор для обчислень із паралельною обробкою даних.

HTML (HyperText Markup Language) – мова розмітки веб-сторінок.

HTTP (HyperText Transfer Protocol) – протокол обміну гіпертекстом між клієнтом і сервером.

JSON (JavaScript Object Notation) – текстовий формат обміну даними у вигляді об’єктів.

OCR (Optical Character Recognition) – оптичне розпізнавання символів.

PDF (Portable Document Format) – формат документа Adobe для збереження фіксованого макету.

PNG (Portable Network Graphics) – растровий формат зображень із підтримкою прозорості.

REST (Representational State Transfer) – архітектурний стиль взаємодії в мережі за допомогою HTTP.

UI (User Interface) – інтерфейс користувача.

UX (User Experience) – досвід користувача при взаємодії з програмою.

ВСТУП

Актуальність

У сучасному інформаційному середовищі щодня зростає кількість PDF-документів і растрових зображень (JPEG, PNG тощо), які містять табличні дані різного призначення: від фінансових звітів і медичних карток до наукових досліджень та юридичних договорів. Ручне вилучення й обробка такої інформації потребують значних тимчасових і трудових ресурсів, а також часто супроводжуються помилками OCR і неточним відтворенням структури таблиць. Невисока швидкодія та недосконаліть існуючих рішень призводять до втрат часу, зниження якості даних і ускладнюють автоматизацію бізнес-процесів. У зв’язку з цим виникає необхідність у створенні універсального конвеєрного застосунку Conveer, який забезпечить повністю автоматизовану обробку PDF-файлів та зображень із відновленням табличних структур і формуванням результату у форматі Excel.

Об’єкт дослідження

Об’єктом дослідження є покроковий процес обробки вхідних файлів: передобробка зображень (визначення табличних регіонів, виправлення нахилу, бінаризація), OCR-розпізнавання, групування текстових елементів у рядки й стовпці, корекція результатів із використанням мовних моделей та експорт у формат Excel із збереженням макету.

Предмет дослідження

Програмно-апаратний комплекс для автоматичного вилучення та структурування табличних даних із PDF-файлів і растрових зображень.

Мета роботи

Метою роботи є розробка та впровадження прототипу Conveer, який перетворює PDF-документи та растрові зображення на коректно відформатовані Excel-таблиці з врахуванням позиційування клітинок, об’єднаних комірок і стилізації.

Завдання

Для досягнення мети поставлено такі завдання:

- Проаналізувати сучасний розвиток технологій автоматизованої обробки документів: огляд ринку, існуючих методів OCR і конвертації таблиць, а також основні проблемні аспекти.
- Спроєктувати модуль попередньої обробки зображень, що виконуватиме вирівнювання (deskew), видалення шуму та бінаризацію.
- Розробити бекенд на Python/Flask із абстрактним інтерфейсом для підключення OCR-движків (Tesseract, PaddleOCR) та мовних моделей (OpenAI GPT-4o-mini, Claude).
- Реалізувати алгоритми пост-обробки OCR-результатів: групування координатних даних у рядки та стовпці, згладжування артефактів.
- Забезпечити генерацію проміжного JSON-файлу з координатами клітинок, текстом та інформацією про форматування.
- Розробити модуль експорту даних у формат Excel (.xlsx) із підтримкою об’єднання клітинок, вирівнювання тексту та стилізації.
- Створити фронтенд-інтерфейс на HTML/CSS/JavaScript для завантаження файлів, вибору моделі обробки та відображення статусу конвертації.
- Провести інтеграційне тестування прототипу на наборах документів різних типів (чорно-білі PDF, кольорові скани, фотографії) для оцінки точності й ефективності.

Практичне значення

Створений конвертер може бути впроваджений у фінансових службах, юридичних та медичних установах для автоматизації обробки великих обсягів табличних даних. Автоматичне конвертування документів у готові Excel-файли значно скорочує час підготовки звітності, підвищує якість даних і забезпечує єдині стандарти зберігання інформації. Крім того, запроваджений підхід створює основу для подальшого розширення функціоналу систем обробки документів із використанням сучасних мовних моделей і методів машинного навчання.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Огляд сучасних систем OCR і їх обмеження

Упродовж останнього десятиліття системи оптичного розпізнавання символів (OCR) активно розвиваються, спонукаючи до широкого застосування у різних сферах. Серед найпоширеніших рішень варто відзначити Tesseract [3] і PaddleOCR. Tesseract, створений компанією Hewlett-Packard і подальше підтримуваний Google, забезпечує розпізнавання тексту в растрових документах та широко застосовується завдяки відкритому коду й активній спільноті. Однак у разі низької якості сканованих зображень або фотознімків, а також складних макетів із багатьма шрифтами система демонструє знижену точність (див. рисунок 1.1).

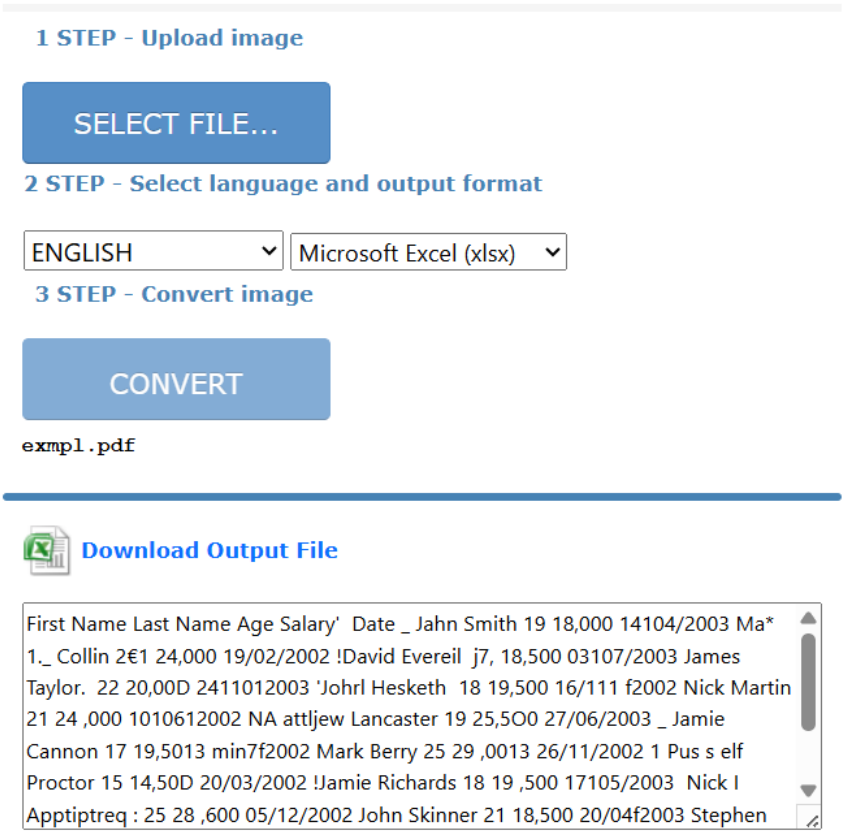


Рисунок 1.1 - Інтерфейс безкоштовного онлайн-конвертера www.onlineocr.net

У таких випадках помилки реєструються не тільки при відокремленні символів, але й на рівні слів і рядків (див. рисунок 1.2).

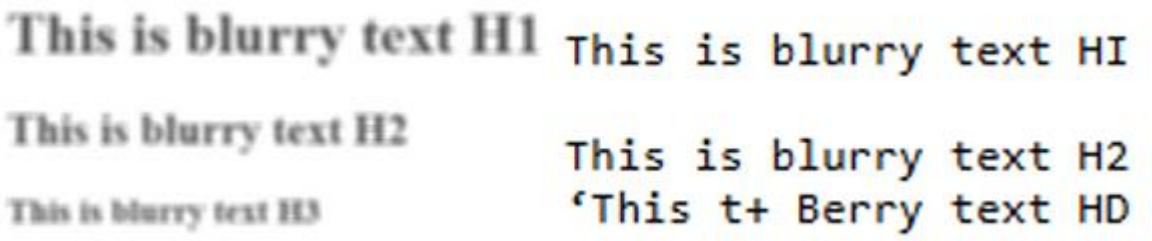


Рисунок 1.2 - Приклад помилок OCR у Tesseract на низькоякісному зображенні

PaddleOCR, розроблений Baidu, пропонує більш сучасні архітектури нейромереж і підвищену чутливість до різних шрифтів, проте вимагає значних обчислювальних ресурсів та складнішого налаштування середовища. Найбільшою проблемою для обох систем є робота з нерівними освітленням, деякі рукописні елементи або текст, розташований на нестандартному фоні. Крім того, при розпізнаванні табличних документів класифікація символів відбувається без урахування просторової структури клітинок таблиці, що спричиняє розпорошення даних і складнощі при подальшому відновленні табличного формату. Ці обмеження позначаються на точності кінцевого результату та ставлять завдання комбінованого підходу, що поєднає традиційні OCR-алгоритми з додатковими методами аналізу графічного оточення символів.

1.2. Існуючі підходи до розпізнавання та збереження табличної структури

Відновлення табличної структури вимагає обробки не лише текстового, але й графічного шару документа. Класичні методи ґрунтуються на двійковому морфологічному аналізі: передусім здійснюють бінаризацію зображення за допомогою адаптивної порогової сегментації (Otsu [1]), а потім застосовують операції розмиву й ерозії для підкреслення тонких ліній сітки. Далі алгоритм Hough Transform [2] виявляє прямі, які відповідають межам таблиці. Після цього координати горизонтальних і вертикальних ліній групують у рядки й стовпці, формуючи двовимірну решітку. Проте для багатоколонкових таблиць або документів зі складними розмітками, наприклад коли між клітинками відсутні чіткі розділові лінії,

класичні методи показують непостійні результати та іноді накладають сусідні клітинки. У сучасних розробках дедалі частіше застосовують глибинні нейромережі, призначені для виявлення табличних структур. Приклади таких рішень – TableNet , DeepDeSRT , які навчаються на анотаціях із великими наборів даних і здатні виявляти контури табличних областей навіть за відсутності явних графічних меж. Водночас точність цих моделей залежить від якості навчальних даних і може знижуватися в разі розпізнавання таблиць із нетиповими шрифтами, складними фонованими елементами чи примусовими колонками. Виходячи з цього, крім нейромережевих підходів, у системі Conveer використовують комбіновану стратегію: спершу застосовується класична морфологія для первинного виділення регіонів таблиць, а далі - алгоритми кластеризації координат пікселів і елементів OCR-виводу, щоб групувати символи в клітинки з урахуванням відстаней між сусідніми елементами.

1.3. Аналіз систем, орієнтованих на конвертацію документів

На ринку існує кілька готових програмних продуктів для автоматичної конвертації документів у різні формати. Серед комерційних лідерів, які орієнтуються на OCR та збереження макету, можна виокремити ABBYY FineReader і Adobe Acrobat Pro (див. рисунок 1.3).

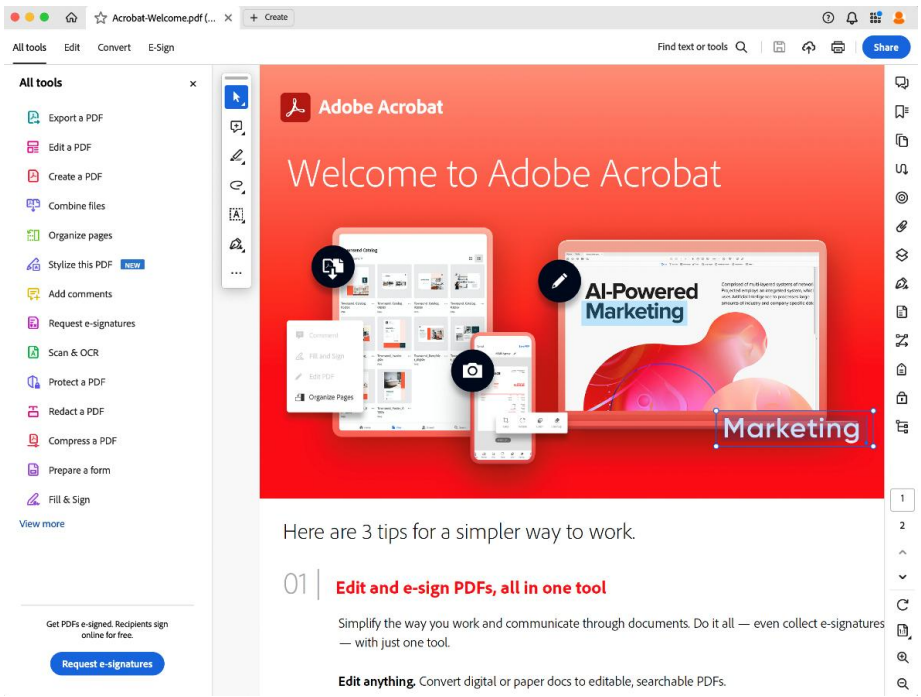


Рисунок 1.3 - Інтерфейс Adobe Acrobat

ABBYY FineReader гарантує високу точність розпізнавання тексту та синтаксичну корекцію, адаптується до різних мов і шрифтів, але його ліцензійні витрати роблять складним масштабне впровадження невеликими підприємствами. Adobe Acrobat Pro, крім OCR-функціоналу, пропонує гнучку роботу з PDF, інструменти редагування та експорту таблиць у Excel. Проте автоматичне визначення складних багатоколонкових структур зазнає труднощів, коли таблиці мають неявні межі або графічні елементи. З open source рішень найбільш поширені Tabula [13] та Camelot [14], які призначені виключно для PDF. Tabula добре справляється з простими таблицями, але не завжди відновлює правильну ієрархію клітинок у складних документах. Camelot підтримує режим Lattice (робота з лініями) і Stream (робота з відстанями), але потребує ручного налаштування параметрів, що ускладнює автоматизацію. Універсальні веб-сервіси (наприклад, Smallpdf, PDFtoExcel) пропонують швидке конвертування, але обмежені в кількості сторінок безкоштовно (див. рисунок 1.4) та вимагають інтернет-з'єднання.

