НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО
Декан факультету (Директор ННІ)
інформаційних технологій
(назва факультету (ННІ))

_____ Ігор Болбот
(підпис) (ім'я ПРІЗВИЩЕ)

"___" _____ 2025 р.

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри
комп'ютерних наук
(назва кафедри)

_____ Белла Голуб
(підпис) (ім'я ПРІЗВИЩЕ)

"___" _____ 2025 р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему Програмне забезпечення розпізнавання алфавітно-цифрової інформації

Спеціальність 121 «Інженерія програмного забезпечення»
(код і найменування)

Освітня програма Програмне забезпечення інформаційних систем
(назва)

Орієнтація освітньої програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Гарант освітньої програми
к.ф.-м.н., доцент _____ Віктор Кириченко
(науковий ступінь та вчене звання) (підпис) (ім'я ПРІЗВИЩЕ)

Керівник магістерської кваліфікаційної роботи
к.ф.-м.н., доцент _____ Володимир СЕМКО
(науковий ступінь та вчене звання) (підпис) (ім'я ПРІЗВИЩЕ)

Виконав _____ Віталій ВДОВИЧЕНКО
(підпис) (ім'я ПРІЗВИЩЕ здобувача)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет (ННІ) __інформаційних технологій_____

ЗАТВЕРДЖУЮ
Завідувач кафедри__комп'ютерних наук_
___доцент, к.т.н._____ _____ Голуб Б. Л._
(науковий ступінь, вчене звання) (підпис) (ПІБ)
"_10_" ___вересня_____2025 року

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТУ

_____ Вдовиченко Віталій Вячеславович_____
(прізвище, ім'я, по батькові)

Спеціальність 121 «Інженерія програмного забезпечення»
(код і назва)

Освітня програма ____ Програмне забезпечення інформаційних систем
(назва)

Орієнтація освітньої програми_____освітньо-професійна_____

Тема магістерської кваліфікаційної роботи Програмне забезпечення розпізнавання алфавітно-цифрової інформації
затверджена наказом ректора НУБіП України від "_01_"_листопада____2024р. №1963 «С»
Термін подання завершеної роботи на кафедру _. _29.11.2025_____
(рік, місяць, число)

Вихідні дані до магістерської кваліфікаційної роботи: є зображення текстових об'єктів з друкованим, рукописним або машинно-генерованим алфавітно-цифровим вмістом, представлені у вигляді растрових файлів (JPEG, PNG, BMP, TIFF) або у форматі відеопотоку (кадри з камер спостереження, мобільних пристроїв, сканерів).
Перелік питань, що підлягають дослідженню:

1.Аналіз основних факторів, що впливають на точність розпізнавання алфавітно-цифрової інформації в зображеннях з різною якістю та типом викривлень.

2.Дослідження методів попередньої обробки зображень для підвищення ефективності сегментації та нормалізації символів.

3.Оцінка застосування класичних та нейромережевих алгоритмів для розпізнавання символів і структурування тексту.

4.Визначення вимог до архітектури програмного забезпечення для забезпечення модульності, масштабованості та підтримки оновлюваних моделей.

Дата видачі завдання "__01__" __листопада_____2024_ р.

Керівник магістерської кваліфікаційної роботи _____ _Семко В.В.
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ _____ Вдовиченко В.В.
(підпис) (прізвище та ініціали студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

1. API – Application Programming Interface, інтерфейс взаємодії програмних компонентів.
2. CNN – Convolutional Neural Network, згорткова нейронна мережа для розпізнавання образів.
3. CRNN – Convolutional Recurrent Neural Network, гібридна модель для послідовного розпізнавання символів.
4. OCR – Optical Character Recognition, оптичне розпізнавання текстових даних.
5. OLAP – Online Analytical Processing, технологія багатовимірного аналізу даних.
6. PCA – Principal Component Analysis, метод головних компонент для зниження розмірності.
7. k-means – алгоритм кластеризації, що мінімізує внутрішньокластерну відстань SSE.
8. SSE – Sum of Squared Errors, сумарна квадратична похибка усередині кластерів.
9. KPI – Key Performance Indicator, ключовий показник ефективності системи.
10. conf – Confidence Score, рівень впевненості моделі в розпізнаному символі.
11. JSON – JavaScript Object Notation, формат структурованих відповідей REST-API.
12. REST – Representational State Transfer, архітектурний стиль побудови веб-сервісів.
13. TLS – Transport Layer Security, протокол криптографічного захисту передавання даних.
14. PDF – Portable Document Format, формат цифрових документів.

15. DPI – Dots Per Inch, показник роздільної здатності зображення.
16. ML – Machine Learning, машинне навчання.

ВСТУП

Зростання обсягів візуальної інформації в ділових, технічних і адміністративних процесах, зокрема сканованих документів, фотофіксацій, відеокадрів і форм, створює запит на ефективні автоматизовані засоби перетворення зображень у структуровану текстову форму.

Це особливо актуально в умовах цифрової трансформації підприємств, коли ключовим стає зменшення залежності від ручного вводу, підвищення точності даних і забезпечення сумісності з іншими інформаційними системами. Сучасні технології оптичного розпізнавання символів (OCR) часто демонструють обмежену адаптивність до реальних умов - змінної якості зображень, різноманіття шрифтів, наявності артефактів, а також специфіки української мови й форматів запису алфавітно-цифрової інформації. Водночас класичні реалізації OCR-систем рідко надають достатній рівень конфігурованості, обробки винятків або інтеграції з бізнес-логікою організацій. Це формує необхідність у розробці інтелектуального програмного забезпечення для розпізнавання символів, яке поєднує гнучку архітектуру, підтримку навчання моделей на спеціалізованих датасетах, модулі валідації результатів і можливість вбудовування в API-орієнтовані системи керування документами [1].

Метою дослідження є створення програмного забезпечення для розпізнавання алфавітно-цифрової інформації на зображеннях, яке забезпечує автоматичну локалізацію текстових областей, попередню обробку, класифікацію символів, валідацію результатів і формування структурованого вихідного представлення.

Для досягнення поставленої мети необхідно виконати такі **завдання:**

– проаналізувати сучасні методи та бібліотеки оптичного розпізнавання символів і визначити їхні обмеження;

– сформулювати функціональні та технічні вимоги до програмного забезпечення;

– побудувати UML-діаграми варіантів використання, активності, послідовності та компонентів;

– розробити архітектуру підсистем: завантаження зображень, препроцесингу, детекції тексту, класифікації, постобробки та API-комунікації;

– реалізувати програмний прототип із використанням Python, OpenCV, PyTorch і FastAPI;

– провести тестування точності розпізнавання на контрольних датасетах і оцінити продуктивність рішень на різних типах вхідних зображень.

Об’єктом дослідження є процеси автоматизованого вилучення текстової інформації з растрових зображень.

Предметом дослідження є інформаційні моделі, алгоритми і програмні компоненти, що реалізують цикл OCR-обробки: від зчитування зображення до виводу валідуваного тексту.

Методи дослідження включають: системний аналіз, об’єктно-орієнтоване моделювання, використання нейронних мереж для розпізнавання символів, а також програмну реалізацію з використанням Python-бібліотек (OpenCV, PyTorch, Tesseract) і сервісної інфраструктури FastAPI, uvicorn, SQLite, pytest.

Практична цінність полягає в можливості застосування розробленого програмного забезпечення для автоматичного оцифрування форм, документів, номерних знаків, медичних карт, бланків опитування та інших структурованих матеріалів, де присутня змішана алфавітно-цифрова інформація. Рішення легко інтегрується у веб-сервіси, мобільні додатки та системи електронного документообігу.

Наукова новизна полягає в поєднанні технологій комп’ютерного зору, сегментації й контекстної класифікації символів у цілісну архітектуру OCR-системи, здатної до адаптації під конкретні мови, формати та шаблони

введення, із вбудованою можливістю аномалійної фільтрації та регламентованої валідації результатів.

Апробація результатів

Структура роботи відображає повний життєвий цикл розробки програмного забезпечення:

у першому розділі проведено аналіз предметної області, огляд сучасних методів і постановку задачі;

у другому - побудовано UML-діаграми, сформовано вимоги та спроєктовано архітектуру компонентів;

третій розділ присвячено реалізації функціональних модулів, їхній інтеграції та механізмам валідації;

у четвертому - представлено результати тестування, аналіз продуктивності й оцінку точності OCR-модуля.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Предметна область програмного забезпечення для розпізнавання алфавітно-цифрової інформації охоплює сукупність процесів, методів і засобів, що забезпечують автоматизоване перетворення візуальних даних (зображень, сканів, відеокадрів) у машинно-читану текстову форму з високим ступенем достовірності. Сучасні OCR-системи виступають критично важливими компонентами в інформаційних системах документообігу, обліку, цифрової ідентифікації, логістики, фінансів, охорони здоров'я, забезпечуючи зменшення ручної праці, підвищення точності даних і прискорення бізнес-процесів. Класифікація систем розпізнавання здійснюється за кількома ключовими ознаками, що формують вимоги до архітектури, моделей даних і алгоритмів. На рис. 1.1 подано узагальнену схему класифікації OCR-систем.