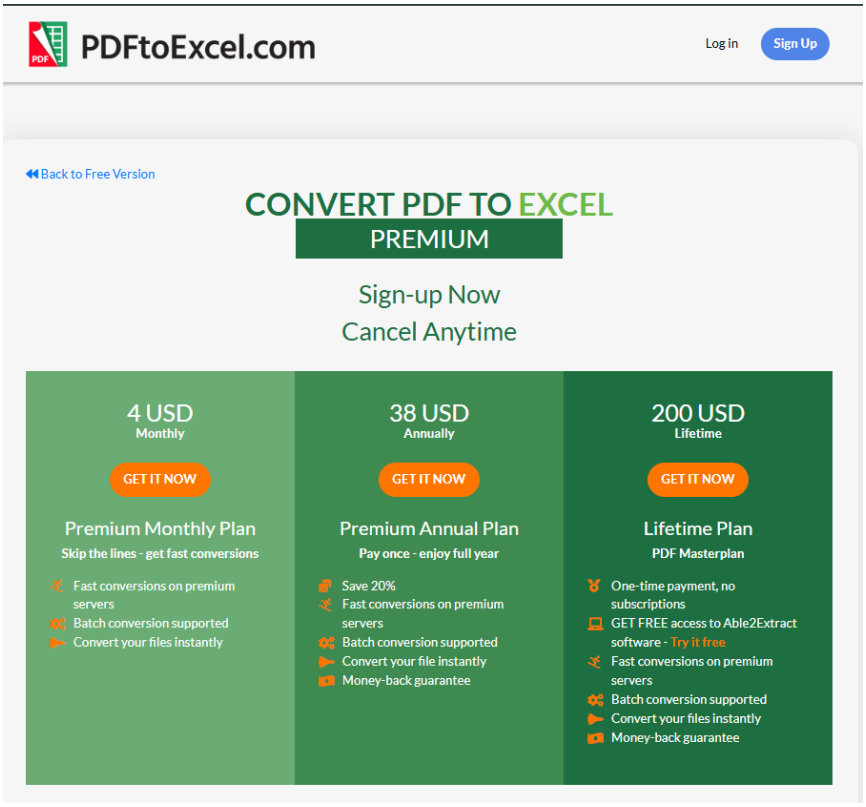


Рисунок 1.4 - Платні плани сайту pdftoexcel.com

Таким чином, серед ключових недоліків багатьох існуючих систем - обмежене налаштування алгоритмів під конкретні вхідні дані, висока вартість ліцензій та необхідність підключення до мережі для хмарних сервісів. Conveer вирішує ці проблеми, пропонуючи відкритий код, можливість локального розгортання та налаштування параметрів конвеєра під різні типи документів.

1.4. Проблемні аспекти й обґрунтування напрямку власного дослідження

Аналіз існуючих засобів обробки документів дозволяє виокремити низку ключових проблемних аспектів. Перш за все, збереження точного макету таблиць ускладнюють нестандартні розмітки, коли між клітинками відсутні чіткі лінії або вони мають перекошену орієнтацію через нерівномірне сканування. Наявні колонки зливаються в єдиний блок, а відсутність контрасту між кольором тексту й фоном призводить до неправильного визначення меж. Окрім того, у документах, створених зі старих архівних джерел, часто трапляються розмиті зображення з низькою роздільною здатністю, що потребує вдосконалених методів передобробки (видалення шуму, покращення контрасту, вирівнювання перспективи).

Сучасні LLM-моделі демонструють здатність до виправлення помилок OCR на рівні сенсу, але не завжди точно відновлюють локальні особливості форматування. Тому необхідний гібридний підхід: класичні алгоритми комп’ютерного бачення працюють на етапі виділення регіонів і корекції зображення, а мовні моделі - на рівні семантики, виправляючи неточності розпізнавання.

Для Conveer обґрунтовано обрати поетапну архітектуру, що складається з модулів попередньої обробки, OCR, сегментації та LLM-корекції. Така схема дає змогу адаптуватися до різних умов вхідних даних, забезпечити масштабованість і підтримати локальне розгортання без необхідності хмарних сервісів. Враховуючи всі перелічені чинники, власне дослідження спрямоване на розробку гнучкого рішення з можливістю налаштування параметрів кожного етапу, що дозволить отримувати високоточний результат з відтворенням макету табличних структур у Excel.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Інформаційна модель

У системі Conveer основними сутностями є: *Document, Page, TextElement, Row, Cell*.

Document – контейнер для одного PDF-файла чи растрового зображення (JPEG, PNG), що надходить на обробку. Має атрибути: document_id, file_path, file_type (PDF або Image), page_count.

Page – унікальний аркуш документа, що містить геометричні параметри (width, height), кольорові канали або сірий канал у вигляді матриці пікселів та розпізнаний масив TextElement. Атрибути: page_number, resolution, orientation.

TextElement – результат виклику OCR (Tesseract або PaddleOCR), що включає координати (x, y, width, height), текстовий вміст text, поріг довіри confidence, page_id.

Row – група TextElement із близьким значенням координат по осі Y; відповідає одному рядкові перекладених символів. Має поля: row_id, page_id, top_coordinate, список TextElement.

Cell – окрема клітинка таблиці, утворена на основі перетину груп по X та Y; складається з атрибутів: cell_id, координати меж (x, y, width, height), список належних елементів TextElement, атрибути стилю (alignment, merged, font_style), індекси row_index, col_index.

Розглянемо UML-діаграму класів (див. рисунок 2.1), що ілюструє зв’язки між сутностями.

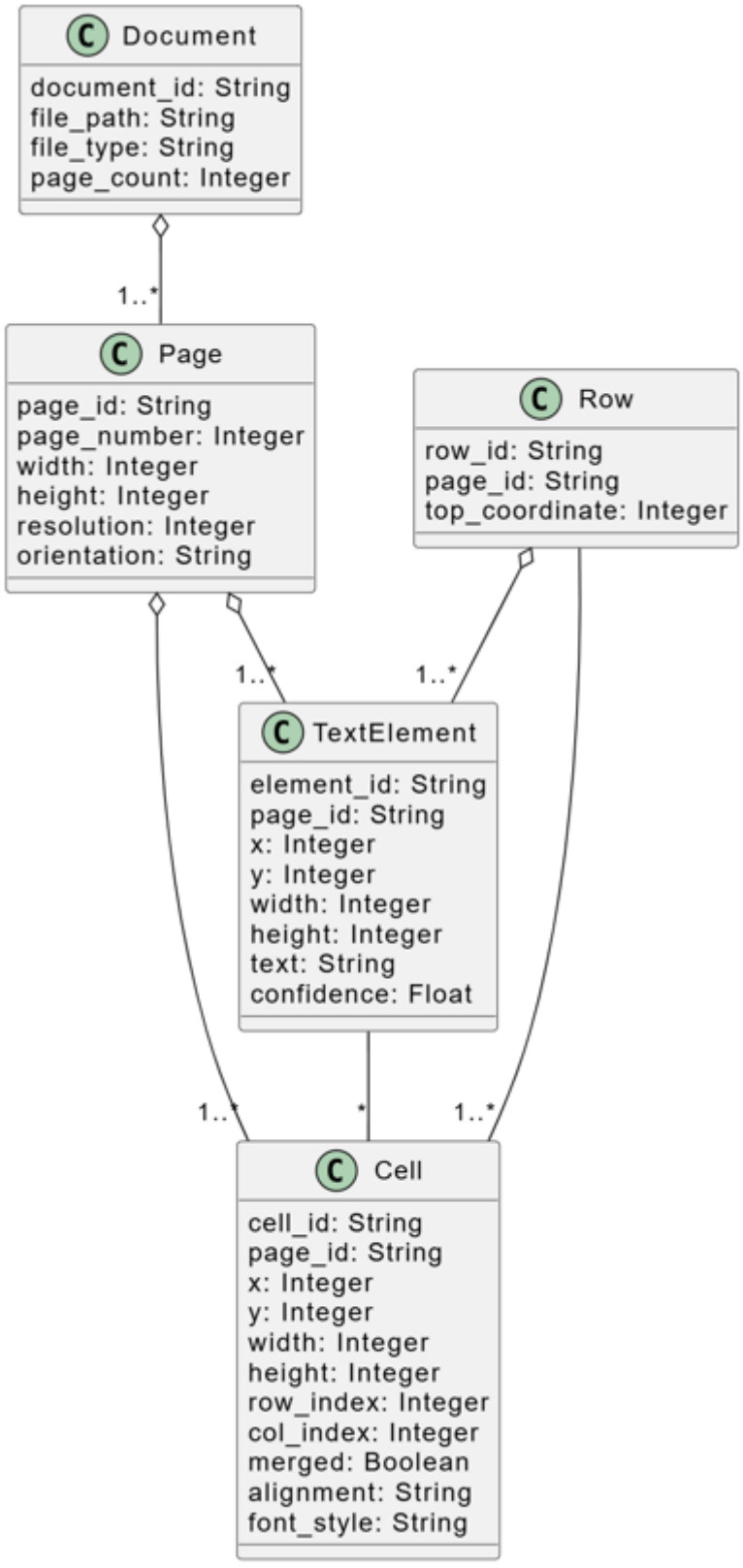


Рисунок 2.1 – UML-діаграма класів інформаційної моделі Conveer

Document агрегує декілька Page (агрегація 1–N).

Page асоціюється з багатьма TextElement (аварійне зв’язування), оскільки кожний фрагмент текста належить конкретному аркушу.

Row формуються з груп TextElement за горизонтальною близькістю координат, а Cell – із перетину груп по рядках і стовпцях.

Cell зберігає посилання на набір TextElement, що входять до меж цієї клітинки.

ER-діаграма для проміжного JSON

У якості проміжного формату даних система використовує структуру на основі JSON. Щоб візуалізувати відношення сутностей у реляційній моделі, наведемо ER-діаграму (див. рисунок 2.2), де описано таблицю **Cells** із полями:

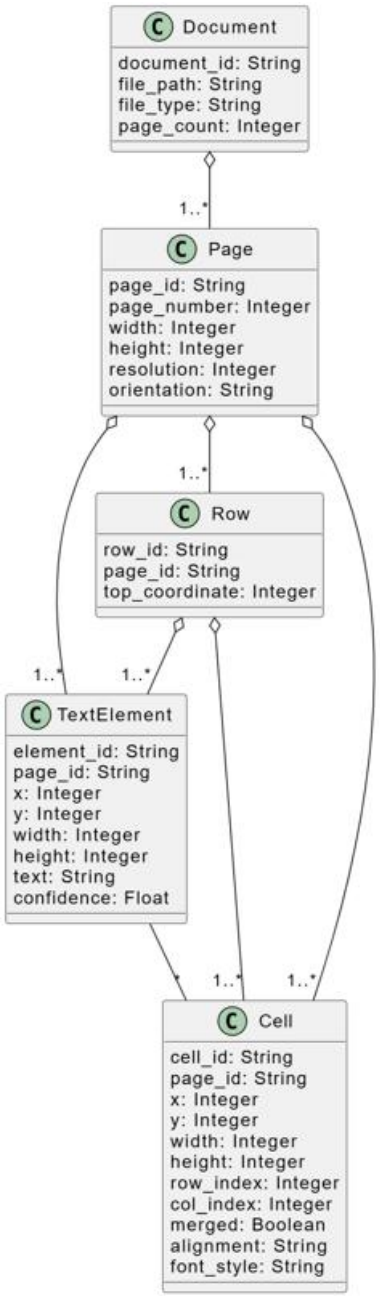


Рисунок 2.2 – ER-діаграма бази даних проміжного JSON

- document_id (FK)
- page_number (частина мультиключа)
- cell_id (PK)
- x, y, width, height
- text
- style
- row_index, col_index

Зв’язки:

Document (1) - Page (N) через атрибут page_number і document_id.

Page (1) - Cells (N) через document_id і page_number.

Cells (N) - TextElement (M) (через додаткову таблицю cell_text_elements, яка містить cell_id та text_element_id).

2.2 Обмеження на вхідні та вихідні дані

Вхідні дані мають відповідати таким критеріям:

Роздільна здатність (DPI): не нижче 300 dpi (Dots Per Inch). При 150 dpi або нижче суттєво зростає похибка розпізнавання, зменшується якість зондування країв та сегментації.

Контрастність тексту та фону: текст має бути темнішим за фон із мінімальним коефіцієнтом 20 % контрасту.

Чіткість границь: наявність видимих ліній таблиці або достатній контраст між полями клітин.

Формат: підтримуються PDF (версія 1.4 і вище, із вбудованими растровими шарами) та растрові зображення (JPEG, PNG).

Обмеження на вихідний формат JSON і Excel

JSON:

- Поля x, y, width, height мають цілі значення (цілодобові координати в пікселях).
- Текст text - не менше одного символу й не більший за довжину width / character_width приблизно.

- style - об’єкт із полями: alignment («left», «center», «right»), merged (булеве значення), font_style (шрифт, розмір).
- row_index, col_index - індекси, починаючи від нуля, що визначають розташування клітинки у матриці.

Excel (.xlsx):

- Повинні відновлюватися розміри клітинок на основі полів width, height (конвертовано в одиниці Excel — column_width і row_height).
- Об’єднані клітинки створюються за наявності у JSON поля merged=true та визначених меж x, y, width, height.
- Вирівнювання тексту у клітинці встановлюється згідно з полем alignment.

2.3 Моделі сегментації та виділення табличних областей

Для отримання чіткої кількісної карти пікселів використано адаптивну бінаризацію:

Опис методу: на кожну локальну ділянку (вікно 15×15) застосовується поріг, рівний середньому значенню яскравості мінус C (параметр, що визначається експериментально, наприклад, C=5).

Псевдокод:

```
For each pixel (i, j) in gray_image:
   window = gray_image[i - k : i + k, j - k : j + k]
   T = mean(window) - C
   if gray_image[i, j] > T:
      binary[i, j] = 255  # білий
   else:
      binary[i, j] = 0    # чорний
```

Критичний аналіз: адаптивна бінаризація є стійкою до нерівномірного освітлення, але підвищує шум на текстурованих фонах (див. рисунок 2.3). Вибір параметра k (розмір вікна) та C суттєво впливає на результат: занадто велике вікно згладжує деталі; занадто мале - виділяє зайві артефакти.

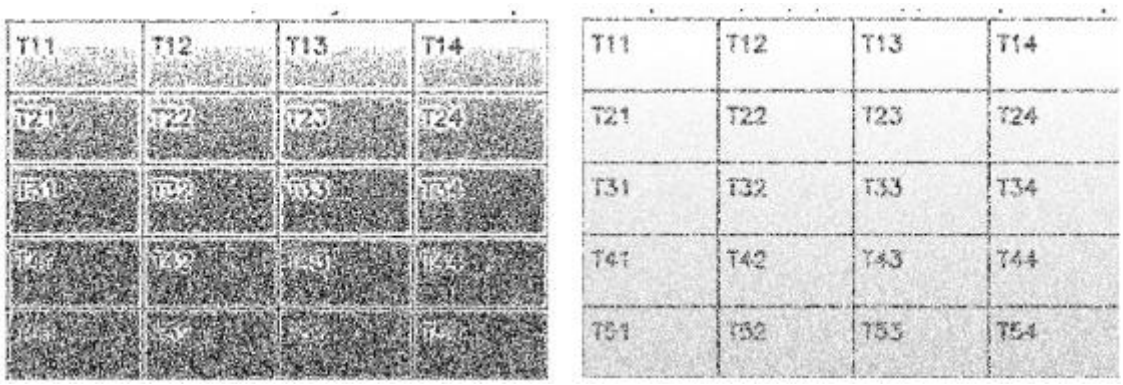


Рисунок 2.3 – Приклад застосування адаптивної бінаризації

Пояснення: (a) вихідна сіре зображення таблиці, (b) результат бінаризації з чіткими лініями та текстом.

Морфологічні операції для визначення контурів

Для підсилення ліній таблиці застосовано морфологічні оператори:

Dilation (cv2.dilate): розширює білі зони, об’єднуючи розірвані фрагменти ліній.

Erosion (cv2.erode): видаляє тонкі шумові елементи та відокремлює літери.

Opening (cv2.morphologyEx з MORPH_OPEN): поєднання erosion та dilation, що видаляє дрібний шум.

Closing (cv2.morphologyEx з MORPH_CLOSE): поєднання dilation та erosion, що “замикає” пропуски у лініях.

Псевдокод морфологічної послідовності:

kernel = structuring_element(shape="rectangle", size=(15, 5))

closed = morphologyEx(binary_image, MORPH_CLOSE, kernel)

Рисунок 2.4 – Результат морфологічного закриття

Пояснення: (a) двійкове зображення до бінаризації, (b) після операції MORPH_CLOSE

Декларація алгоритму Hough Transform

Для детекції прямих, що відповідають межам клітинок, використовується **детектор Хафа** (див. рисунок 2.5).

Опис методу: у просторі параметрів (ρ, θ) кожна точка контурів голосує за можливі прямі. Лінії, які набрали більше голосів (вище порогу), вважаються присутніми.

Параметри:

- rho_resolution = 1 піксель
- theta_resolution = π/180 (1°)
- threshold = 100 (мінімальна кількість голосів)

Псевдокод:

lines = HoughLines(closed_image, rho=1, theta=π/180, threshold=100)

For each line (ρ, θ):

compute (x1, y1), (x2, y2) на межах зображення

draw line на окремому полотні

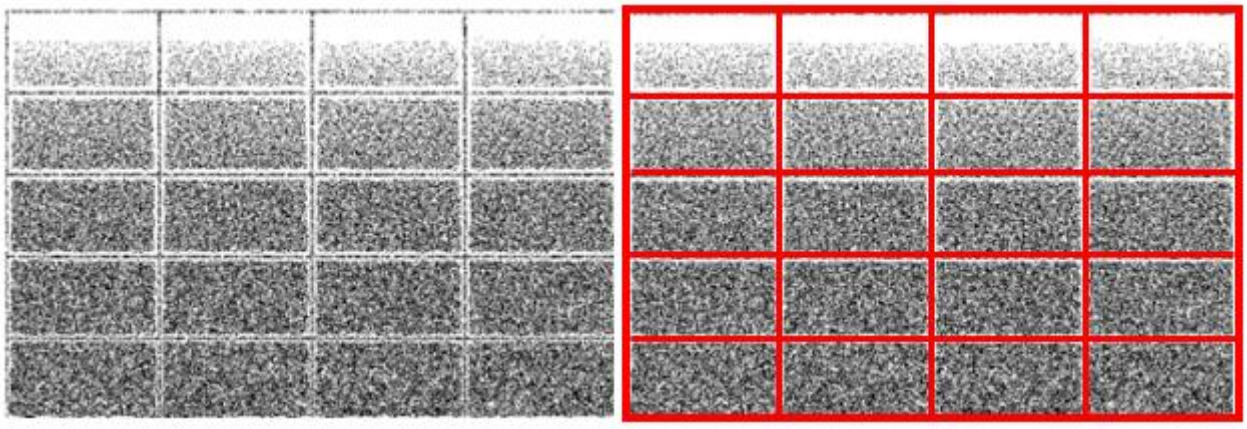


Рисунок 2.5 – Результат детекції ліній за методом Хафа

Пояснення: (a) вихідне двійкове зображення, (b) накладені на нього прямі Хафа (горизонтальні та вертикальні).

Формування каркасу таблиці

Після отримання списку горизонтальних та вертикальних ліній виконується їх групування:

Групування горизонтальних ліній: сортування за координатою Y, об’єднання сусідніх ліній, відстань між якими менша за y_merge_threshold.

Групування вертикальних ліній: аналогічне сортування за X, об’єднання за x_merge_threshold.

Побудова клітинок: перетин кожної горизонтальної групи з кожною вертикальною, що дає координати x, y, width, height.

Псевдокод групування:

sorted_horizontal = sort(horizontal_lines, by="y")

merged_horizontal = merge_lines(sorted_horizontal, y_merge_threshold)

sorted_vertical = sort(vertical_lines, by="x")

merged_vertical = merge_lines(sorted_vertical, x_merge_threshold)

For each h in merged_horizontal:

For each v in merged_vertical:

cell_rect = Rectangle(x=v.x, y=h.y, width=v.width, height=h.height)

add cell_rect до list_of_cells

Коментар до відновлення каркасу без явних меж

У документах зі зниклими або неповними межами клітинок метод Хафа може не спрацювати. У таких випадках варто застосувати **OCRModule**:

На основі координат **TextElement** виявляються області з регулярним розташуванням тексту (рівні інтервали по Y і X).

Формується «імовірнісний каркас» таблиці, де межі задаються як середини інтервалів між близькими текстовими елементами.

2.4 Відновлення табличної структури на основі кластеризації

Схема кластеризації текстових елементів

Після виділення каркасу таблиці результат Хафа (якщо він присутній) або «імовірнісний каркас» передається до модуля **TableDetector**, який об’єднує текст у **Row** та **Cell**.

Кластеризація за координатою Y (рядки) (див. рисунок 2.6):

1. Обчислення середньої висоти слова:

mean_height = df['height'].mean()

y_threshold = int(0.7 * mean_height)

2. Ініціалізація порожнього списку lines = [] та множини used = set().
3. Для кожного слова (TextElement) із DataFrame:
   - Якщо індекс не у used, створити новий список line = [word1].
   - Перебрати інші слова (word2), і якщо abs(word1.top – word2.top) < y_threshold, додати word2 до line та позначити в used.
   - Додати idx1 до used і lines.append(line).
4. Сортування списку lines за мінімальним top у кожному рядку.

Рисунок 2.6 – Приклад кластеризації за Y-координатою

Визначення центрів колонок (кластеризація за X)

Після формування горизонтальних груп перший (заголовний) рядок аналізується як джерело розташування стовпців:

Для кожного слова word обчислюється центр на X: center = left + width/2.

Список column_centers = sorted([center_i for i in header_words]).

Далі кожен елемент наступних рядків обчислюється аналогічним чином (center) і відноситься до найближчого центру стовпця, якщо відстань abs(center – column_centers[j]) < x_threshold (де, наприклад, x_threshold = 60). Таким чином формується двовимірна матриця table_data[row_index][col_index] = clean_text(word.text).

Псевдокод кластеризації за X:

header_words = DataFrame(lines[0]).sort_values(by='left')

column_centers = [left + width/2 for each word in header_words]

For line_index, line in enumerate(lines):

row_data = [""] * len(column_centers)

For each word in line:

center = word.left + word.width/2

closest_col = argmin(abs(center - column_centers))

if abs(center - column_centers[closest_col]) < x_threshold:

row_data[closest_col] = clean_text(word.text)table_data.append(row_data)

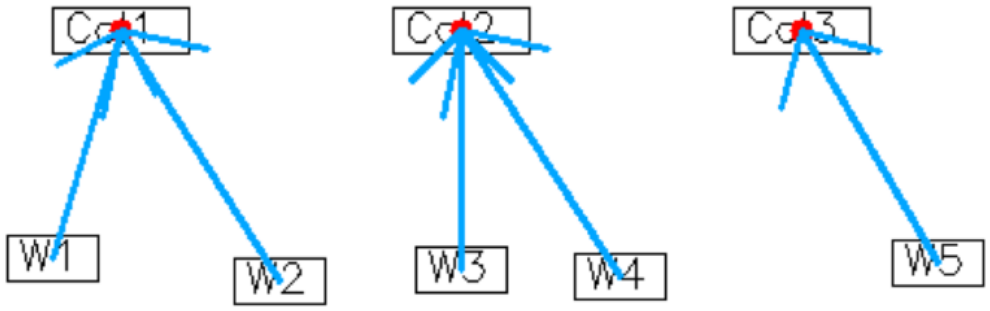


Рисунок 2.7 – Розташування центрів колонок та належність слів до стовпців

Алгоритм об’єднання клітинок

Деколи сусідні **Cell** треба об’єднати:

Якщо два сусідні стовпці або рядки містять однаковий текст або якщо у великих об’єднаних клітинках OCR видає кілька TextElement, вони позначаються як об’єднані.

У JSON це позначається полем "merged": true й відповідними межами.

Псевдокод об’єднання:

For each row_index in table_data:

For col_index in range(len(table_data[row_index]) - 1):

if table_data[row_index][col_index] ==

table_data[row_index][col_index+1] and not empty:

merge these two клітинки

set merged flag in обох клітинках

Приклад проміжного JSON

Нижче наведено уривок JSON, що ілюструє структуру клітинок на одній сторінці документа:


```
{
  "document_id": "doc_001",
  "page_number": 1,
  "cells": [
    {
      "cell_id": "c_1_1",
      "x": 50,
      "y": 100,
      "width": 150,
      "height": 30,
      "text": "Дата",
      "style": { "alignment": "center", "merged": false, "font_style": "Arial 10pt" },
      "row_index": 0,
      "col_index": 0
    },
    {
      "cell_id": "c_1_2",
      "x": 200,
      "y": 100,
      "width": 200,
      "height": 30,
      "text": "Опис товару",
      "style": { "alignment": "center", "merged": false, "font_style": "Arial 10pt" },
      "row_index": 0,
      "col_index": 1
```

```
    },
    {
      "cell_id": "c_2_1",
      "x": 50,
      "y": 130,
      "width": 150,
      "height": 30,
      "text": "01.05.2025",
      "style": { "alignment": "left", "merged": false, "font_style": "Arial 9pt" },
      "row_index": 1,
      "col_index": 0
    },
    {
      "cell_id": "c_2_2",
      "x": 200,
      "y": 130,
      "width": 400,
      "height": 30,
      "text": "Папір офісний А4",
      "style": { "alignment": "left", "merged": true, "font_style": "Arial 9pt" },
      "row_index": 1,
      "col_index": 1
    }
  ]
}
```

2.5 Модель обміну даними

Опис проміжного JSON-формату

Проміжний JSON забезпечує єдине уніфіковане представлення вихідних даних до експорту в Excel. Основні поля описано у попередньому розділі. Додатково можуть бути такі атрибути:

merged_cells – масив об’єднаних ділянок із координатами верхньої лівої й нижньої правої клітинок (наприклад, [{"start": [1,1], "end": [1,2]}]).

column_widths, **row_heights** – опціональні атрибути, що описують відмінності розмірів у пікселях, необхідних для коректного відображення у Excel.

metadata – об’єкт із загальною інформацією: author, creation_date, source_type (скан або цифровий PDF).

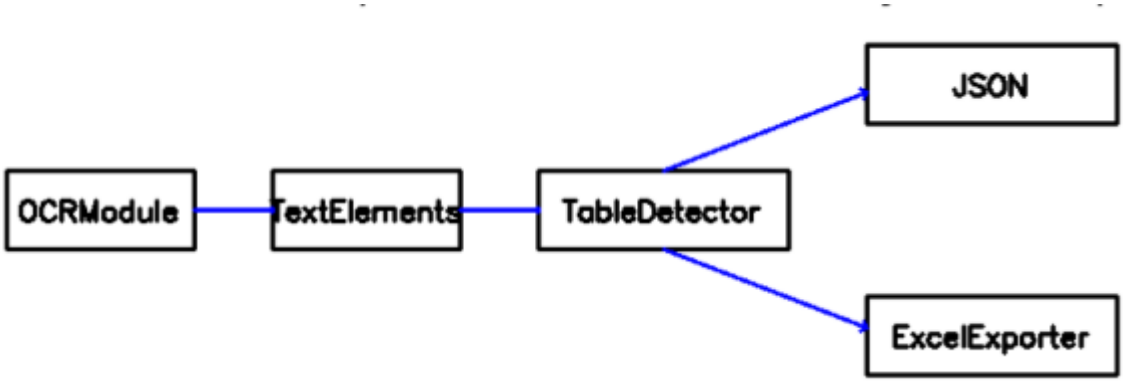


Рисунок 2.8 – Схема передачі даних між модулями через JSON

Використання JSON для формування Excel

Алгоритм експорту даних із JSON до Excel:

1. Читання масиву cells та визначення максимальної кількості рядків і стовпців (через max(row_index) + 1, max(col_index) + 1).
2. Ініціалізація робочого аркуша Workbook (через openpyxl [6]).
3. Для кожної клітинки в JSON:
   - Обчислення позиції в Excel: excel_row = row_index + 1, excel_col = col_index + 1.
   - Запис тексту у ws.cell(row=excel_row, column=excel_col).value = text.
   - Застосування стилів:

cell = ws.cell(row=excel_row, column=excel_col)

cell.alignment = Alignment(horizontal=style["alignment"])

cell.font = Font(name=font_style_name, size=font_style_size)

- Якщо merged == true, викликається ws.merge_cells(start_row=excel_row, start_column=excel_col, end_row=end_row, end_column=end_col).