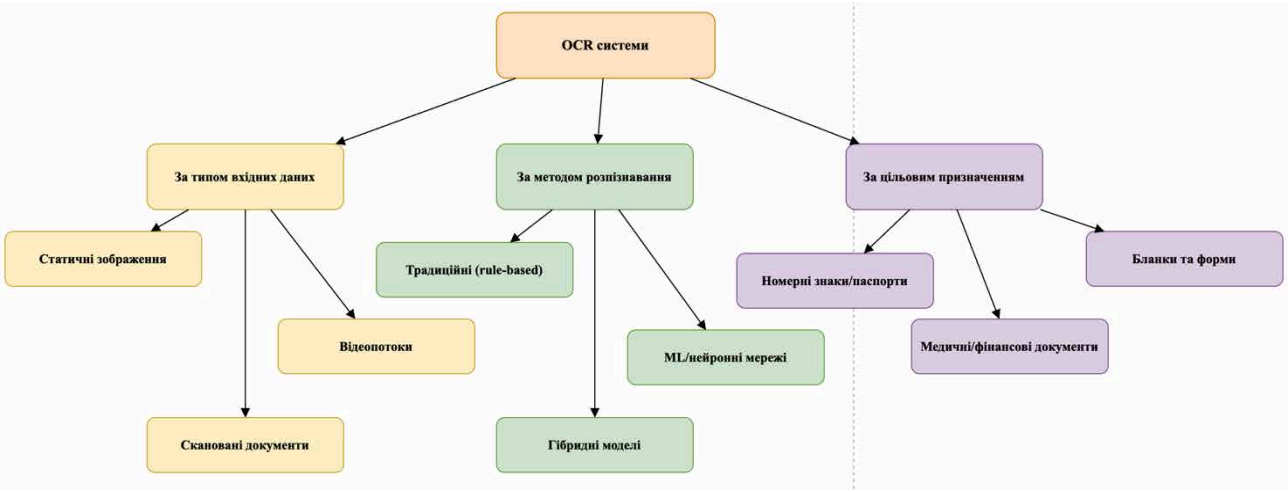

Рис. 1.1 – Класифікація систем розпізнавання алфавітно-цифрової інформації

Першою класифікаційною ознакою виступає тип вхідних даних, який визначає передобробку, точність позиціонування символів і ресурсоємність алгоритмів. До типових джерел належать статичні зображення (наприклад, фотографії документів), скановані сторінки, а також відеопотоки з мобільних або стаціонарних камер. Наступною ознакою є метод розпізнавання, що

включає класичні rule-based системи (на основі евристик), нейромережеві моделі, а також гібридні підходи, в яких застосовується попередня класифікація сегментів, подальше уточнення або валідація результатів. Третім критерієм класифікації є цільове призначення, що визначає особливості семантичної структури вхідних даних і вимоги до точності. Зокрема, розрізняють системи для обробки бланків і форм (де важлива розмітка), документи ідентифікації (паспорти, ID), номерні знаки, а також специфічні галузеві рішення - медичні, фінансові та юридичні документи.

Для систематизації основних характеристик предметної області сформовано аналітичну таблицю, що узагальнює типи джерел, характер розпізнаваних символів, складність обробки та типову архітектуру розв'язків. Узагальнені дані наведено в табл. 1.1.

Таблиця 1.1 – Основні характеристики підкласів OCR-систем

№	Тип джерела	Формат символів	Складність розпізнавання	Типова архітектура
1	Сканований документ	Латиниця, кирилиця	Низька–середня	Tesseract + постфільтри
2	Фото з мобільного	Алфавіт + цифри	Середня–висока	CNN + CTC decoder
3	Відео (реєстрація)	Цифри (номерні знаки)	Висока	YOLO + CRNN pipeline
4	Медичні бланки	Алфавіт + спецсимволи	Середня	UNet + attention OCR
5	Ідентифікаційні картки	Змішаний	Висока	Segmenter + LSTM OCR

Класифікація систем за типом вхідних даних, підходом до розпізнавання та цільовим призначенням визначає вимоги до архітектури програмного забезпечення, вибору моделей, форматів обробки та точності. Це обґрунтовує необхідність побудови адаптивних, модульних OCR-рішень, здатних до роботи в умовах різнорідних джерел, нестабільної якості вхідних зображень і високих вимог до точності, а також до подальшої формалізації функціональних вимог і технічних характеристик системи, що розробляється.

1.2 Огляд інформаціних джерел та існуючих рішень

Розвиток систем оптичного розпізнавання символів (OCR) здійснюється у двох взаємопов’язаних напрямках: з одного боку — розширення функціональності та мовної підтримки існуючих інструментів, з іншого — інтеграція нейронних мереж для підвищення точності та гнучкості. Першим прикладом є Tesseract OCR, який є одним із найвідоміших open source OCR-рушіїв. Він підтримує багатомовне розпізнавання, декілька рівнів сегментації сторінки та інтеграцію з бібліотекою Leptonica, що дозволяє працювати з TIFF, PNG, JPEG (рис. 1.2). Tesseract широко використовується в задачах цифровізації архівів і формалізації текстових даних у сканованих документах, хоча демонструє обмежену продуктивність на складних зображеннях з шумом або з малоконтрастними символами.

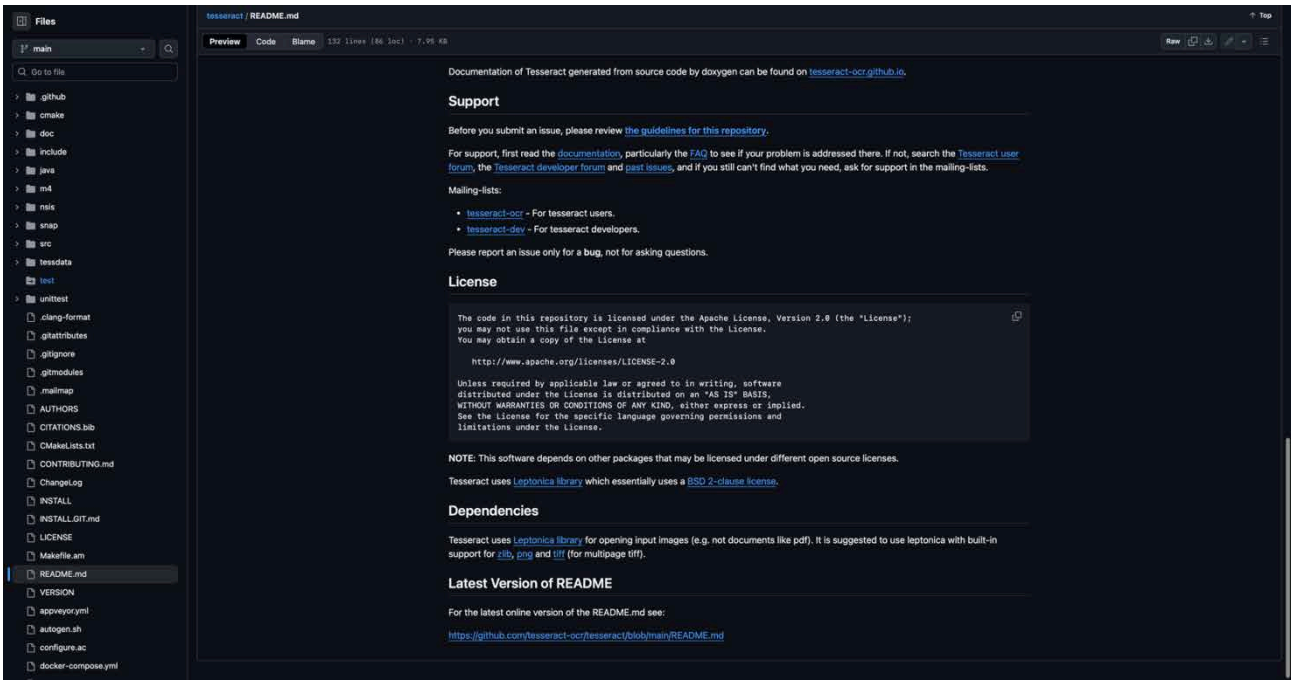

Рис. 1.2 – Вміст репозиторію та документація Tesseract OCR

Наступним прикладом є Google Cloud Vision OCR, який реалізує розпізнавання через хмарні сервіси з доступом через REST API. Цей продукт підтримує функції TEXT_DETECTION і DOCUMENT_TEXT_DETECTION, які дозволяють витягувати текст з вуличних знаків, бланків, щільно заповнених

документів і сканів. Результат повертається у форматі JSON з координатами, абзацами, словами, лініями і блоками, що забезпечує гнучкість при подальшій обробці (рис. 1.3). Сервіс має високу точність і зручну масштабованість, але потребує підключення до Google Cloud, що ускладнює автономне використання.

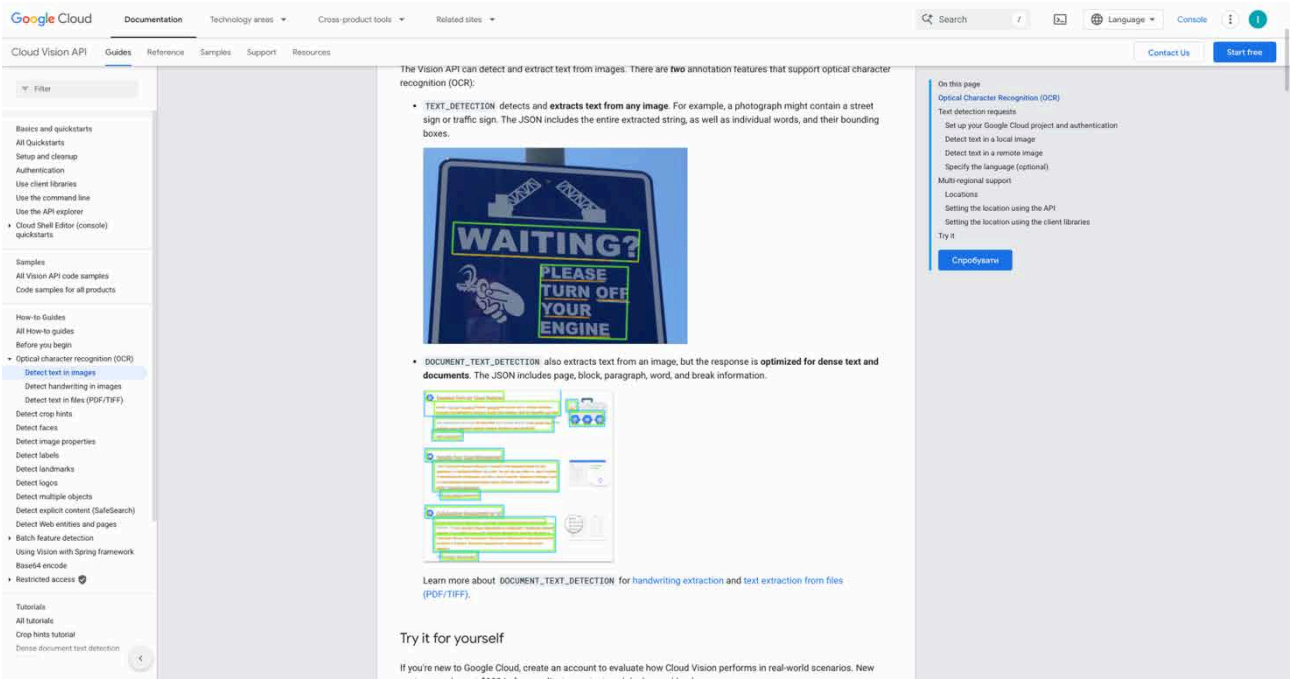

Рис. 1.3 – Приклад використання Cloud Vision API для текстових вивісок

Ще одним прикладом високоточних рішень є ABBYY FineReader Engine, який призначений для інтеграції в комерційні ПЗ-продукти. Платформа підтримує AI-оптимізовані алгоритми для обробки багатомовного друкованого та рукописного тексту, обробку таблиць, полів форм, а також аналіз структури документа. Система позиціонується як SDK для розробників, що потребують індустріального рівня точності, контролю якості та відповідності регламентам (рис. 1.4). До переваг належить підтримка складних сценаріїв, проте вона є платною та має обмеження в кастомізації моделей.