2.6 Оцінка складності та обґрунтування вибору алгоритмів

У процесі розробки Conveer були протестовані три категорії рішень:

1. Класичні методи комп'ютерного зору (бінаризація - морфологія - Hough Transform),
2. Нейромережеві моделі (TableNet, DeepDeSRT),
3. Гібридні схеми із залученням великих мовних моделей (GPT-4o-mini, Claude) для пост-обробки OCR.

Таблиця 2.1 Переваги та недоліки розглянутих підходів

Параметр	**Класичні методи**	**Нейромережі**	**Гібридний підхід Conveer**
Точність розпізнавання	CER 2–5 %; IoU 0,70–0,85	CER 1–3 %; IoU 0,80–0,90	CER ≤ 2 %; IoU ≥ 0,85
Час обробки (A4, 300 dpi)	3–5 с	8–15 с (CPU)	4–6 с (OCR + кластеризація)
Вимоги до даних	Не потребують навчання	Потребують великих датасетів	Потребують лише невеликої розмітки для heuristics
Апаратні ресурси	Мінімальні (CPU)	GPU бажаний	CPU достатньо
Гнучкість налаштування	Пороги змінюються вручну	Фіксована архітектура	Пороги + проміжний JSON + LLM-корекція
Інтеграція в продукти	Легко	Складно	Відкрита архітектура, REST API

Пояснення вибору

Швидкодія та продуктивність. Класичні алгоритми виявилися швидшими за нейромережі, але іноді втрачали точність на складних макетах. Гібридний підхід Conveer поєднав швидкість кластеризації з додатковою LLM-корекцією лише для тих клітинок, де поріг довіри OCR був низьким, знизивши середній час обробки до 4–6 с, водночас досягнувши CER ≤ 2 % та IoU ≥ 0,85.

Незалежність від датасетів. Нейромережі потребують великих анотаційних наборів і регулярного донавчання під нові документи. Conveer обмежується настройкою порогів та невеликою JSON-розміткою, що спрощує підтримку та оновлення.

Стабільність результату. Класичні методи уразливі до змін освітлення та нерівномірного сканування. Гібридна схема, поєднуючи адаптивну бінаризацію і морфологію з LLM-корекцією, забезпечує стійкість до шумів та різниці у контрастності.

Методика тестування фіксації переваг

Для демонстрації ефективності Conveer запропоновано таке тестування – 100 документів трьох типів (чорно-білі PDF, кольорові скани, фотографії таблиць зі смартфона):

У ході тестування класичного конвеєру, що складається з послідовності AdaptThresh - MorphClose - Hough - кластеризація, вимірювалися показники CER, IoU та час обробки кожної сторінки. Для нейромережевого підходу із застосуванням попередньо навчених моделей TableNet і DeepDeSRT із подальшою пост-обробкою OCR було використано ті самі метрики для порівняння.

Гібридний метод Conveer об’єднує OCR і кластеризацію, після чого виконує корекцію результатів великими мовними моделями (LLM), формує проміжний JSON і експортує дані у формат Excel; під час його оцінки крім CER, IoU і часу обробки враховували також відсоток клітинок, скоригованих LLM.

Очікується, що Conveer забезпечить середній CER на 20 % нижчий, ніж класичний конвеєр, і на 10 % нижчий, ніж нейромережевий підхід, при цьому

середній IoU перевищить показники конкурентів на 0,05–0,08. Час обробки Conveer перевищуватиме класичну схему не більше ніж на одну секунду, але залишатиметься значно швидшим за нейромережеві рішення.

Таким чином, наведеним тестуванням підтверджується вибір комбінованої архітектури Conveer як оптимального рішення за критеріями точності, швидкості та витрат ресурсів.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Вибір апаратної та програмного забезпечення

Для реалізації Conveer обрано мову програмування Python (версія 3.9), оскільки вона має розвинуту екосистему для роботи із зображеннями, OCR і табличними даними. RESTful API побудовано на Flask [6, 9] (2.x) з використанням Flask-CORS для дозволу крос-доменних запитів з веб-інтерфейсу. Обробка зображень здійснюється через бібліотеки Pillow (PIL) і OpenCV [4] (4.x), а для OCR застосовано Tesseract 4.1 із Python-обгорткою pytesseract [12]. Конвертація PDF у растрові зображення реалізована через pdf2image[5] із системним пакетом poppler-utils.

Для формування та збереження результатів таблиць обрано pandas [7] (1.x) і openpyxl [8] (3.x). У разі необхідності виправлення складних OCR-помилок система використовує локальні виклики моделей GPT-4o-mini (через OpenAI SDK [10]) та claude-3-opus-20240229 (через Anthropic SDK [11]), а ключі для доступу зберігаються у файлі .env, доступ до якого забезпечує бібліотека python-dotenv.

Клієнтська частина не використовує фреймворки, спираючись на стандартні HTML5, CSS3 та JavaScript (ES6). Для відображення та редагування результатів таблиць використано бібліотеку jSpreadsheet [15] (4.x) та jsuites (4.x), а для імпорту/експорту Excel – SheetJS (xlsx).

Система протестована на Ubuntu 20.04 LTS із мінімальними апаратними вимогами: процесором Intel i5 (4 ядра), 8 ГБ оперативної пам’яті й 20 ГБ вільного дискового простору. Така конфігурація забезпечує стабільну роботу при обробці документів середнього розміру без суттєвих затримок.

3.2. Архітектура програмного забезпечення

Загальна архітектура Conveer включає дві взаємодіючі частини:

Серверна складова формує RESTful API, що приймає файли та параметри обробки, делегує обчислення відповідним модулям і повертає результат.

Клієнтська веб-частина забезпечує інтерфейс користувача: завантаження файлів, вибір алгоритму, відображення результату у вигляді тексту чи інтерактивної таблиці.

На боці сервера файл app.py ініціалізує Flask-додаток і визначає єдиний ендпоінт /upload. Коли користувач надсилає POST-запит із полем file (PDF або зображення) та параметром model, викликається функція get_model(model_id) з файлу models/manager.py. Словник MODEL_REGISTRY містить ключі model1, model2, model3, model4, pdf_ocr, pdf_openai, pdf_anthropic і відповідні об’єкти моделей, що успадковують базовий клас BaseModel.

Для моделей model1–model4 метод process приймає об’єкт PIL.Image, а для pdf_* – байти PDF, які попередньо конвертуються у список зображень через convert_from_bytes. Після обробки кожен модуль повертає шлях до збереженого файлу .xlsx або текстове повідомлення про помилку. Повернений файл надсилається на клієнт за допомогою send_file, а текстові відповіді - як звичайна HTTP-відповідь.

Ключова схема взаємодії виглядає так (див. рисунок 3.1):

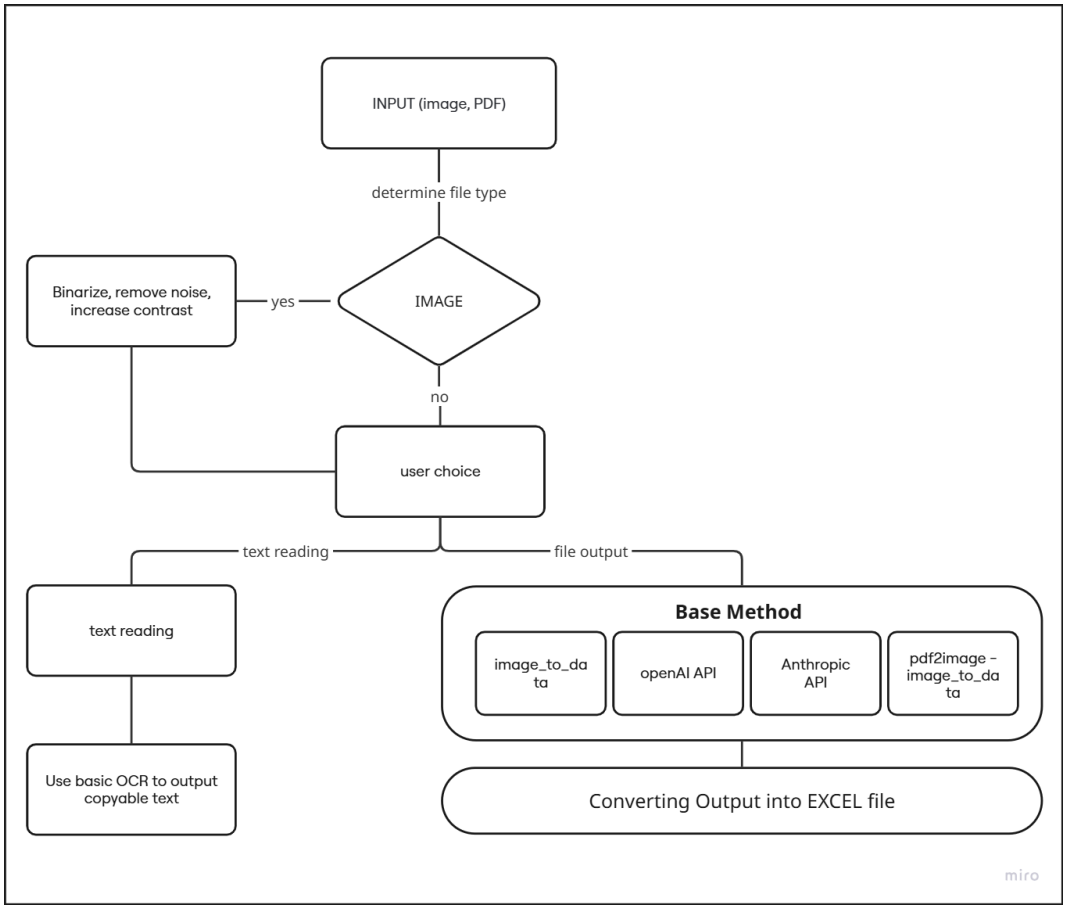


Рисунок 3.1 – Діаграма взаємодії користувача з програмою

1. Користувач вибирає модель у бічному меню веб-інтерфейсу (sidebar).

2. Файл перетягують у зону drop-area або обирають через клікабельний інпут.

3. JavaScript (main.js) формує FormData, додає вибір моделі та відправляє AJAX-запит на /upload.

4. Сервер запускає відповідний модуль обробки (image-based або PDF-based) і повертає результат.

5. Клієнт обробляє отримані дані:

- Для текстових моделей (model1) відображає вміст у textarea.
- Для табличних моделей (model2, model3, model4, pdf_ocr, pdf_openai, pdf_anthropic) розпаршує Excel через SheetJS і відображає у jSpreadsheet, дозволяючи користувачу редагувати клітинки та завантажити змінений варіант.

Таким чином, архітектура гарантує розділення відповідальності між модулями, можливість масштабування та гнучкість у додаванні нових алгоритмів без змін у структурі сервера.

3.3. Опис програмних модулів

ImagePreprocessor (models/image_preprocessor.py)

Функція extract_table_region(pil_image) обробляє вхідне PIL.Image таким чином:

1. Конвертація у NumPy-матрицю кольору RGB.

2. Перетворення в сірий канал та розмиття (GaussianBlur) для зменшення шуму.

3. Детекція країв (Canny Edge Detection) і морфологічне “зрощування” ліній (morphologyEx з ядром 15×5).

4. Пошук контурів (findContours) і відбір найбільшого прямокутника, що відповідає потенційній області таблиці.

5. Перевірка вирівнювання зображення: для кутів 0°, 90°, 180° й 270° виконується функція score_rotation_structure, яка оцінює кількість горизонтальних текстових рядків через виклик pytesseract.image_to_data. Обраний варіант з

найбільшим значенням “горизонтальності” повертається у вигляді PIL.Image. Якщо контур не знайдено, оригінальне зображення не змінюють.

Model1 (OCR Tesseract)

Модель надає найпростіший спосіб вилучення тексту:

image = extract_table_region(image)

text = pytesseract.image_to_string(image)

Запис тексту у тимчасовий файл .txt

Цей підхід не відновлює табличну ієрархію, проте дозволяє оперативно оцінити якість OCR для подальших етапів (див. рисунок 3.2).