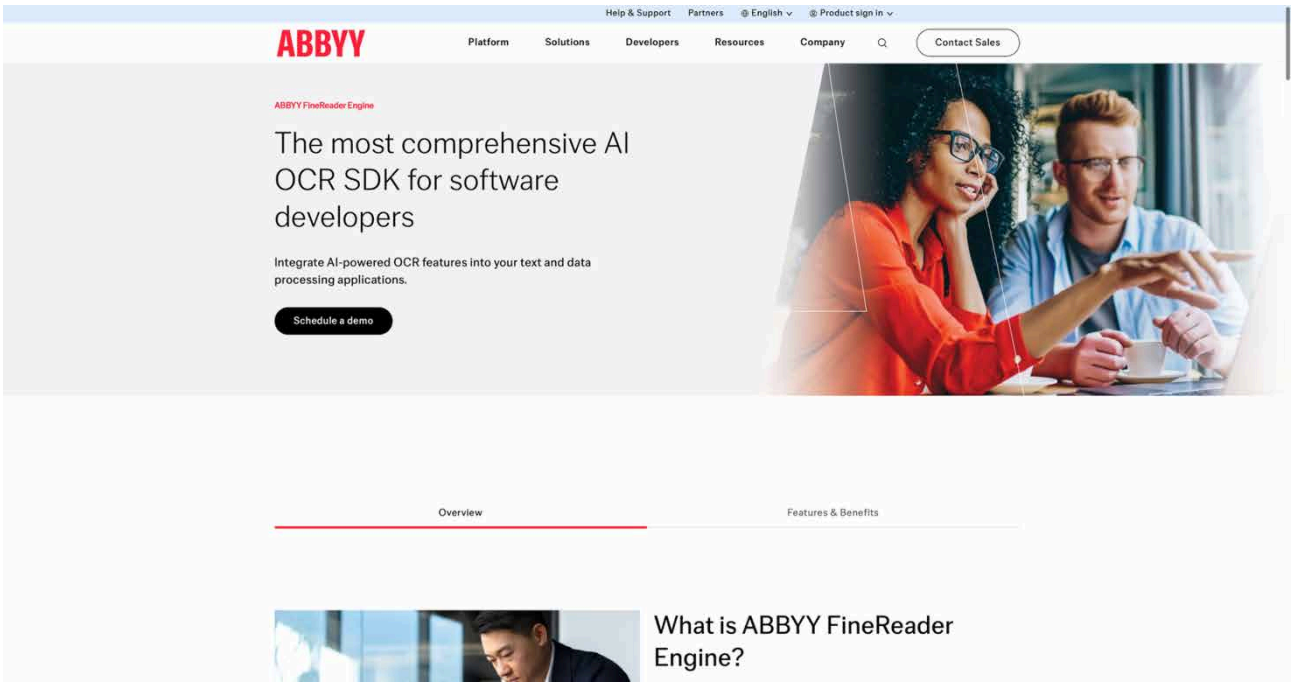

Рис. 1.4 – Головна сторінка ABBYY FineReader Engine SDK

На відміну від попередніх, бібліотека EasyOCR розроблена як lightweight-рішення з використанням нейромережевого стека ResNet + LSTM + CTC decoder. Вона працює з понад 80 мовами, не потребує переднавчання, має активне співтовариство та гнучку інтеграцію з PyTorch/OpenCV. Приклад її застосування наведено на рис. 1.5, де представлено розпізнавання вивісок, багатомовного тексту та результат JSON-виводу. Недоліком EasyOCR є відсутність модулів валідації, обмежена робота зі складними структурами документа, а також відсутність офіційної підтримки кастомних моделей.

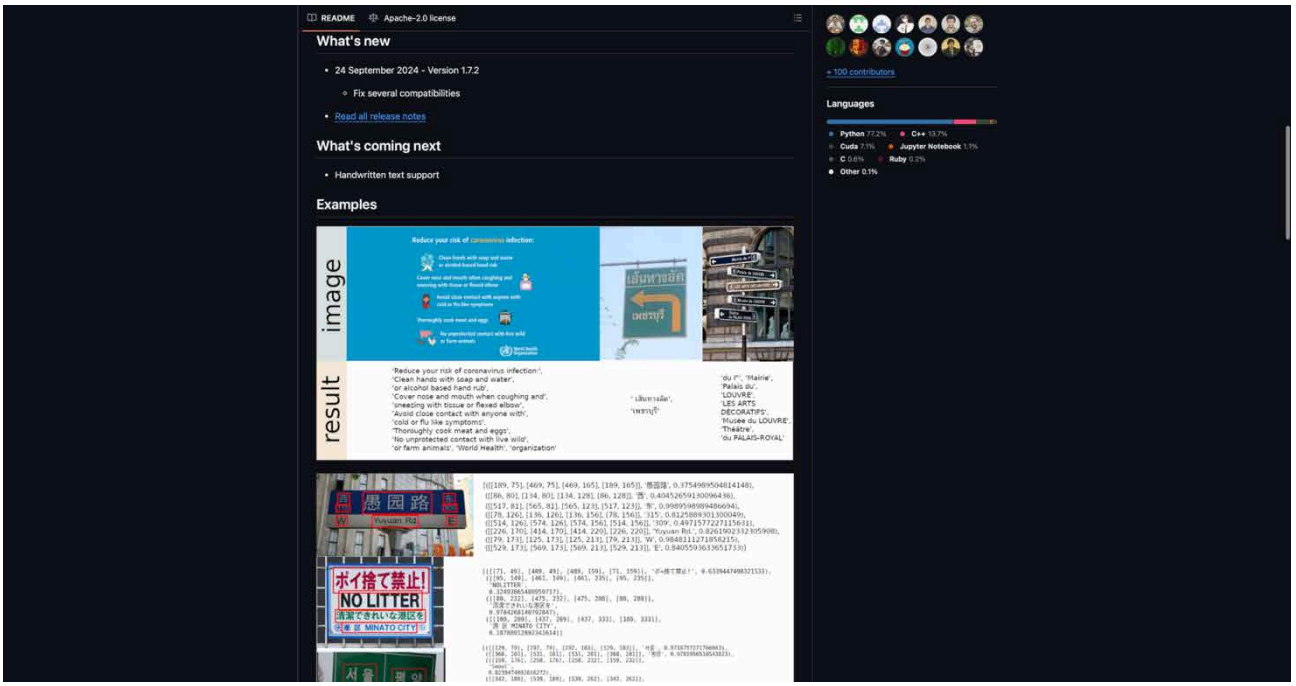

Рис. 1.5 – Демонстрація багатомовного розпізнавання в EasyOCR

Для систематизації переваг, недоліків і технічних характеристик наведених рішень доцільно використати табл. 1.2, де відображено ключові параметри порівняння з орієнтацією на функціональність, точність, гнучкість, вимоги до ресурсів і доступність. В останній колонці наведено коротку характеристику розроблюваної системи, що дозволяє позиціонувати її серед аналогів.

Таблиця 1.2 – Порівняння існуючих OCR-рішень та розроблюваної системи

№	Система	Тип розпізнавання	Мовна підтримка	Структурна обробка	Формат виводу	Гнучкість налаштувань	Ліцензія	Наша система
1	Tesseract OCR	LSTM + CTC	100+ мов	Частково	TXT/HOCR/ALTO	Середня	Apache 2.0	Підтримка сегментації, розширена валідація
2	Google Cloud Vision	CNN + хмара	50+ мов	Повна	JSON	Обмежена	Пропрієтарна	Локальне виконання, API,

								кастомні моделі

Продовження таблиці 1.2

3	ABBYY FineReader Engine	AI + rule-based	200+ мов	Повна	XML/ALTO/PDF	Висока	Комерційна	Підтримка розгортання, модульна архітектура
4	EasyOCR	ResNet + LSTM	80+ мов	Мінімальна	JSON	Висока	Apache 2.0	Вбудована логіка фільтрації та нормалізації

Розроблюване рішення поєднує переваги відкритих нейромережевих моделей з кастомізованою постобробкою, формалізованою структуризацією результатів і можливістю гнучкого налаштування на специфічні набори символів, мов і шаблонів. Це забезпечує високу придатність системи до інтеграції у корпоративні, мобільні й офлайн-сценарії використання.

Науково-технічні джерела, присвячені OCR-системам, концентруються на трьох ключових аспектах: архітектурах нейронних моделей, стійкості до спотворень вхідних даних і структурній обробці результатів. У сучасних публікаціях пріоритет надається поєднанню CNN-модулів для локалізації тексту з рекурентними (LSTM/GRU) або трансформерними декодерами для генерації символів у послідовності. Наприклад, Acharya та ін. запропонували комбінацію ResNet-архітектури з Bidirectional-LSTM і CTC loss-функцією для обробки реалістичних текстових зображень [1], тоді як Baek et al. у своїй праці продемонстрували ефективність модульної OCR-архітектури, що включає окремі блоки для препроцесингу, детекції, розпізнавання та валідації, з можливістю кастомізації моделей [2]. Дослідження також вказують на ефективність застосування attention-механізмів у трансформерах, зокрема для довгих послідовностей, та гнучких форматів постобробки. Окрему увагу приділено методам перевірки результату через регулярні вирази, структури шаблонів і контекстну валідацію. Ці підходи формують технічну основу для

розробки адаптивних OCR-систем із високою точністю, масштабованістю та підтримкою користувацьких правил.

1.3 Моделювання предметної області

Моделювання предметної області є базовим етапом формалізації вимог до програмного забезпечення, що розробляється, та визначає структуру взаємодії між користувачами, підсистемами, зовнішніми джерелами даних і внутрішніми сервісами системи. На рис. 1.6 представлено діаграму варіантів використання, що описує функціональні можливості ПЗ розпізнавання алфавітно-цифрової інформації відповідно до ролей користувачів та зовнішніх систем.

Рис. 1.6 – Діаграма варіантів використання ПЗ розпізнавання алфавітно-цифрової інформації

Як видно з діаграми, основні взаємодії охоплюють оператора (ініціатор процесу розпізнавання), адміністратора (управління параметрами, аудит),

аналітика якості (контроль точності та звітність), а також зовнішні компоненти — камери/сканери, корпоративні ERP-системи, хмарні сервіси OCR та IdP-платформи автентифікації. Серед ключових прецедентів — імпорт зображень/потоків, попередня обробка, виявлення текстових областей (ROI), класифікація символів з використанням CNN/LSTM/Transformer, сегментація, валідація, ручна корекція, експорт результатів і аудит. Структура варіантів використання враховує розширення (extend) і включення (include), що дозволяє гнучко адаптувати систему під різні сценарії — від розпізнавання бланків до обробки реєстраційних номерів.

На рис. 1.7 зображено діаграму послідовності, яка відображає міжкомпонентну взаємодію при виконанні основної операції розпізнавання. Ініціація процесу відбувається через графічний інтерфейс користувача, після чого клієнт надсилає зображення на сервер OCR-API.

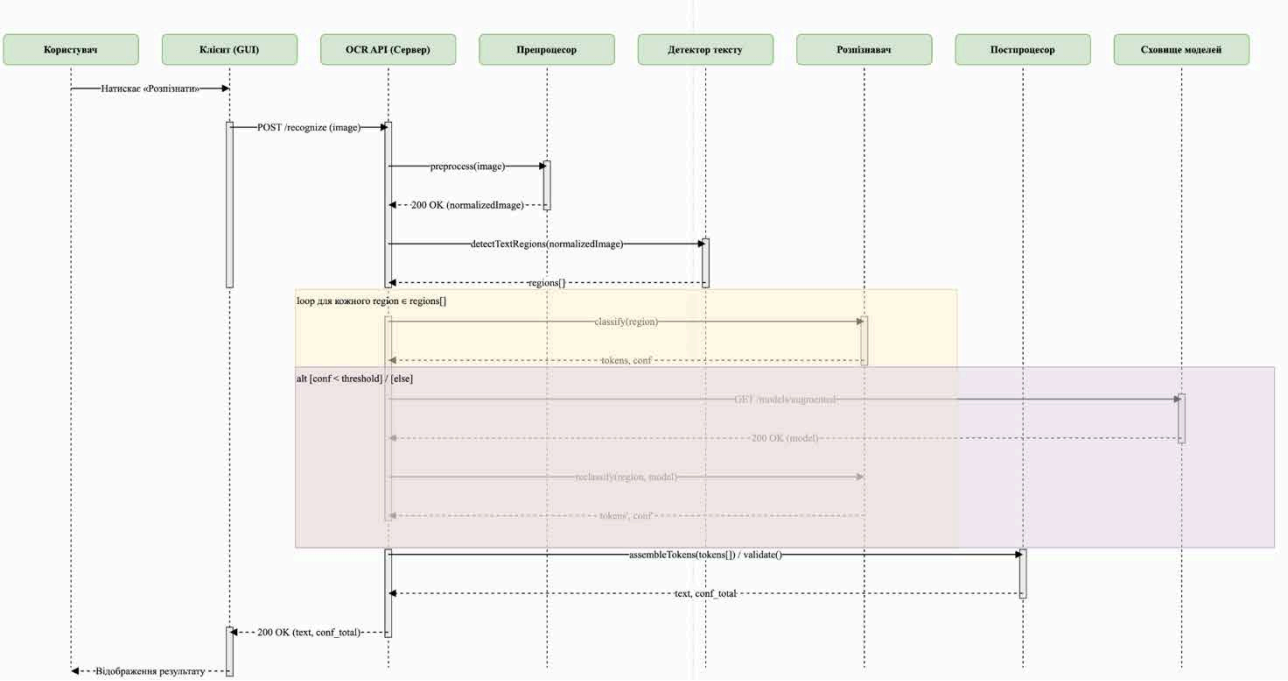

Рис. 1.7 – Діаграма послідовності взаємодії компонентів під час розпізнавання

На серверному боці зображення передається до препроцесора, де виконується нормалізація, бінаризація та подавлення шумів. Далі детектор тексту виділяє області (regions[]), для кожної з яких розпізнавач виконує

класифікацію символів. Якщо рівень довіри (confidence) недостатній, система звертається до сховища моделей для довантаження альтернативної нейронної мережі. Після остаточного формування токенів виконується постобробка: валідація через правила/regex, перевірка формату, збирання результату, який повертається користувачеві.

Бізнес-логіка процесу моделюється за допомогою діаграми активності (рис. 1.8), яка демонструє послідовність дій та умовні розгалуження залежно від наявності областей, точності класифікації та конфігурації фільтрів. Діаграма також відображає логіку повторної класифікації при зниженій точності, підтримку циклічної обробки кількох регіонів і точку входу/виходу в API-сервіс.

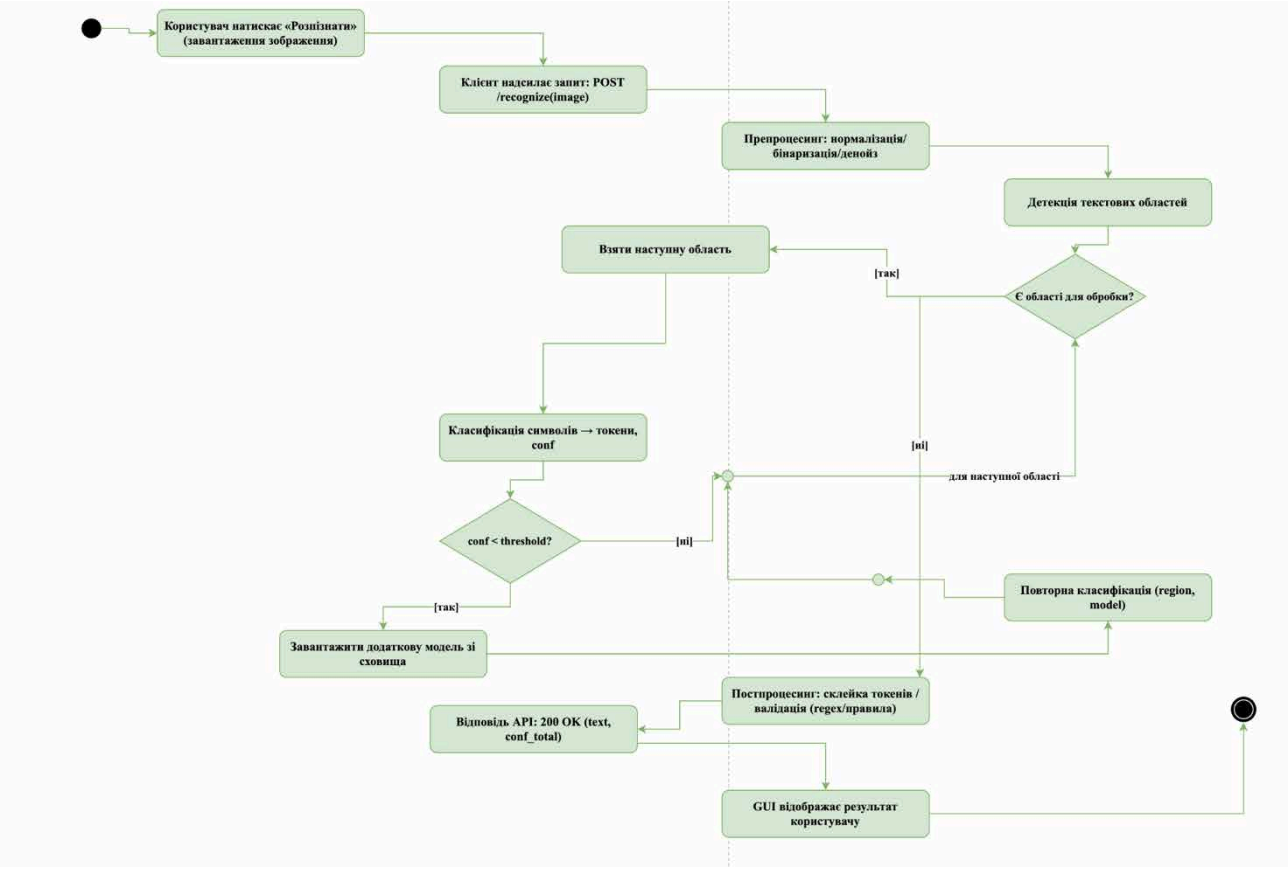

Рис. 1.8 – Діаграма активності основного процесу OCR-розпізнавання

Застосування трьох типів UML-моделей - варіантів використання, послідовності та активності - дозволяє формалізувати функціональні сценарії, динамічні взаємодії та деталі внутрішньої логіки системи. Ці моделі слугують основою для подальшого проєктування архітектури, формування інтерфейсів

взаємодії між компонентами та реалізації контролю коректності на рівні підсистем.

1.4 Аналіз вимог програмної системи

Аналіз вимог до програмного забезпечення для розпізнавання алфавітно-цифрової інформації базується на поетапному виокремленні функціональних, нефункціональних і технічних характеристик системи з урахуванням специфіки предметної області, архітектурної моделі та сценаріїв взаємодії, визначених у попередньому моделюванні. На основі сценаріїв використання, представлених на рис. 1.6–1.8, та логіки обробки зображень (див. п. 1.3), узагальнені функціональні вимоги подано в табл. 1.3.

Таблиця 1.3 – Функціональні вимоги до OCR-системи

№	Вимога	Опис реалізації
F1	Імпорт зображення або відеопотоку	Підтримка завантаження локальних зображень, сканів або кадрів із зовнішніх пристроїв (IP-камера, сканер)
F2	Автоматична попередня обробка	Нормалізація яскравості, шумозаглушення, бінаризація, ресайз
F3	Виявлення текстових областей (ROI)	Детекція потенційних фрагментів із текстом на основі геометричних/глибинних моделей
F4	Класифікація символів	Розпізнавання символів за допомогою моделей CNN/LSTM/Transformer
F5	Довантаження моделі при низькій довірі	Запит до сховища моделей для повторної класифікації при conf < threshold
F6	Постобробка та валідація	Склейка токенів, перевірка за шаблонами (regex), контроль формату
F7	Маскування конфіденційних даних	Автоматичне приховування ПІБ, серій, кодів, якщо вказано у політиках
F8	Ручне редагування результату	Інтерфейс ручної правки розпізнаного тексту з підсвічуванням невпевнених фрагментів
F9	Експорт результатів	Збереження у форматах CSV, JSON, XML
F10	Аудит, логування, версіювання	Фіксація подій API, зміни тексту, контроль версій

Нефункціональні вимоги визначають обмеження на поведінку системи та гарантують її якісні характеристики. До критичних аспектів належать

продуктивність, масштабованість, сумісність із зовнішніми API, доступність та відповідність вимогам захисту даних. Узагальнені нефункціональні вимоги представлено в табл. 1.4.