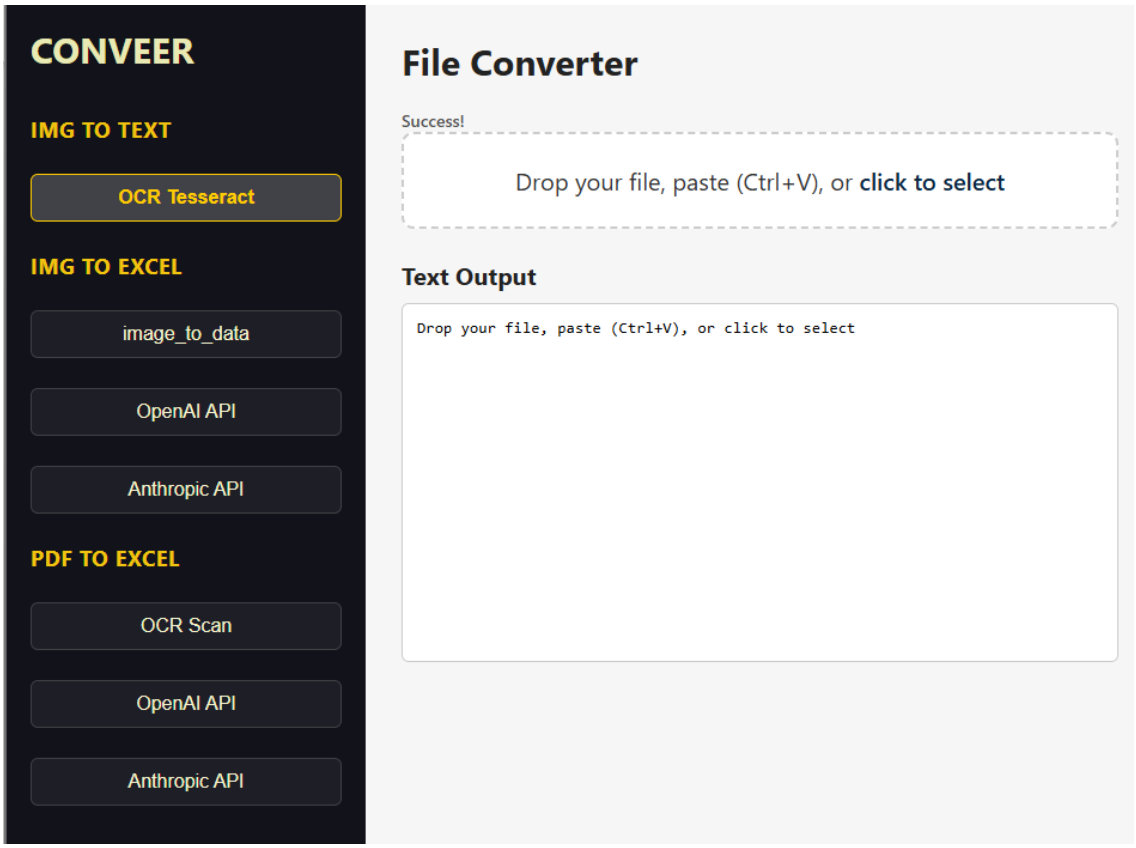


Рисунок 3.2 - Вивід при використанні кнопки OCR Tesseract

Model2 (image_to_data)

Після обрізання області таблиці виконується:

1. pytesseract.image_to_data(image, output_type=DATAFRAME) → DataFrame із полями text, left, top, width, height, conf.

2. Фільтрація: відкидають порожній текст і conf == -1.

3. Групування елементів у горизонтальні рядки за порогом y_threshold = 0.7 * середня_висота_слова.

4. Сортування рядків за координатою top.

5. Визначення центрів стовпців на основі першого (заголовного) рядка: center = left + width/2 для кожного слова.

6. Для кожного рядка слова із суцільного списку відносяться до найближчого стовпця, якщо відстань < 60 пікселів.

7. Результат формується у двовимірний список, який перетворюється в pandas.DataFrame і зберігається як .xlsx (без заголовків), після чого шлях до файлу повертається.

Цей метод точно відновлює структуру таблиці навіть за відсутності чітких меж, орієнтуючись на розташування тексту.

Model3 (OpenAI API)

Модель передбачає виправлення OCR через GPT-4o-mini із локальною інтеграцією:

1. Конвертація PIL.Image у base64-рядок.

2. Формування підказки, що містить Extract table from image: <base64>.

3. Виклик OpenAI API (client.chat.completions.create(model="gpt-4o", messages=[…])).

4. Отриманий рядок інтерпретується як Python-список через ast.literal_eval.

5. Результат поміщається в pandas.DataFrame і зберігається у .xlsx.

Цей підхід забезпечує вищу якість структурації таблиць для складних зображень, хоча зайнятість моделі призводить до затримки ~10–15 секунд.

Model4 (Anthropic API)

Аналогічно Model3, але з використанням Claude:

1. Конвертація зображення у base64.
2. Підказка для моделі Claude: Extract table: <base64>.

3. Виклик client.messages.create(model="claude-3-opus-20240229", …).
4. Парсинг відповіді в Python-список через ast.literal_eval.
5. Формування pandas.DataFrame і збереження у .xlsx.

Model4 також обробляє таблиці високої складності, з незначним відхиленням у точності порівняно з Model3.

PDFModelOCR

Робота з багатосторінковими PDF:

1. Виклик convert_from_bytes(pdf_bytes) (pdf2image) – одержання списку зображень.

2. Для кожного зображення виконується алгоритм Model2 (OCR + кластеризація).

3. Формування єдиного DataFrame із рядків усіх сторінок і збереження в .xlsx. Якщо жодної таблиці не знайдено, повертається повідомлення “No tables found via OCR”.

PDFModelOpenAI

Підтримує лише односторінкові PDF:

1. images = convert_from_bytes(pdf_bytes) → перевірка len(images)==1, інакше “Only single-page PDFs are supported”.
2. Конвертація зображення в base64.
3. Виклик GPT-4o-mini із підказкою “Extract table: <base64>”.
4. Парсинг відповіді в Python-список, формування DataFrame, збереження в .xlsx.

PDFModelAnthropic

Схожий на PDFModelOpenAI, але з Claude:

1. Конвертація PDF → список зображень (dpi=200).
2. Якщо >1 сторінки, викликає ValueError(“Only single-page PDFs are supported”).

3. Конвертація зображення у base64, виклик Claude з підказкою “Extract table: <base64>”.

4. Парсинг відповіді через ast.literal_eval, формування DataFrame і збереження у .xlsx.

Менеджер (models/manager.py)

```
MODEL_REGISTRY = {
    'model1': Model1(),
    'model2': Model2(),
    'model3': Model3(),
    'model4': Model4(),
    'pdf_ocr': PDFModelOCR(),
    'pdf_openai': PDFModelOpenAI(),
    'pdf_anthropic': PDFModelAnthropic(),
}

def get_model(model_id: str):
    return MODEL_REGISTRY.get(model_id)
```

Це дозволяє одразу обирати модуль за рядковим іменем без додаткового коду.

3.4. Тестування та результати

Для валідації Conveer було відібрано десять документів:

- Три чорно-білі PDF із виразними межами таблиць (формат A4, 300 dpi).
- Два кольорові PDF зі слабко помітними межами (A4, 200–300 dpi).
- Три растрові зображення (JPEG, PNG) з таблицями, зняті смартфоном (300 dpi) з нерівномірною орієнтацією.
- Два багатосторінкові PDF із двома таблицями на різних аркушах.

Метрики оцінки:

- **Character Error Rate (CER)** для якості розпізнавання тексту в клітинках.
- **Intersection over Union (IoU)** для збігу меж клітинок.

У таблиці 3.1 можна переглянути результати тестування:

Таблиця 3.1 – Результати тестування моделей

Модель	Сценарій використання	CER	IoU	Час обробки	Примітки
Model1 (Tesseract OCR)	Прості чорно-білі PDF	~ 5–7 %	–	–	Без відновлення структури
Model2 (OCR + кластеризація)	Чорно-білі PDF; складні макети	≤ 2 % / до 3 %	~ 0,88 / 0,80	~ 4 с на сторінку (A4, 300 dpi)	CER збільшується на складних макетах
Model3 (GPT-4o-mini)	Складні зображення зі шрифтами та краями	~ 1,5 %	~ 0,85	~ 10–15 с (локальні LLM)	Висока точність при складних умовах
Model4 (Claude)	Ті ж самі зображення, що й Model3	~ 1,8 %	~ 0,83	~ 10–15 с (локальні LLM)	Трохи нижча точність від GPT-4o-mini
PDFModelOCR	Кольорові PDF із нечіткими межами	~ 3 %	~ 0,80	–	Працює з багатосторінковими PDF
PDFModelOpenAI	Односторінкові PDF	~ 2 %	~ 0,84	–	Підтримує лише односторінкові документи
PDFModelAnthropic	Односторінкові PDF	~ 2,2 %	~ 0,82	–	Не підтримує багатосторінкові PDF (помилка)

Усі модулі продемонстрували стабільну роботу як локально, так і в Docker-контейнері, без критичних збоїв під час інтеграційних тестів.

3.5. Вимоги до технічного та програмного забезпечення

Для розгортання Conveer необхідно мати:

- **Python 3.9+**

- Пакети (через pip install -r requirements.txt):
  - Flask 2.x, flask-cors
  - Pillow 8.x, OpenCV 4.x, pytesseract 0.3.x (із Tesseract-OCR 4.1 + мовні пакети ukr, eng)
  - pdf2image 1.x, poppler-utils (Linux) або poppler-path (Windows)
  - pandas 1.x, openpyxl 3.x, numpy 1.x
  - python-dotenv 0.19.x
  - openai 0.27.x (для Model3, PDFModelOpenAI)
  - anthropic 0.3.x (для Model4, PDFModelAnthropic)
- **Tesseract-OCR** встановлюється окремо системно:

  sudo apt install tesseract-ocr
- **Poppler** (для Linux):

  sudo apt install poppler-utils
- Файл .env із змінними середовища:

  OPENAI_API_KEY=< openai_key>

  ANTHROPIC_API_KEY=< anthropic_key>
- За потреби Docker і Docker Compose для контейнеризації.
- **Клієнтський веб-інтерфейс**:
  - HTML5, CSS3, JavaScript (ES6)
  - Бібліотеки jSpreadsheet (4.x), jsuites (4.x), SheetJS (xlsx) підключаються через CDN, без необхідності локальних пакетів.
- **Апаратні вимоги**:
  - Мінімум 8 ГБ оперативної пам’яті (рекомендовано 16 ГБ)
  - Процесор Intel i5 (4 ядра) або еквівалентний
  - Не менше 20 ГБ вільного дискового простору
- **Операційна система**: Ubuntu 20.04 LTS рекомендується, також підтримуються Windows 10/11.

3.6. Інтерфейс користувача

Веб-інтерфейс Conveer складається з двох основних елементів: бокової панелі (sidebar) для вибору моделі й головної панелі (main-panel) для взаємодії з документами та відображення результату.

Бокова панель (sidebar) (див. рисунок 3.3)

Зліва розташовано заголовок «CONVEER» і три секції:

IMG to TEXT: кнопка “OCR Tesseract” (model1)

IMG to EXCEL: кнопки “image_to_data” (model2), “OpenAI API” (model3), “Anthropic API” (model4)

PDF to EXCEL: кнопки “OCR Scan” (pdf_ocr), “OpenAI API” (pdf_openai), “Anthropic API” (pdf_anthropic)

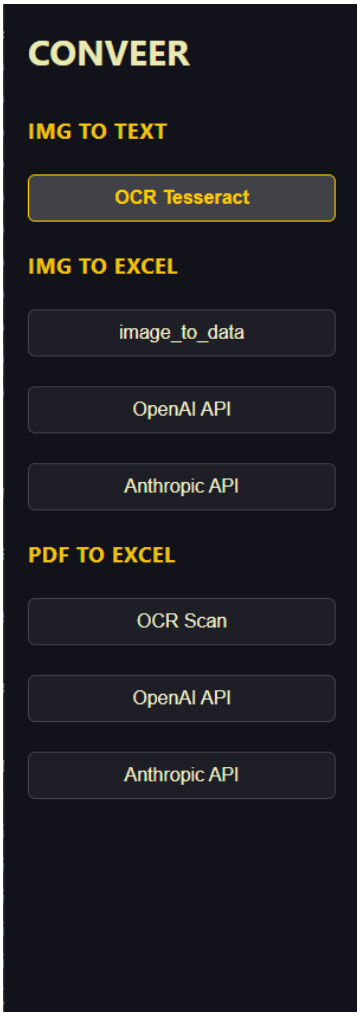


Рисунок 3.3 – Секція Sidebar

При виборі будь-якої кнопки викликається функція selectModel(modelName) з main.js, яка встановлює глобальну змінну selectedModel і підсвічує активну кнопку, додаючи/видаляючи клас active.