Таблиця 1.4 – Нефункціональні вимоги до OCR-системи

№	Вимога	Опис
NF1	Продуктивність	Обробка 1 зображення ≤ 1 с (на CPU), або ≤ 300 мс (на GPU), з підтримкою batch-режиму
NF2	Масштабованість	Горизонтальне масштабування через брокери черги (Redis, RabbitMQ)
NF3	Інтероперабельність	REST API з OpenAPI-специфікацією; підтримка інтеграції з ERP/ECM системами
NF4	Відмовостійкість	Підтримка повтору при таймаутах, черги недоставлених запитів
NF5	Аудит і прозорість	Логування ключових етапів обробки, формування логів змін
NF6	Безпека	JWT-автентифікація, RBAC, валідація вхідних файлів, фільтрація шкідливого контенту
NF7	Доступність	Доступ до GUI та API не менше ніж 99.5% часу
NF8	Міжплатформеність	Підтримка Linux/macOS/Windows у CLI/GUI; контейнеризація через Docker
NF9	Локалізація	Підтримка кирилиці, латиниці, чисел; UI українською й англійською

Технічні вимоги деталізують інфраструктурні, програмні та апаратні обмеження, необхідні для коректного функціонування системи на цільових платформах. Зведена характеристика представлена у табл. 1.5.

Таблиця 1.5 – Технічні вимоги до OCR-системи

№	Компонент	Вимога
T1	Платформа	Python 3.10+, FastAPI, PyTorch, OpenCV, PyQt6
T2	БД	PostgreSQL ≥ 13; Redis для кешування
T3	Інтерфейс	GUI (PyQt6), CLI (Typer), REST API
T4	Контейнеризація	Docker, підтримка Docker Compose
T5	GPU-інтеграція	Підтримка CUDA ≥ 11.3, GPU inference (optional)
T6	Зовнішні API	REST інтерфейси до ERP, SSO, OCR-сервісів (Google Cloud Vision, ABBYY SDK тощо)
T7	Ресурси	RAM ≥ 4GB, CPU ≥ 2 cores, Storage ≥ 1GB, при GPU – VRAM ≥ 2GB

T8	CI/CD	Інтеграція з GitHub Actions або GitLab CI, деплой через Docker Hub / SSH
T9	Тестування	Підтримка unit/integration тестів через Pytest, API-тести через Postman/Newman

На основі аналізу вимог сформовано цілісну систему специфікацій, що охоплює як функціональну логіку розпізнавання, так і якісні, технічні й інфраструктурні обмеження. Ці вимоги будуть покладені в основу подальшого архітектурного проєктування, моделювання бази даних і реалізації окремих підсистем.

1.5 Постановка завдання

Було вивчено сучасні підходи до побудови OCR-систем, зокрема гібридні нейронні архітектури, модульні сервіси препроцесингу й постобробки, а також типові вимоги до точності, стійкості, конфігурованості й захисту даних. Сформовано перелік функціональних сценаріїв системи, зокрема: імпорт зображень, детекція текстових областей, класифікація символів, повторна обробка при зниженій довірі, постобробка результату, експорт і маскування чутливої інформації. На основі вимог побудовано діаграми варіантів використання, послідовності та активності, які формалізують бізнес-логіку OCR-сервісу, включно з API-взаємодією, ролями користувачів, інтерфейсами ручного редагування, звітністю та контролем якості. Також було створено таблиці функціональних, нефункціональних і технічних вимог, що деталізують очікувану поведінку системи, обмеження продуктивності, вимоги до інтерфейсів, інфраструктурні залежності та форматування результатів.

На основі проведеного аналізу сформульовано перелік конкретизованих технічних завдань, реалізація яких забезпечить проєктування та впровадження цілісного, адаптивного OCR-рішення:

– реалізувати графічний інтерфейс користувача з використанням PyQt6, що дозволяє обирати зображення, запускати розпізнавання, переглядати результат і вносити ручні правки;

– побудувати серверну частину на FastAPI, з REST-архітектурою, підтримкою OpenAPI-специфікацій і асинхронною обробкою запитів до моделей;

– створити модуль препроцесингу з фільтрацією шуму, нормалізацією яскравості, бінаризацією й масштабуванням вхідного зображення;

– реалізувати детектор текстових областей із підтримкою векторного представлення ROI для подальшої класифікації;

– впровадити модуль розпізнавання символів на основі моделей CNN/LSTM або Transformer з механізмами довантаження альтернативних моделей при низькій довірі;

– забезпечити постобробку результату (склейка токенів, regex-валідація, маскування чутливих фрагментів, валідація структурованих даних);

– реалізувати модуль експорту у формати CSV, JSON, XML, з можливістю інтеграції в зовнішні системи (ERP, ECM);

– додати підтримку інтеграції з хмарними OCR API (Google Cloud Vision, ABBYY SDK) як альтернативне джерело розпізнавання;

– реалізувати систему логування подій, журналювання змін результатів і контрольних метрик точності;

– підготувати середовище контейнеризації (Docker) та CI/CD-конвеєр для автоматизованого тестування, деплою та оновлення системи.

Поставлені завдання відображають перехід від системного аналізу до архітектурного й прикладного проєктування OCR-системи з відкритою, гнучкою структурою, придатною до розширення, адаптації до різних форматів вхідних даних і вимог до точності у галузевих застосуваннях.

1.6 Висновки до першого розділу

У першому розділі було здійснено комплексний системний аналіз предметної області розпізнавання алфавітно-цифрової інформації, що дозволило сформувати цілісне уявлення про сучасний стан технологій OCR, їх

обмеження та практичні виклики, притаманні реальним умовам обробки зображень. Проаналізовано типи вхідних даних, методи розпізнавання, класифікацію систем за призначенням, а також наведено порівняльну характеристику провідних рішень - Tesseract, Google Cloud Vision, ABBYY FineReader Engine та EasyOCR. У ході огляду встановлено, що наявні системи демонструють високу ефективність у стандартизованих умовах, проте недостатньо придатні для роботи з різноманітними джерелами даних, неоднорідною якістю зображень і специфічними вимогами бізнес-логіки, що обґрунтовує актуальність розроблення адаптивного OCR-рішення.

У межах моделювання предметної області побудовано UML-діаграми варіантів використання, послідовності та активності, які формалізують ключові сценарії роботи системи, взаємодію між її компонентами та логіку циклу обробки зображень. Сформовано функціональні, нефункціональні та технічні вимоги до програмного забезпечення, що визначають очікувану продуктивність, безпеку, масштабованість, інтероперабельність та якість результатів. Додатково проаналізовано архітектурні підходи, властиві сучасним OCR-модулям, зі зосередженням на гібридних нейромережевих моделях, засобах препроцесингу, постобробки та контекстної валідації. На основі виконаного аналізу сформульовано узагальнену постановку завдання, яка окреслює необхідність побудови модульної, гнучкої та масштабованої програмної системи для високоточного розпізнавання алфавітно-цифрової інформації з підтримкою кастомних моделей і розширюваних механізмів валідації результатів. Такий фундамент є необхідною передумовою для подальшого архітектурного проєктування і реалізації програмного забезпечення у наступних розділах.

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Логічна модель даних визначає формалізовану структуру інформаційного простору програмної системи, описуючи сутності, їх атрибути та взаємозв’язки, необхідні для забезпечення узгодженості, цілісності й ефективності зберігання даних. Вона є проміжною ланкою між концептуальним проєктуванням і фізичною реалізацією бази даних, виступаючи основою для побудови SQL-схем, нормалізації таблиць і визначення ключових зв’язків між об’єктами. Логічна модель орієнтована на представлення предметної області в уніфікованому вигляді, де кожна сутність відображає об’єкт або процес, а відношення між ними - взаємодію та підпорядкування інформаційних структур. У контексті розроблюваної OCR-системи логічна модель даних охоплює документи, сторінки, рядки, токени, області розпізнавання, результати, моделі, а також звіти, які забезпечують аналітичну й сервісну підтримку системи.

На рис. 2.1 подано логічну ER-модель бази даних системи розпізнавання алфавітно-цифрової інформації, побудовану з урахуванням принципів третьої нормальної форми (3NF) для уникнення дублювання та підвищення узгодженості даних.

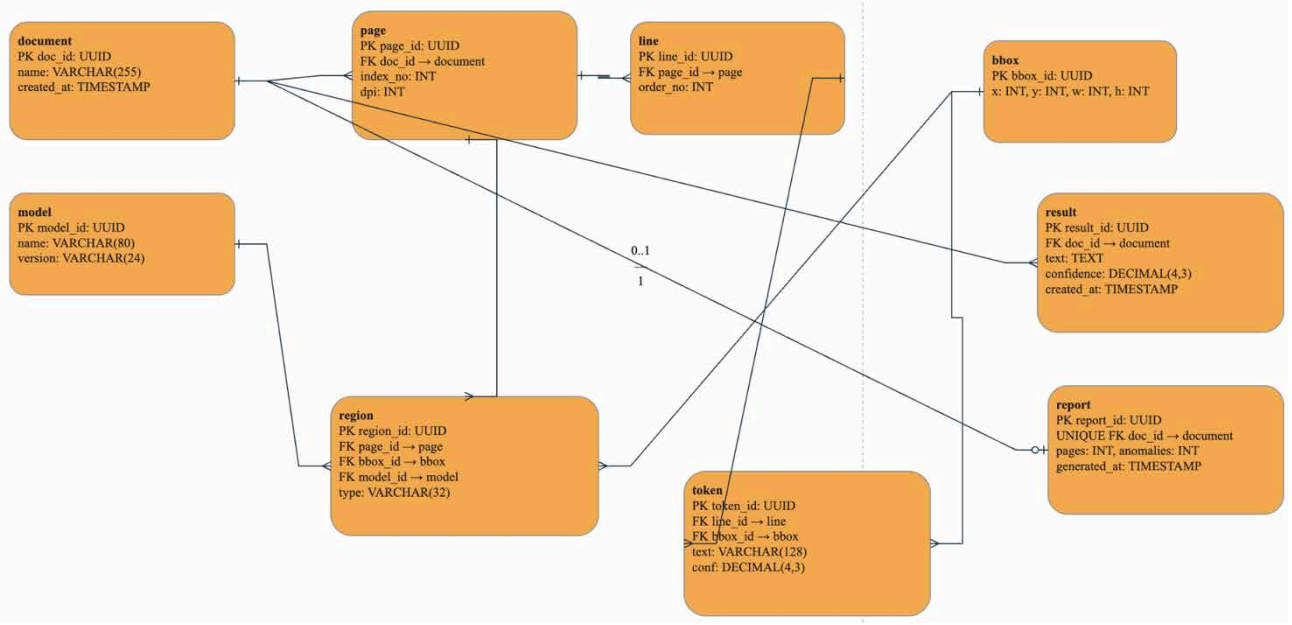

Рис. 2.1 – Логічна модель даних системи розпізнавання алфавітно-цифрової інформації

У центрі моделі розташована сутність document, яка виконує роль основного контейнера даних і зберігає метаінформацію про оброблювані документи. Кожен документ має кілька page, що описують сторінки з параметрами роздільної здатності та порядковими номерами. Кожна сторінка поділяється на line, які репрезентують текстові рядки, що в подальшому деталізуються у вигляді token - елементарних текстових одиниць із координатами прив’язки до області (bbox) та коефіцієнтом упевненості моделі. Сутність bbox формалізує геометричні межі фрагментів тексту, а region поєднує сторінку, модель і область, визначаючи контекст сегментації. Таблиця model зберігає параметри використаних нейронних мереж, що виконують розпізнавання символів, а сутність result акумулює підсумковий текст та його середній рівень достовірності. Завершує модель сутність report, у якій зберігається зведена статистика щодо документа - кількість сторінок, кількість аномалій і час формування звіту. Така структуризація забезпечує ієрархічну цілісність даних, підтримує масштабованість, інтеграцію з REST-API та ефективну роботу із зовнішніми системами зберігання [10].

Для повнішого розуміння структури бази даних у табл. 2.1 наведено узагальнений опис сутностей, їхніх полів, типів даних і ключових зв’язків, що формують основу інформаційної моделі системи.

Таблиця 2.1 – Опис сутностей логічної моделі даних

№	Назва сутності	Поля та типи даних	Ключі	Призначення
1	document	doc_id: UUID, name: VARCHAR(255), created_at: TIMESTAMP	PK(doc_id)	Зберігає метадані документа, що обробляється системою
2	page	page_id: UUID, doc_id: UUID, index_no: INT, dpi: INT	PK(page_id), FK(doc_id → document)	Містить параметри сторінки документа, зокрема роздільну здатність
3	line	line_id: UUID, page_id: UUID, order_no: INT	PK(line_id), FK(page_id → page)	Представляє логічні рядки тексту, отримані після сегментації
4	bbox	bbox_id: UUID, x: INT, y: INT, w: INT, h: INT	PK(bbox_id)	Визначає координати області тексту (bounding box) на зображенні
5	token	token_id: UUID, line_id: UUID, bbox_id: UUID, text: VARCHAR(128), conf: DECIMAL(4,3)	PK(token_id), FK(line_id → line), FK(bbox_id → bbox)	Зберігає розпізнані текстові одиниці (слова або символи) та рівень упевненості
6	model	model_id: UUID, name: VARCHAR(80), version: VARCHAR(24)	PK(model_id)	Містить ідентифікаційну інформацію про нейронну модель розпізнавання
7	region	region_id: UUID, page_id: UUID, bbox_id: UUID, model_id: UUID, type: VARCHAR(32)	PK(region_id), FK(page_id → page), FK(bbox_id → bbox), FK(model_id → model)	Поєднує модель, сторінку та область розпізнавання в єдину аналітичну одиницю
8	result	result_id: UUID, doc_id: UUID, text: TEXT, confidence: DECIMAL(4,3), created_at: TIMESTAMP	PK(result_id), FK(doc_id → document)	Накопичує фінальні результати OCR-процесу з оцінкою точності

Продовження таблиці 2.1

9	report	report_id: UUID, doc_id: UUID, pages: INT, anomalies: INT, generated_at: TIMESTAMP	PK(report_id), UNIQUE FK(doc_id → document)	Формує підсумкові аналітичні звіти для кожного документа

Запропонована логічна модель даних забезпечує узгоджене зберігання структурованої інформації, спрощує модифікацію модулів системи, дозволяє підключати нові алгоритми розпізнавання без зміни структури бази та забезпечує цілісність взаємодії між підсистемами клієнтського та серверного рівнів. Завдяки чітко визначеним зв'язкам між сутностями модель є придатною для масштабування, розподіленої обробки даних і подальшої оптимізації SQL-запитів у процесі функціонування системи [12].

2.2 Діаграма класів та кооперації

Діаграма класів відображає об'єктно-орієнтовану структуру програмного забезпечення та логіку взаємодії між компонентами системи. Вона визначає класи, їхні атрибути, методи, асоціації та інтерфейси, формуючи архітектурну основу програмної реалізації OCR-модуля. У контексті розроблюваної системи діаграма класів забезпечує формалізацію внутрішніх залежностей між підсистемами — препроцесингом, детекцією, розпізнаванням, постобробкою, збереженням результатів і генерацією звітів. Така структурна модель дозволяє на рівні проєктування визначити зв'язки між об'єктами, принципи інкапсуляції даних і забезпечити модульність програмної архітектури. На рис. 2.2 представлено UML-діаграму класів системи розпізнавання алфавітно-цифрової інформації, побудовану відповідно до специфікації об'єктів і механізмів їхньої взаємодії.

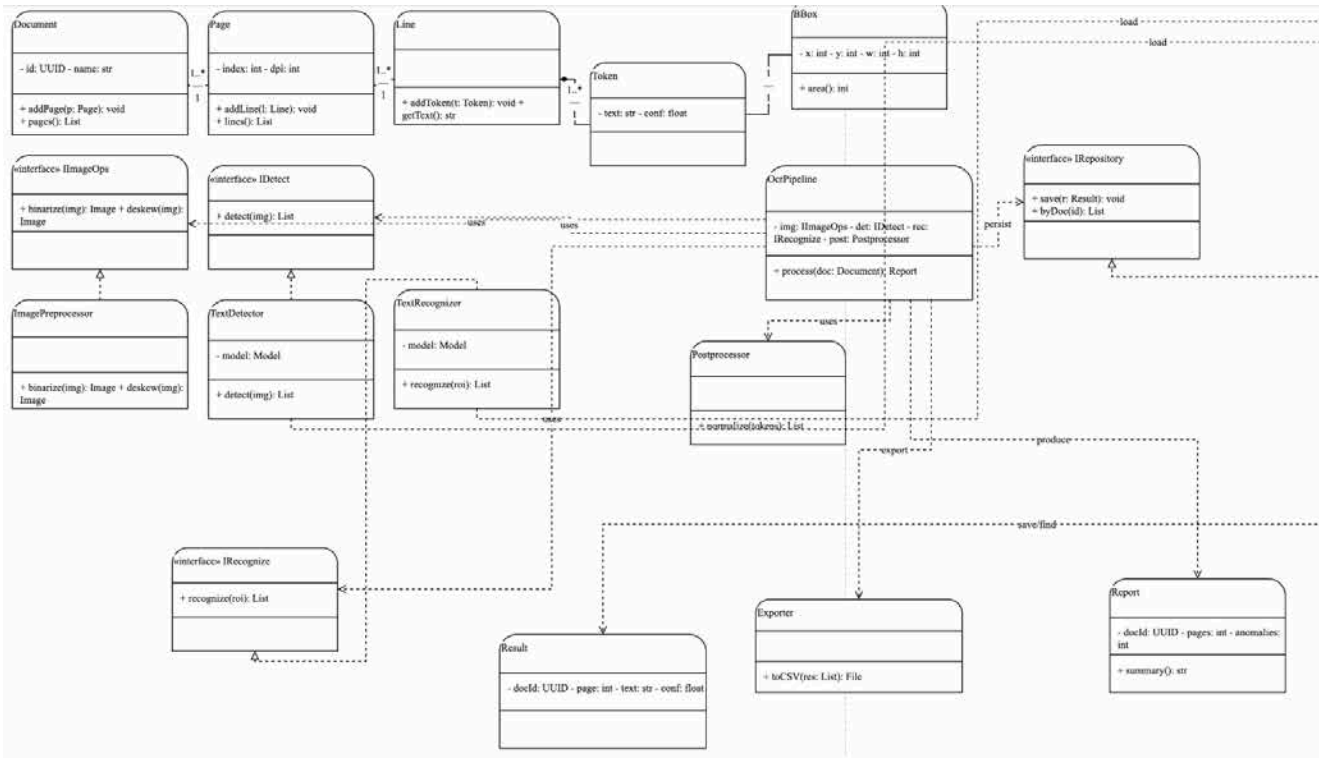

Рис. 2.2 – UML-діаграма класів системи розпізнавання алфавітно-цифрової інформації

У центрі архітектури розташовано клас OcrPipeline, який координує послідовність операцій під час обробки документа. Він використовує інтерфейси IImageOps, IDetect, IRecognize та клас Postprocessor, що забезпечують гнучкість заміни компонентів і розширюваність системи. Класи Document, Page, Line, BBox, Token, Result, Report утворюють доменну модель даних, яка відображає логічну структуру OCR-об’єктів. Допоміжні класи ImagePreprocessor, TextDetector, TextRecognizer реалізують конкретні алгоритмічні процедури: нормалізацію зображення, виявлення текстових областей і розпізнавання символів на основі моделі Model, що зберігає метадані нейромережевих ваг. Репозиторії IRepository та PgResultRepository відповідають за збереження та відновлення результатів, тоді як клас Exporter забезпечує експорт даних у зовнішні формати. Завдяки чіткому розділенню відповідальностей між класами досягається інкапсуляція функцій, повторне використання компонентів і висока тестованість системи [11].

Для підвищення формалізації архітектурної моделі в табл. 2.2 подано узагальнену характеристику основних класів системи із зазначенням їхніх атрибутів, методів і функціонального призначення.

Таблиця 2.2 – Опис основних класів UML-моделі

№	Клас	Основні атрибути	Методи	Призначення
1	Document	id: UUID, name: str	addPage(), pages()	Зберігає метадані документа та сторінкову структуру
2	Page	index: int, dpi: int	addLine(), lines()	Представляє сторінку документа з рядками тексту
3	Line	order: int	addToken(), getText()	Містить послідовність токенів, що формують рядок
4	Token	text: str, conf: float	—	Представляє одиницю розпізнавання (символ/слово) з упевненістю
5	BBox	x: int, y: int, w: int, h: int	area()	Визначає координати області тексту
6	Model	name: str, version: str	load()	Зберігає інформацію про нейромережеву модель
7	OcrPipeline	img: ImageOps, det: IDetect, rec: IRecognize, post: Postprocessor	process()	Керує основним циклом OCR-обробки документа
8	ImagePreprocessor	—	binarize(), deskew()	Виконує бінаризацію, нормалізацію та виправлення нахилу зображення
9	TextDetector	model: Model	detect()	Виконує пошук текстових областей на зображенні
10	TextRecognizer	model: Model	recognize()	Реалізує розпізнавання символів у межах області ROI

11	Postprocessor	—	normalize()	Проводить очищення, фільтрацію й валідацію розпізнаного тексту

Продовження таблиці 2.2

12	IRepository	—	save(), byDoc()	Інтерфейс доступу до сховища результатів
13	PgResultRepository	conn: Connection	save(), byDoc()	Реалізація репозиторію для PostgreSQL
14	Exporter	—	toCSV()	Формує звіт або експорт результатів у CSV-файл
15	Report	docId: UUID, pages: int, anomalies: int	summary()	Узагальнює результати обробки документа

Побудована структура демонструє чітке розділення ролей між об’єктами, підтримуючи принципи SOLID і сприяючи незалежності бізнес-логіки від технічної реалізації. Класи утворюють замкнутий цикл обробки, де OcrPipeline виконує центральну роль координатора, послідовно використовуючи препроцесор, детектор, розпізнавач, постпроцесор та модулі збереження результатів, що гарантує узгодженість і повторюваність процесу.