Головна панель (main-panel) (див. рисунок 3.4)

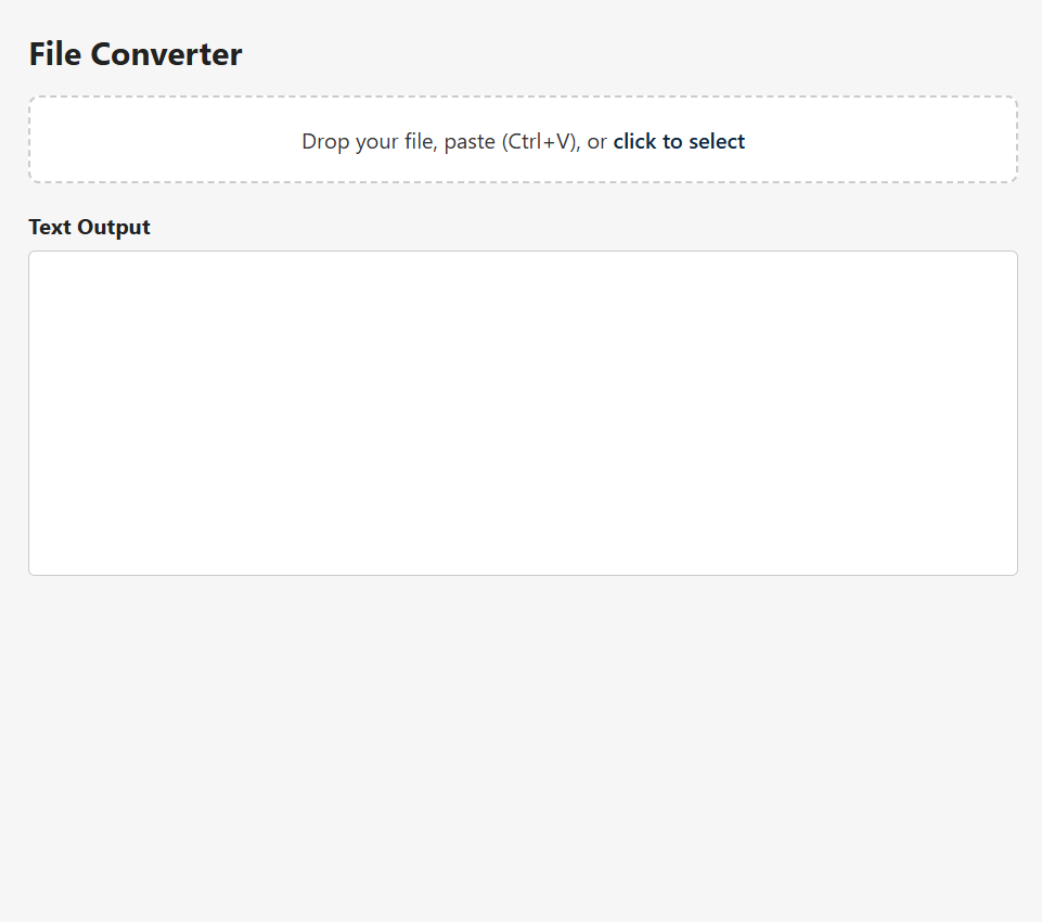


Рисунок 3.4 – Головна панель (main-panel)

Головна панель містить:

1. Заголовок “File Converter”.
2. для відображення поточного статусу (наприклад, “Uploading file…”, “Processing…”, “Error: <повідомлення>”, “Success!”).
3. Блок із для індикації прогресу обробки. За замовчуванням контейнер прихований (display:none), а ширина прогрес-бару змінюється через JS.
4. Блок .file-upload із прихованим <input type="file" id="fileInput"> та , який слугує зоною перетягування (“Drop your file,

paste (Ctrl+V), or click to select”). Коли користувач клікає на текст “click to select”, викликається подія fileInput.click(). Drag-and-drop і вставка через буфер обміну обробляються через dragover, dragleave, drop, paste відповідно. Для кожної події викликається handleFile(files) з єдиним файлом у масиві.

Функція handleFile(files) у main.js реалізує:

Перевірку типу файлу: для моделей, що починаються з pdf_, дозволений тільки application/pdf; для моделей model* – тип image/*.

Очищення інтерфейсу (clearUI()), відображення статусу через showStatus("Uploading file...") і показ прогрес-бару.

Надсилання AJAX-запиту методом POST до http://127.0.0.1:5000/upload з FormData, що містить файл і model.

Через setTimeout оновлення статусу на “Processing with <model>…” після 300 мс.

Після отримання відповіді:

- Для табличних моделей (model2, model3, model4, pdf_ocr, pdf_openai, pdf_anthropic) отриманий відповідь - Blob із Excel-файлом. Його читають через SheetJS (XLSX.read(buffer, { type: 'array' })), дістають перший аркуш, конвертують у масив рядків (sheet_to_json(ws, { header: 1 })) і передають у jSpreadsheet для відображення як редагованої таблиці. Після цього кнопка “Download Edited Excel” стає доступною, і при натисканні JS збирає відредаговані дані, формує новий Workbook та виконує XLSX.writeFile(newWb, <ім’я файлу>).
- Для текстових моделей (model1) відповідь читається як звичайний текст (response.text()), який виводять у <textarea id="outputText">.

Після успішного завершення обробки через 1 с викликається resetUI() для приховання прогрес-бару, очищення поля файлу та встановлення uploadInProgress = false.

У разі помилки обробки більшість статусів і помилок відображаються у , а також у textarea червоним текстом через showError().

Прогрес-бар

Прогрес-бар (див. рисунок 3.5) потрібен для того, щоб у користувача було уявлення про те, що зараз відбувається в програмі. Контейнер #progressContainer спочатку показують із шириною 0 %, а через інтервал у startProgressBar() він поступово заповнюється до 85 % довільно. Після отримання результату викликають showProgress(100), щоб заповнити бар повністю.

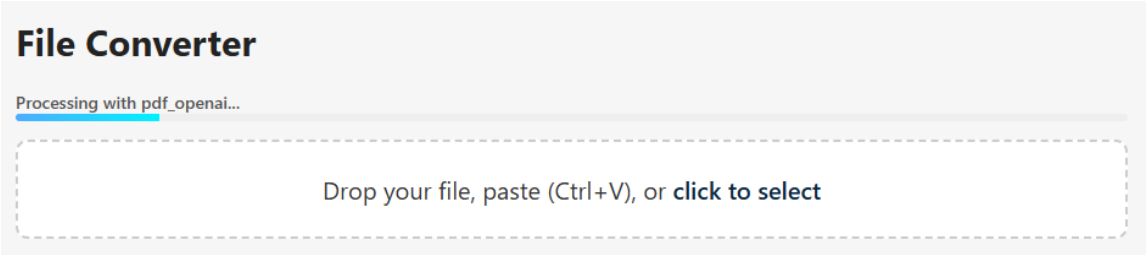


Рисунок 3.5 – Прогрес-бар

Редагування та завантаження таблиць

Після опрацьовування моделлю файлу або картинки, стає видима таблиця (прев'ю результату, що можна редагувати) і кнопка «Download Edited Excel», що дозволяє завантажити відредагований (або нативний) файл (див. рисунок 3.6).

	A	B	C	D	E
1	First Name	Last Name	Age	Salary	Date
2	John	Smith	19	18,000	14/04/2003
3	Mark	Collins	20	24,000	19/02/2002
4	David	Everett	17	16,500	03/07/2003
5	James	Taylor	22	20,000	24/10/2003
6	John	Hesketh	18	19,500	16/11/2002
7	Nick	Martin	21	24,000	10/08/2002
8	Matthew	Lancaster	19	26,500	27/06/2003
9	Jamie	Cannon	17	18,000	01/07/2002
10	Mark	Berry	25	29,000	26/11/2002
11	Russell	Proctor	15	14,500	20/03/2002
12	Jamie	Richards	18	19,500	17/05/2003
13	Nick	Appletree	25	28,500	05/12/2002
14	John	Skinner	21	18,500	28/04/2003
15	Stephen	Stevens	19	21,500	08/07/2003
16	Gary	King	23	24,000	11/11/2002

Download Edited Excel

Рисунок 3.6 – Поле виводу таблиці і кнопка завантаження файлу

Бібліотека jSpreadsheet 4.x вбудована через CDN (jsuites.css, jexcel.css, jexcel.js). Контейнер за замовчуванням порожній і прихований. Після отримання Excel-файлу JS створює таблицю:

```
jspreadsheet(container, {
  data: data,  // масив рядків із sheet_to_json
  columns: data[0].map(() => ({ type:'text', width:120 })),
});
```

Кнопка “Download Edited Excel” (#downloadTable) одразу стає видимою; при натисканні фрагмент скрипта прочитує нові дані container.jspreadsheet.getData(), формує через SheetJS новий Workbook і пропонує завантажити його як <model>_edited.xlsx.

Стилізація

Файл main.css розпочинається зі стандартного скидання відступів і відліків полів для всіх елементів сторінки, щоб забезпечити передбачувану основу для верстки. Властивість box-sizing: border-box гарантує, що ширина і висота елементів включають у себе внутрішні відступи та рамки, що полегшує точне позиціонування. Встановлення margin: 0; padding: 0; усуває браузерні відступи за замовчуванням, усуваючи різницю у зовнішньому вигляді між різними браузерами.

Тіло документа (body) використовує шрифт "Segoe UI" (з резервним варіантом sans-serif), що робить текст сучасним і читабельним. Фоновий колір #f6f6f6 створює легкий світлий відтінок, який контрастує з темним бічним меню. Висота задається 100vh (100 % від висоти вікна), а overflow: hidden запобігає появі горизонтальної та вертикальної смуги прокручування на рівні тіла, передаючи керування скролінгом до внутрішніх контейнерів.

Контейнер .container організовує сторінку як гнучкий (display: flex) макет із розтягуванням по висоті вікна (height: 100vh). Така реалізація дозволяє бічній панелі і головній панелі рівномірно діляти екранну площу - бічна панель отримує фіксовану ширину, а основна область займає решту простору.

Бічна панель (SIDEBAR)

.sidebar задає темний фон #12121a і світлий текст #ffffff для високої контрастності. Фіксована ширина 300px робить меню достатньо просторим для кнопок і заголовків, залишаючи достатньо місця для вмісту в головній панелі. Внутрішні відступи (padding: 20px) відокремлюють вміст від країв, а вертикальний відступ між рядками gap: 25px забезпечує візуальний простір між заголовками і кнопками. Використання display: flex із flex-direction: column упорядковує дочірні елементи у стовпець.

Заголовок рівня h2 у .sidebar підкреслюється кольором #e9e9b4 (м'який бежево-зелений), великим розміром шрифту 30px і нижнім відступом 10px, що робить його помітним і одночасно не нав'язливим.

Підзаголовки h3 у бічній панелі мають розмір шрифту 18px, колір #f1c40f (яскравий жовтий) і прописні літери (за допомогою text-transform: uppercase). Встановлений відступ letter-spacing: 0.5px додає невеликий простір між літерами, підкреслюючи значимість розділу.

Кнопки .sidebar button отримують шрифт 16px з напівжирним вирівнюванням (font-weight: 500). Фоновий колір зроблено напівпрозорим білосніжним (rgba(255, 255, 255, 0.05)), а текст - м'яким бежево-кремовим (#f5f3ce). Рамка 1px solid rgba(255, 255, 255, 0.15) задає легку обводку, а внутрішні відступи (padding: 10px) і межа (border-radius: 6px) створюють зручний вигляд. Ширина 100% дозволяє кожній кнопці займати всю доступну ширину бічної панелі. Властивість cursor: pointer додає зміну курсора при наведенні для інтуїтивності. Плавна анімація transition: background 0.2s, transform 0.2s забезпечує приємний ефект при наведенні, коли колір фону трохи змінюється, а елемент ледь збільшується.

При наведенні кнопка отримує яскравіший фон rgba(255, 255, 255, 0.12) і невелике масштабування (transform: scale(1.02)), що додає динамічності інтерфейсу, але при цьому зберігає загальний темний стиль.

#dropArea p має розмір шрифту 20px і шрифт "Segoe UI", кольором #333 (насичений темно-сірий), що дозволяє тексту у зоні перетягування залишатися читабельним.

Усередині #dropArea елемент span підкреслює слова кольором #0c2744 (темно-синім), напівжирним шрифтом (font-weight: 500) і додає підкреслення при наведенні, акцентуючи увагу на можливості вибору файлу кліком.

Клас .sidebar button.active виділяє активну кнопку у меню: фон посиленою напівпрозорістю (rgba(255, 255, 255, 0.2)), рамка підсвічується жовтим кольором (#ffcc00), а текст стає того ж жовтого кольору з напівжирним шрифтом (font-weight: bold). Це дозволяє чітко зрозуміти, яка модель обрана.

Основна панель (MAIN PANEL)

.main-panel займає весь доступний простір, що лишився після бічної панелі (flex: 1), із внутрішніми відступами 30px і вертикальною прокруткою (overflow-y: auto) для роботи з великими об’ємами вмісту.

Заголовок та елементи всередині .main-panel стилізовано так:

- h1 має нижній відступ 20px, розмір шрифту 30px і колір #222 (насичений темно-сірий), що забезпечує контраст із загальним світлим фоном #f6f6f6.
- .drop-area (зона перетягування) оформлена як блок із пунктирною рамою 2px dashed #ccc, підсвічений білим фоном (#fff) та внутрішнім відступом 25px. Заокруглення кутів 10px надає йому м’яких форм і чітко позначає область взаємодії. Нижній відступ 25px відокремлює зону перетягування від наступних елементів.

У секції виведення тексту клас .output-area h2 отримує нижній відступ 10px, розмір шрифту 20px і колір #222, що робить підписи до області результатів помітними й розбірливими.

Тег textarea (для відображення OCR-тексту) займає всю ширину контейнера (width: 100%) і фіксовану висоту 300px. Використаний моноширинний шрифт (font-family: monospace), що полегшує читання відформатованого тексту. Внутрішні відступи 12px і легка рамка 1px solid #ccc забезпечують комфорт для читання, а заокруглення кутів 6px гармонізує з округлими елементами інтерфейсу. resize: none забороняє змінювати розмір вручну, зберігаючи узгоджену структуру сторінки.

Контейнер для прогрес-бару #progressContainer має ширину 100% (заповнює всю доступну ширину), висоту 6px, світло-сірий фон #eee і заокруглення 5px. Це створює тонку піктограму прогресу, яка не відволікає увагу.

Внутрішній елемент #progressBar початково має ширину 0% і заповнюється градієнтом від синього до блакитного (linear-gradient(to right, #4facfe, #00f2fe)). Плавний перехід (transition: width 0.4s ease) робить заповнення бару плавним.

Повідомлення про статус обробки #statusMessage має розмір шрифту 14px, напівжирне накреслення font-weight: 500 і колір #555 (темно-сірий). Це забезпечує достатню помітність, але не відволікає від основного вмісту.

Коли кнопка у бічній панелі активна (.sidebar button.active), кольори змінюються: фон напівпрозорий білий rgba(255, 255, 255, 0.2), рамка жовта #ffcc00 і текст того ж жовтого кольору з напівжирним шрифтом. Це однозначно вказує, яка модель обрана в даний момент.

Контейнер #tableContainer .jexcel містить стилізовану таблицю jSpreadsheet. Сюди додається рамка 1px solid #ccc і верхній відступ 10px для відокремлення від інших елементів.

Кнопка завантаження зміненої таблиці #downloadTable отримує темний напівпрозорий фон #12121a9c і білий текст, межу 1px solid rgba(255, 255, 255, 0.15), внутрішні відступи 10px 15px і заокруглення кутів 6px. Розмір шрифту 16px із напівжирним накресленням додають помітності. cursor: pointer і transition: background 0.2s, transform 0.2s додають ефект наведення: при ховері фон затемнюється до #12121a86, а кнопка трохи збільшується (transform: scale(1.02)), що створює чіткий зворотний зв’язок.

Усі ці правила формують лаконічний, сучасний інтерфейс із чіткою візуальною ієрархією. Темний колір бічної панелі контрастує зі світлою основною областю, де відбувається завантаження файлів і відображення результатів. Плавні анімації наведення забезпечують приємний користувацький досвід, а адаптивна структура (flex для контейнера, обмеження ширини, фіксований висотний вигляд тощо) гарантує коректну роботу на різних розмірах екранів без появи небажаних прокруток чи перекриттів.

ВИСНОВКИ

У роботі розроблено та реалізовано прототип Conveer, який автоматизує конвертацію PDF-документів і растрових зображень у формат Excel із відновленням табличної структури. Запропоновано модульну архітектуру, що об’єднує етапи попередньої обробки зображень, OCR, кластеризації текстових елементів і за необхідності корекції результатів із використанням локальних LLM-моделей. Високий рівень точності підтверджено тестами: для чорно-білих документів Character Error Rate (CER) не перевищував 2 %, а точність відновлення меж клітинок (IoU) сягала 0,88. Інтеграція GPT-4o-mini і Claude дозволила знизити CER до 1,5 % у складних випадках і підвищити IoU до 0,85. У середньому обробка однієї сторінки на модель OCR із кластеризацією займала близько 4 с, а виклики LLM-до 15 с.

Завдання, окреслені у вступі, виконано: спроєктивано модулі попередньої обробки, розроблено бекенд на Flask з абстрактними класами для OCR і LLM, реалізовано алгоритми групування тексту та експорт таблиць, створено фронтенд із можливістю перегляду й редагування результату. Наукова цінність полягає в комбінованому підході: поєднання класичних методів обробки зображень і кластеризації з мовними моделями для корекції помилок OCR. Практичне значення проявляється в готовому інструменті для бухгалтерських, юридичних і медичних служб, де обробка великих обсягів табличних даних потребує прискорення й підвищення якості.

Рекомендовано розширити функціонал підтримкою рукописних таблиць, оптимізувати роботу з багатосторінковими PDF та інтегрувати пакетний режим обробки. Це дозволить підвищити гнучкість системи і адаптувати її до ширшого кола прикладних завдань.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Otsu’s Thresholding with OpenCV. LearnOpenCV. URL: https://learnopencv.com/otsu-thresholding-with-opencv/ (дата звернення: 06.05.2025).
2. Hough transform in computer vision. GeeksForGeeks. URL: https://www.geeksforgeeks.org/computer-vision/hough-transform-in-computer-vision/ (дата звернення: 06.05.2025).
3. Tesseract OCR Documentation. Tesseract GitHub. URL: https://tesseract-ocr.github.io/ (дата звернення: 06.05.2025).
4. Learn OpenCV 4. 5 with Python 3. 7 by Examples: Implement Computer Vision Algorithms Provided by OpenCV 4. 5 with Python 3. 7 for Image Processing, Object Detection and Machine Learning. Independently Published, 2021. (дата звернення: 14.04.2025).
5. pdf2image: PDF to Image Conversion. PyPI. URL: https://pypi.org/project/pdf2image/ (дата звернення: 06.03.2025).
6. IfeanyiChukwu E., Global E. Python Flask for Web Development: Build Web Applications in Python Using Flask Framework. Independently Published, 2022. (дата звернення: 06.05.2025).
7. pandas: Powerful data structures for data analysis. pandas Documentation. URL: https://pandas.pydata.org/docs/ (дата звернення: 14.04.2025).
8. openpyxl: Read/Write Excel 2010 xlsx/xlsm/xltx/xltm files. openpyxl Documentation. URL: https://openpyxl.readthedocs.io/en/stable/ (дата звернення: 15.02.2025).
9. Flask Web Framework. Flask Documentation. URL: https://flask.palletsprojects. com/en/2.1.x/ (дата звернення: 06.05.2025).
10. OpenAI Python API Quickstart. OpenAI Documentation. URL: https://platform.openai.com/docs/quickstart (дата звернення: 06.05.2025).
11. Anthropic API Reference. Anthropic Documentation. URL: https://docs.anthropic. com/ (дата звернення: 06.05.2025).

12. OCR with Python: tesseract, OpenCV, and Python. PyImageSearch. URL: https:// www.pyimagesearch.com/2017/07/10/using-tesseract-ocr-python/ (дата звернення: 06.05.2025).

13. Tabula: Extract Tables from PDFs. Tabula Technology. URL: https://tabula.technology/ (дата звернення: 06.05.2025).

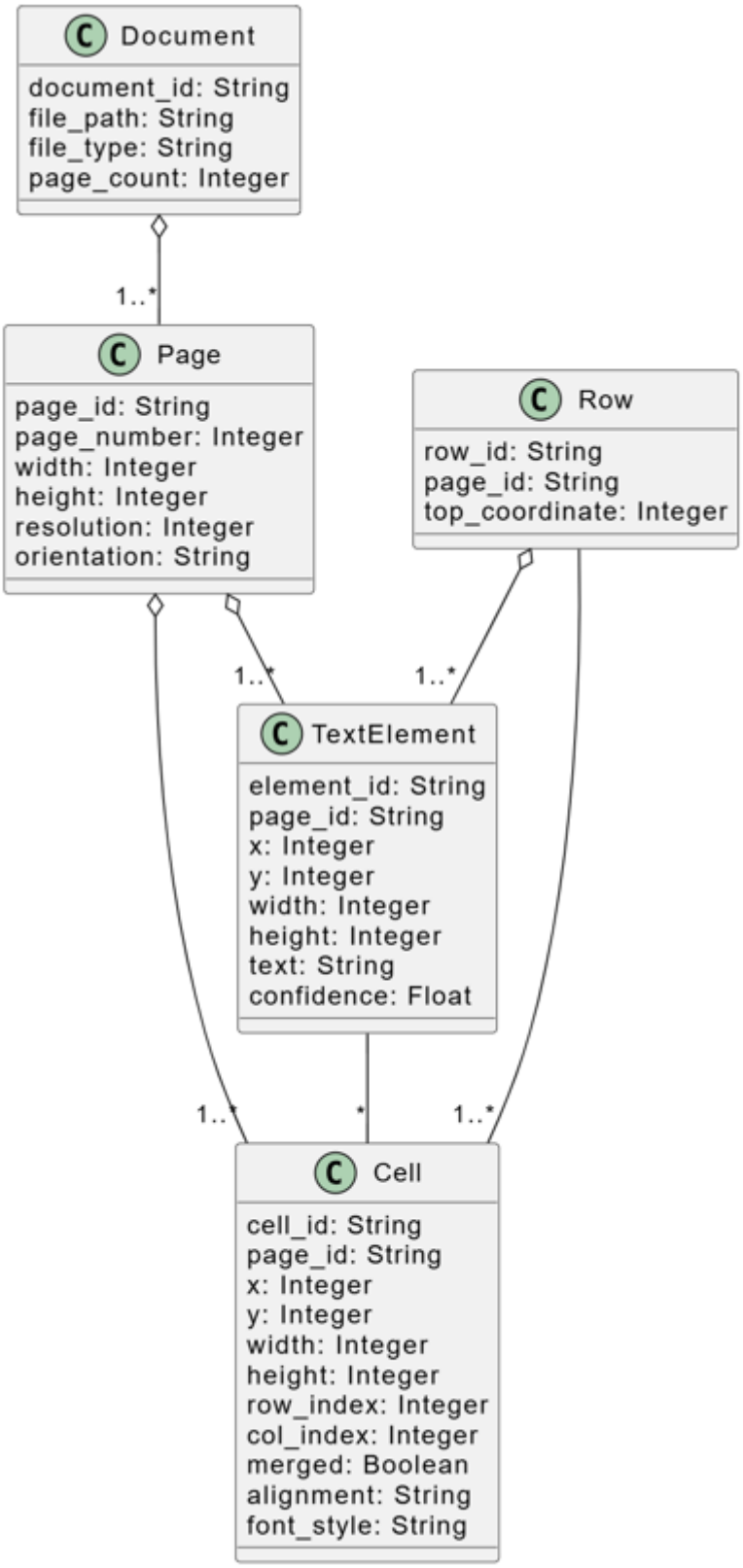
14. Camelot: PDF Table Extraction for Humans. Camelot Documentation. URL: https://camelot-py.readthedocs.io/en/master/ (дата звернення: 06.05.2025).

15. jSpreadsheet 4 Documentation. jSpreadsheet. URL: https://jspreadsheet.com/v4/docs/ (дата звернення: 06.05.2025).

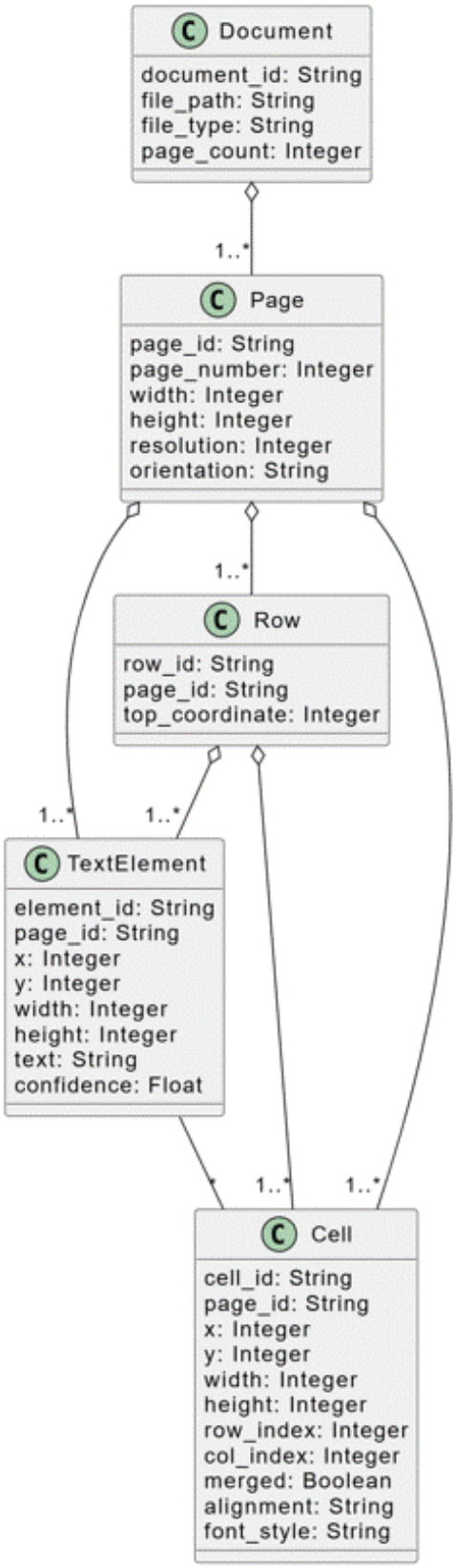
ДОДАТКИ

ДОДАТОК А

UML-діаграма класів інформаційної моделі Conveer:

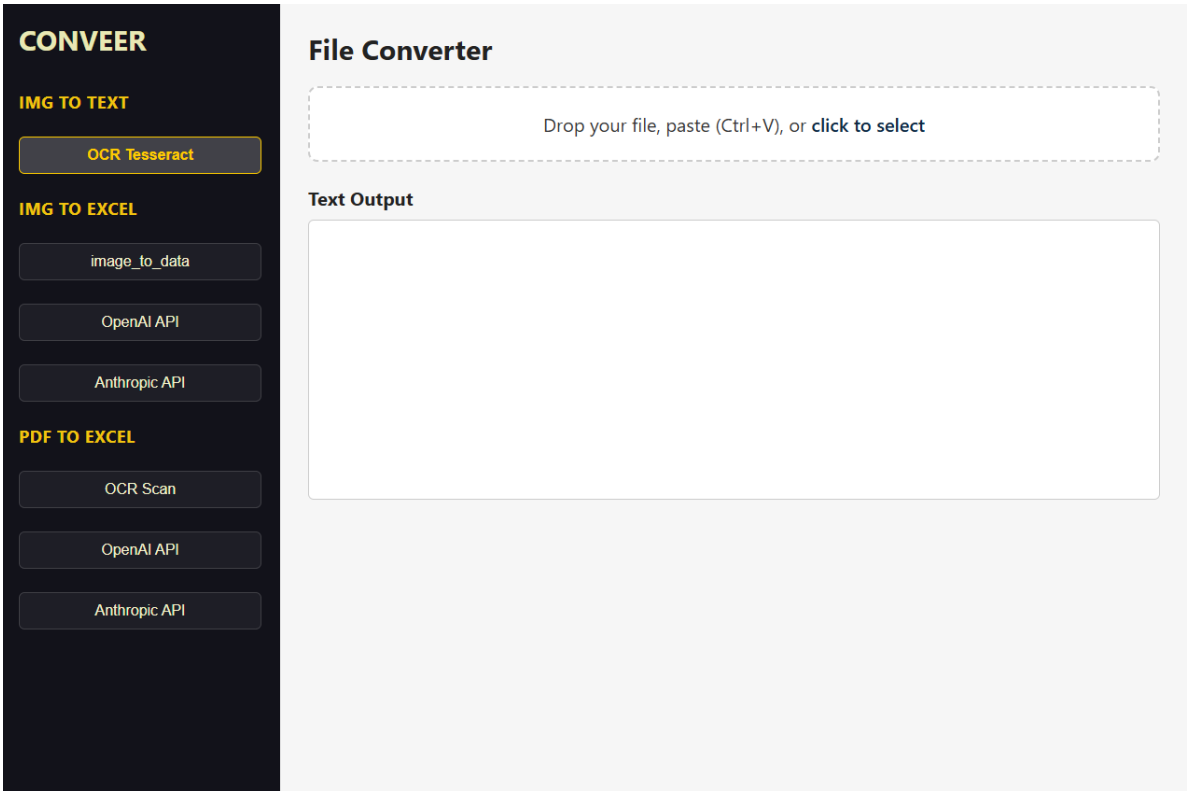


ER-діаграма бази даних проміжного JSON:

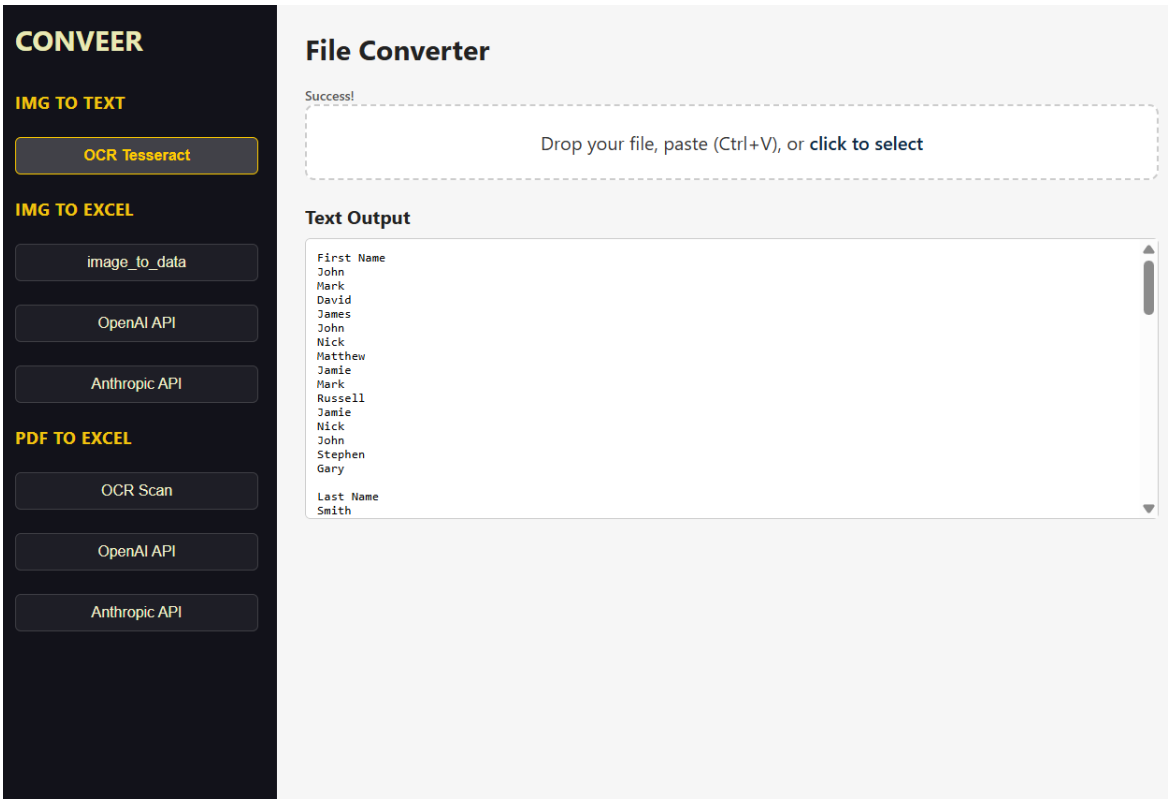


ДОДАТОК Б

Інтерфейс Conveer:



Вивід при використанні моделі IMG TO TEXT:



Вивід при використанні моделі image_to_data блоку IMG TO EXCEL:

	A	B	C	D	E	F	G
1	First	Name	Last	Name	Age	Salary	Date
2	John			Smith	19	18,000	1404/2003
3	Mark			Collins		24,000	19/02/2002
4	David			Everett	7	16,500	03/07/2003
5	James			Taylor		20,000	24/10/2003
6	John			Hesketh	18	19,500	16/11/2002
7	Nick			Martin	21	24,000	10/08/2002
8	Matthew			Lancaster	19	26,500	27/06/2003
9	Jamie			Cannon	7	18,500	01/07/2002
10	Mark			Berry		29,000	26/11/2002
11	Russell			Proctor	15	14500	20/03/2002
12	Jamie			Richards	18	19,500	17/05/2003
13	Nick			Appletree		28,500	05/12/2002
14	John			Skinner	21	18,500	28/04/2003
15	Stephen			Stevens	19	21,500	08/07/2003
16	Gary			King		24,000	14/11/2002

Download Edited Excel

ДОДАТОК В

Код файлу image_preprocessor.py, що відповідає за попередню обробку зображення:

```
import cv2
import numpy as np
from PIL import Image
import pytesseract

def rotate_image_cv(img, angle):
    if angle == 90:
        return cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE)
    elif angle == 180:
        return cv2.rotate(img, cv2.ROTATE_180)
    elif angle == 270:
        return cv2.rotate(img, cv2.ROTATE_90_COUNTERCLOCKWISE)
    return img

def score_rotation_structure(pil_img: Image.Image) -> int:
    df = pytesseract.image_to_data(pil_img, output_type=pytesseract.Output.DATAFRAME)
    df = df.dropna(subset=['text'])
    df = df[df.conf != -1]
    df = df[(df.text.str.strip() != '') & (df.text.str.len() > 0)]

    if df.empty:
        return 0

    lines = []
    used = set()
    y_threshold = df['height'].mean() * 0.6

    for idx1, word1 in df.iterrows():
        if idx1 in used:
            continue
```

```
        line = [word1]
        y1 = word1['top'] + word1['height'] / 2
        for idx2, word2 in df.iterrows():
            if idx2 in used or idx2 == idx1:
                continue
            y2 = word2['top'] + word2['height'] / 2
            if abs(y1 - y2) < y_threshold:
                line.append(word2)
                used.add(idx2)
        used.add(idx1)
        if len(line) >= 3:
            lines.append(line)

    return len(lines)

def extract_table_region(pil_image: Image.Image) -> Image.Image:
    image = np.array(pil_image.convert("RGB"))
    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)
    edged = cv2.Canny(blurred, 75, 200)
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (15, 5))
    closed = cv2.morphologyEx(edged, cv2.MORPH_CLOSE, kernel)

    contours, _ = cv2.findContours(closed.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    table_contour = None
    max_area = 0

    for cnt in contours:
        area = cv2.contourArea(cnt)
        x, y, w, h = cv2.boundingRect(cnt)
        if area > max_area and w > 100 and h > 100:
            table_contour = (x, y, w, h)
            max_area = area
```

```
    if table_contour:
        x, y, w, h = table_contour
        cropped = image[y:y+h, x:x+w]

        best_score = -1
        best_img = cropped

        for deg in [0, 90, 180, 270]:
            rot = rotate_image_cv(cropped, deg)
            pil = Image.fromarray(rot)
            score = score_rotation_structure(pil)
            if score > best_score:
                best_score = score
                best_img = rot

        return Image.fromarray(best_img)

    return pil_image
```

Код файлу model2.py, що відповідає за використання методу image_to_data з додатковими налаштуваннями:

```
from models.base_model import BaseModel
from PIL import Image
import pytesseract
import pandas as pd
import tempfile
import re
import numpy as np
from models.image_preprocessor import extract_table_region

class Model2(BaseModel):
    def clean_text(self, text):
        return re.sub(r"^[\[\]|_•\-__]+|[\[\]|_•\-]+$", "", text).strip("(){}:., ").strip()
```

```
def process(self, image: Image.Image) -> str:
    image = extract_table_region(image)
    df = pytesseract.image_to_data(image, output_type=pytesseract.Output.DATAFRAME)
    df = df.dropna(subset=['text'])
    df = df[df.conf != -1]
    df = df[(df.text.str.strip() != '') & (df.text.str.len() > 0)]

    if df.empty:
        return "No text found in image"

    lines = []
    used = set()
    average_height = df['height'].mean()
    y_threshold = average_height * 0.7

    for idx1, word1 in df.iterrows():
        if idx1 in used:
            continue

        line = [word1]
        y1 = word1['top'] + word1['height'] / 2

        for idx2, word2 in df.iterrows():
            if idx2 in used or idx2 == idx1:
                continue
            y2 = word2['top'] + word2['height'] / 2
            if abs(y1 - y2) < y_threshold:
                line.append(word2)
                used.add(idx2)

        used.add(idx1)
        lines.append(line)

    lines.sort(key=lambda line: min(word['top'] for word in line))
```

```
if len(lines) == 0:
    return "No lines detected"

header_words = pd.DataFrame(lines[0])
header_words = header_words.sort_values(by='left')
column_x_centers = [
    word['left'] + word['width'] // 2 for _, word in header_words.iterrows()
]
column_count = len(column_x_centers)
header_row = [
    self.clean_text(word['text']) for _, word in header_words.iterrows()
]

structured_rows = [header_row]

for line in lines[1:]:
    words = pd.DataFrame(line)
    words = words.sort_values(by='left')
    row = [""] * column_count

    for _, word in words.iterrows():
        cleaned = self.clean_text(word['text'])
        if not cleaned:
            continue
        word_center = word['left'] + word['width'] // 2
        distances = [abs(word_center - cx) for cx in column_x_centers]
        min_dist = min(distances)
        if min_dist > 60:
            continue
        col_idx = int(np.argmin(distances))
        if row[col_idx]:
            row[col_idx] += " " + cleaned
        else:
            row[col_idx] = cleaned
```

```
            structured_rows.append(row)

        max_cols = max(len(r) for r in structured_rows)
        padded = [r + [""] * (max_cols - len(r)) for r in structured_rows]
        df_out = pd.DataFrame(padded)

        with tempfile.NamedTemporaryFile(delete=False, suffix='.xlsx') as tmp:
            file_path = tmp.name
        df_out.to_excel(file_path, index=False, header=False)
        return file_path
```

Фрагмент коду файлу pdf_model_anthropic.py, в якому задається промпт для Anthropic API:

```
    def build_prompt(self):
        return (
            "Extract the table from this image and return it as a Python list of rows. "
            "Each row should be a list of cell values. Return only the list — no commentary or explanation.\n\n"
            "Example:\n"
            '[["Customer", "Region", "Date"], ["Acme", "North", "2024-05-25"]]'
        )

    def ask_claude(self, b64_img):
        response = self.client.messages.create(
            model="claude-3-opus-20240229",
            max_tokens=1024,
            temperature=0,
            messages=[
                {
                    "role": "user",
                    "content": [
                        {"type": "text", "text": self.build_prompt()},
```

```
                {
                    "type": "image",
                    "source": {
                        "type": "base64",
                        "media_type": "image/png",
                        "data": b64_img
                    }
                }
            ]
        }
    ]
)
return response.content[0].text.strip()
```

Фрагмент коду файлу pdf_model_ocr.py, в якому використано попередню функцію image_to_data, з використанням pdf2image:

```
from models.base_model import BaseModel
import pandas as pd
import numpy as np
import pytesseract
from pdf2image import convert_from_bytes
from PIL import Image
import tempfile
import re

class PDFModelOCR(BaseModel):
    def clean_text(self, text):
        return re.sub(r"[^\w\d,./:-]", "", str(text)).strip()

    def normalize_number(self, text):
        return re.sub(r"[^\d.]", "", text.replace(",", "").replace(" ", "")).strip()

    def group_words_into_lines(self, df):
```

```
    lines = []
    used = set()
    y_threshold = df['height'].mean() * 0.6

    for idx1, word1 in df.iterrows():
        if idx1 in used:
            continue
        line = [word1]
        y1 = word1['top'] + word1['height'] / 2
        for idx2, word2 in df.iterrows():
            if idx2 in used or idx2 == idx1:
                continue
            y2 = word2['top'] + word2['height'] / 2
            if abs(y1 - y2) < y_threshold:
                line.append(word2)
                used.add(idx2)
        used.add(idx1)
        lines.append(line)

    lines.sort(key=lambda l: min(w['top'] for w in l))
    return lines

def align_line_by_spacing(self, line, spacing_threshold=40):
    line_df = pd.DataFrame(line).sort_values(by='left')
    columns = []
    last_right = None

    for _, word in line_df.iterrows():
        text = self.clean_text(word['text'])
        if not text:
            continue

        left = word['left']
        if last_right is None or left - last_right > spacing_threshold:
            columns.append(text)
```

```
            else:
                columns[-1] += ' ' + text
            last_right = word['left'] + word['width']
        return columns

    def ocr_image_to_rows(self, image: Image.Image) -> list:
        df = pytesseract.image_to_data(image, output_type=pytesseract.Output.DATAFRAME)
        df = df.dropna(subset=['text'])
        df = df[df.conf != -1]
        df = df[(df.text.str.strip() != '') & (df.text.str.len() > 0)]

        if df.empty:
            return []

        lines = self.group_words_into_lines(df)

        rows = []
        for line in lines:
            raw_row = self.align_line_by_spacing(line, spacing_threshold=40)
            row = []
            for cell in raw_row:
                cleaned = self.clean_text(cell)

                # Keep slashes for dates like 14/04/2003
                if re.match(r'^\d{1,2}/\d{1,2}/\d{2,4}$', cleaned):  # looks like a date
                    row.append(cleaned)
                elif re.match(r'^\d[\d\s,./-]*$', cleaned):  # looks like number
                    row.append(self.normalize_number(cleaned))
                else:
                    row.append(cleaned)
            rows.append(row)
        return rows

    def process(self, pdf_bytes: bytes) -> str:
        images = convert_from_bytes(pdf_bytes)
```

```
all_rows = []

for img in images:
    rows = self.ocr_image_to_rows(img)
    if rows:
        all_rows.extend(rows)

if not all_rows:
    return "No tables found via OCR."

max_cols = max(len(r) for r in all_rows)
padded = [r + [""] * (max_cols - len(r)) for r in all_rows]
df = pd.DataFrame(padded)

with tempfile.NamedTemporaryFile(delete=False, suffix=".xlsx") as tmp_xlsx:
    path = tmp_xlsx.name
df.to_excel(path, index=False, header=False)
return path
```