Асоціативні зв’язки між класами визначають статичну структуру об’єктних відношень у системі, формуючи логічну основу для збереження, взаємодії та контролю узгодженості даних. Вони описують сталі залежності між об’єктами, що забезпечують цілісність моделі та правильність передачі інформації між рівнями архітектури. На рис. 2.3 подано асоціацію між класами Document і Result, яка реалізує відношення типу «один-до-багатьох». Це означає, що один документ може містити множину результатів розпізнавання, отриманих у різних сесіях або за різних налаштувань моделей.

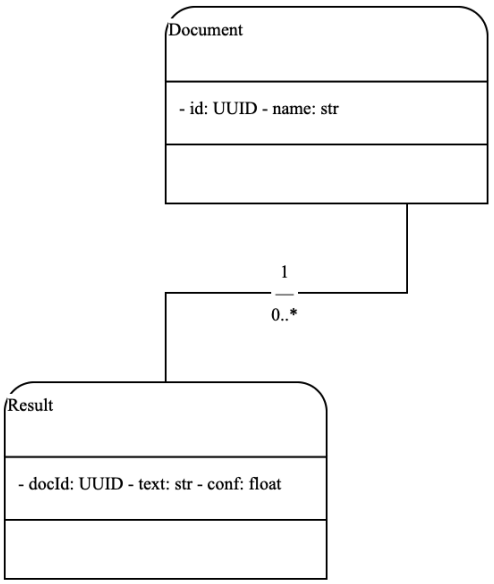

Рис. 2.3 – Асоціація класів Document ↔ Result

Зв’язок Document–Result забезпечує можливість зберігання історії розпізнавань для кожного документа, підтримуючи повторну обробку та накопичення статистичних даних. Кожен об’єкт Result містить текст розпізнаного фрагмента та коефіцієнт упевненості, що дозволяє системі виконувати аналіз ефективності моделей і реалізовувати контроль якості результатів.

На рис. 2.4 наведено асоціацію між класами Document і Report**,** яка описує відношення типу «один-до-одного або відсутності» (1↔0..1). Це означає, що кожен документ може мати лише один звіт, який агрегує результати OCR-обробки, або не мати його, якщо розпізнавання ще не завершено.

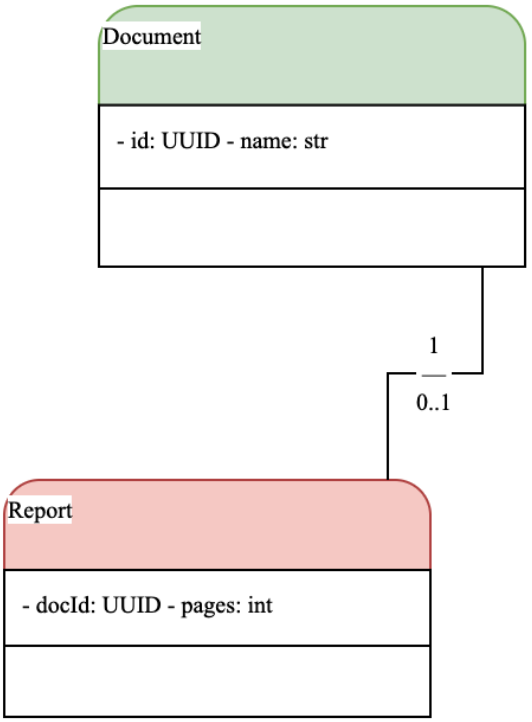

Рис. 2.4 – Асоціація класів Document ↔ Report

Зв’язок Document–Report визначає аналітичну функцію системи, оскільки звіт узагальнює результати обробки, кількість сторінок, виявлені аномалії та часові характеристики виконання. Така асоціація є важливою для формування фінальних аналітичних звітів, автоматичного моніторингу продуктивності моделей і контролю достовірності результатів розпізнавання.

Третя асоціація, подана на рис. 2.5, відображає відношення між класами Token і BBox, що має тип «один-до-одного». Кожен токен (розпізнаний символ або слово) має унікальну просторову прив’язку до області зображення, представленої об’єктом BBox, який визначає координати x, y, ширину та висоту області.

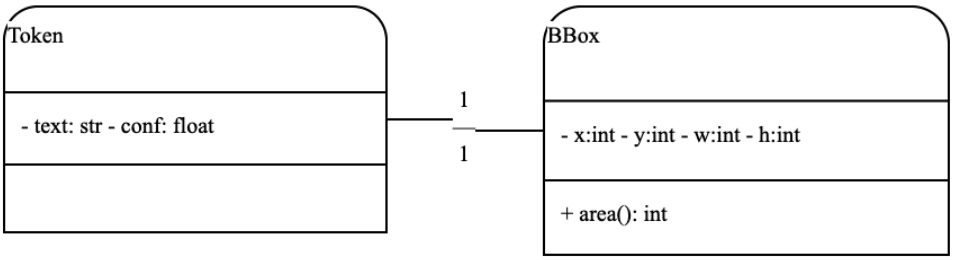

Рис. 2.5 – Асоціація класів Token ↔ BBox

Дана залежність забезпечує точну просторову відповідність між текстовими елементами та їхніми графічними проєкціями, що дозволяє реалізувати візуалізацію результатів OCR, підсвічування символів у графічному інтерфейсі та перевірку геометричної коректності. Вона також є ключовою для модулів постобробки, де координати bounding box використовуються для валідації позицій символів і адаптації до нестандартних форматів документа.

Асоціації Document–Result**,** Document–Report та Token–BBox формують логічну основу статичної структури системи, забезпечуючи узгодженість даних, підтримку повторного використання об’єктів і можливість інтеграції аналітичних модулів. Така модель зберігає баланс між гнучкістю архітектури й суворістю структурної дисципліни, що сприяє масштабуванню системи, спрощує її супровід і розширення функціональності в майбутніх версіях [14].

2.3 Діаграма компонентів

Діаграма компонентів відображає архітектурну структуру програмного забезпечення та демонструє логічну декомпозицію системи на окремі модулі, що взаємодіють між собою через стандартизовані інтерфейси. Вона є ключовим елементом архітектурного проєктування, оскільки визначає межі відповідальності компонентів, способи їхньої комунікації та напрямки потоків даних. Для системи розпізнавання алфавітно-цифрової інформації компонентна модель побудована за принципами модульності, слабкого зв’язку та відкритої інтеграції, що забезпечує гнучкість, масштабованість і сумісність із зовнішніми сервісами. На рис. 2.3 подано UML-діаграму компонентів OCR-системи, структуровану за чотирирівневою архітектурною моделлю: Presentation, Application, Domain та Infrastructure.

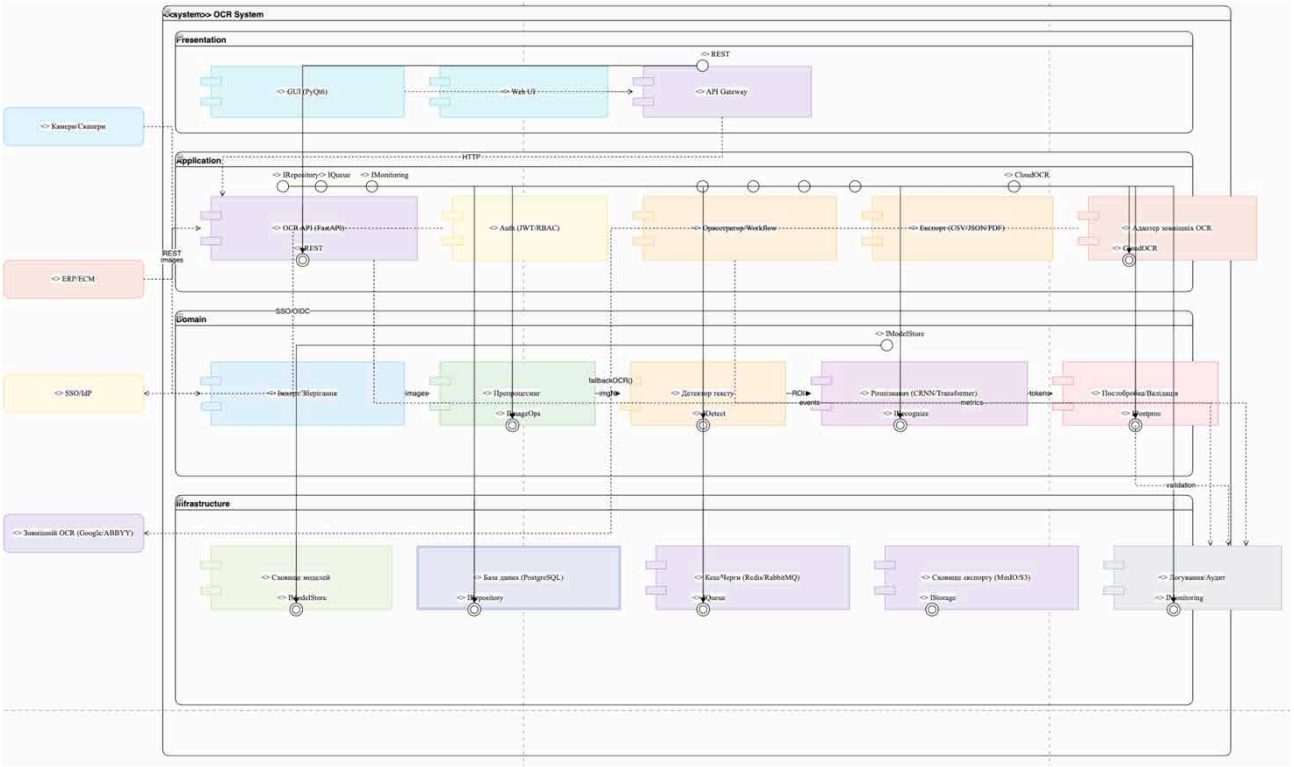

Рис. 2.3 – Діаграма компонентів системи розпізнавання алфавітно-цифрової інформації

На рівні Presentation реалізовано компоненти графічного та вебінтерфейсу - GUI (PyQt6) і Web-UI, що забезпечують взаємодію користувача із системою через REST-шлюз (API Gateway). Рівень Application містить керуючі модулі, серед яких OCR API (FastAPI), Auth (JWT/RBAC), Processor Workflow та Exporter, що реалізують бізнес-логіку, аутентифікацію користувачів, керування потоками задач і формування результатів у форматах CSV, JSON або PDF. Взаємодія з корпоративними сервісами (ERP/ECM) і зовнішніми OCR-платформами (Google Cloud OCR, ABBYY) відбувається через REST-інтерфейси.

Рівень Domain представлений основними функціональними модулями: Препроцесинг (ImageOps), Детектор тексту (Detect), Розпізнавач (Recognize) та Постобробка (Postprocess), що реалізують послідовність обробки зображення — від фільтрації шумів до валідації та нормалізації розпізнаного тексту. Доменна частина також включає компоненти Імпорт

зображень, Інтерфейс SSO/OIDC, ModelStore для завантаження моделей та Monitoring для фіксації метрик продуктивності.

На рівні Infrastructure розміщено системні ресурси: База даних (PostgreSQL), Кеш/Черга (Redis/RabbitMQ), Сховище моделей, Сховище експорту (MinIO/S3) та Моніторинг/Аудит, які забезпечують стабільність і надійність функціонування системи. Інфраструктурний рівень виконує роль базового середовища, що підтримує транзакційність, обмін даними, резервування, журналювання та контейнеризацію.

У табл. 2.3 наведено узагальнений опис компонентів системи, їх функціональних ролей і взаємозв'язків між архітектурними рівнями.

Таблиця 2.3 – Опис основних компонентів системи OCR

№	Компонент	Архітектурний рівень	Основне призначення	Інтерфейси взаємодії
1	GUI (PyQt6)	Presentation	Клієнтський графічний інтерфейс для локальної взаємодії користувача	HTTP / REST
2	Web-UI	Presentation	Вебінтерфейс для віддаленого доступу до OCR-сервісу	REST / API Gateway
3	Auth (JWT/RBAC)	Application	Аутентифікація користувачів і контроль доступу	OAuth2 / SSO
4	OCR API (FastAPI)	Application	Центральний сервіс обробки запитів і керування потоками обробки	REST / JSON
5	Processor Workflow	Application	Оркестрація задач препроцесингу, розпізнавання та постобробки	Internal API
6	Exporter (CSV/JSON/PDF)	Application	Експорт результатів OCR у зовнішні формати	File / HTTP
7	ImageOps (Препроцесинг)	Domain	Попередня обробка зображень (нормалізація, фільтрація, deskew)	Internal API

Продовження таблиці 2.3

8	Detect (Детектор тексту)	Domain	Виявлення текстових областей (ROI) на зображеннях	Internal API
9	Recognize (CRNN/Transformer)	Domain	Розпізнавання символів і формування токенів тексту	Internal API
10	Postprocess (Валідація)	Domain	Нормалізація, перевірка форматів і фільтрація результатів	Validation API
11	ModelStore	Infrastructure	Зберігання навчальних моделей нейронних мереж	Local FS / Cloud
12	Repository (PostgreSQL)	Infrastructure	Центральне сховище даних і метаданих розпізнавання	SQL / ORM
13	Queue (Redis/RabbitMQ)	Infrastructure	Керування асинхронними задачами та чергами повідомлень	AMQP
14	Storage (MinIO/S3)	Infrastructure	Сховище зображень і результатів експорту	S3 API
15	Monitoring/Audit	Infrastructure	Збір телеметрії, логів, звітів і станів системи	Prometheus / Grafana

Узагальнюючи, компонентна структура системи забезпечує чітке розділення відповідальності між рівнями та спрощує подальшу підтримку й масштабування рішення. Завдяки впровадженню стандартних інтерфейсів REST, ORM і AMQP система може легко інтегруватися з корпоративними сервісами (ERP/ECM, SSO/IdP) та зовнішніми OCR-платформами (Google, ABBYY). Така архітектура підвищує надійність, гнучкість і стійкість системи до навантажень, забезпечуючи можливість розгортання у контейнеризованому середовищі (Docker, Kubernetes) та розподіленої обробки зображень у хмарній інфраструктурі [9].

2.4 Діаграма пакетів

Діаграма пакетів відображає логічну структуру системи на рівні організації модулів, показуючи ієрархічні залежності, взаємозв’язки та напрямки імпорту між окремими частинами програмного забезпечення. Її

основна мета — забезпечити структурованість архітектури, спростити підтримку коду, ізолювати рівні відповідальності та створити передумови для масштабованості системи. У контексті OCR-платформи діаграма пакетів демонструє розподіл системи за доменними, інфраструктурними, прикладними та адаптерними рівнями, кожен з яких реалізує окремий набір функцій. На рис. 2.4 представлено узагальнену UML-діаграму пакетів системи розпізнавання алфавітно-цифрової інформації, побудовану за принципами архітектури Hexagonal (Ports and Adapters).

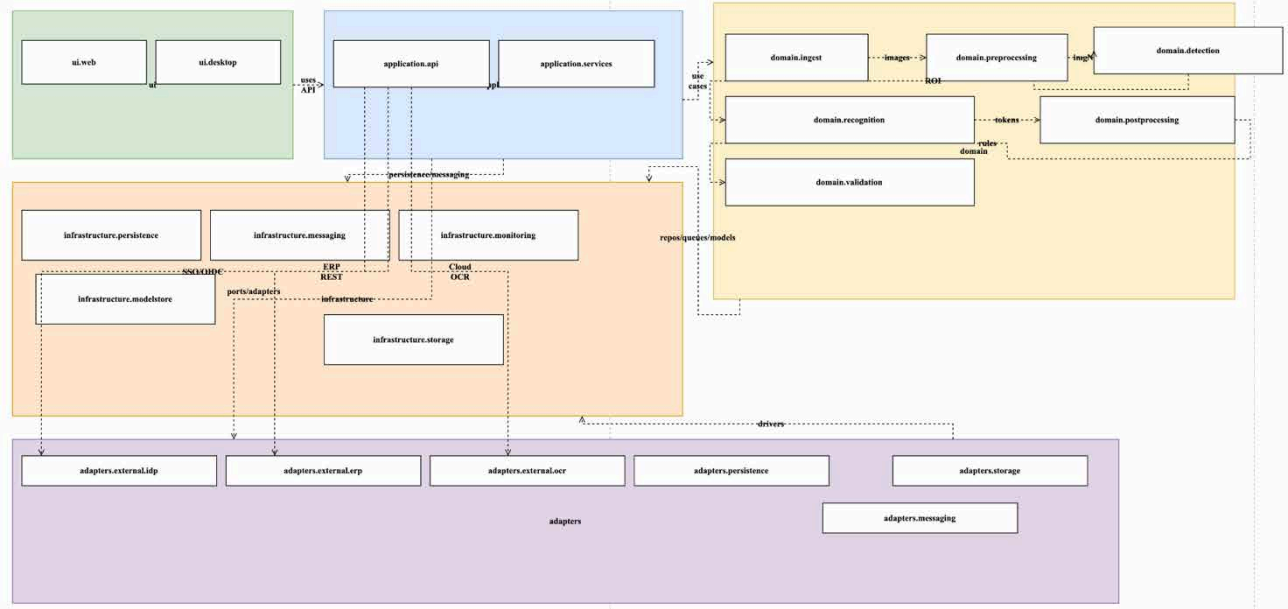

Рис. 2.4 – Діаграма пакетів системи розпізнавання алфавітно-цифрової інформації

Архітектура складається з чотирьох основних шарів: UI, Application, Domain, Infrastructure та Adapters. Пакети рівня UI містять підсистеми ui.web і ui.desktop, які реалізують веб- та настільний інтерфейси користувача. Вони взаємодіють з прикладним рівнем через API, що забезпечує відокремлення користувацької логіки від бізнес-логіки. Рівень Application включає пакети application.api та application.services, які координують сценарії використання системи, ініціюють процеси OCR-обробки та забезпечують послідовність викликів компонентів доменного рівня.

Пакети рівня Domain визначають ключові бізнес-модулі, серед яких domain.ingest, domain.preprocessing, domain.detection, domain.recognition, do main.postprocessing і domain.validation. Вони відповідають за основний цикл розпізнавання: від імпорту зображень і виділення текстових областей до генерації токенів, валідації та нормалізації результатів. Взаємодія між пакетами відбувається через узгоджені об’єкти (ROI, tokens, rules), що забезпечує узгодженість і повторюваність обробки даних.

Інфраструктурний рівень (Infrastructure) об’єднує пакети infrastructure.persistence, infrastructure.messaging, infrastructure.modelstore, infrastructure.monitoring та infrastructure.storage, які реалізують технічні сервіси - роботу з базою даних, чергами повідомлень, зберіганням моделей і контроль стану системи. Вони надають стандартизовані інтерфейси для доменного рівня через шаблони ports/adapters, забезпечуючи низьку зв’язність та незалежність бізнес-логіки від конкретних технологій.

Рівень Adapters інтегрує систему з зовнішніми середовищами, зокрема корпоративними платформами та хмарними OCR-сервісами. Пакети adapters.external.idp, adapters.external.erp, adapters.external.ocr, adapters.pe rsistence, adapters.storage та adapters.messaging реалізують драйвери для взаємодії з сервісами SSO/IdP, ERP/ECM, зовнішніми OCR-API, чергами повідомлень та файловими сховищами. Така структура дає змогу гнучко адаптувати систему під різні інфраструктурні середовища (локальні або хмарні) без зміни внутрішньої логіки [1].

Таблиця 2.4 – Опис основних пакетів системи OCR

№	Пакет	Рівень архітектури	Призначення	Ключові залежності
1	ui.web, ui.desktop	Presentation (UI)	Графічний та вебінтерфейс користувача	Application API
2	application.api, application.services	Application	Реалізація бізнес-сценаріїв, обробка запитів,	Domain, Infrastructure

			координація потоків даних	

Продовження таблиці 2.4

3	domain.ingest, domain.preprocessing, domain.detection	Domain	Імпорт, обробка та виявлення текстових областей	Internal domain modules
4	domain.recognition, domain.postprocessing, domain.validation	Domain	Розпізнавання, нормалізація та перевірка результатів OCR	Infrastructure, Rules
5	infrastructure.persistence	Infrastructure	Збереження даних у базі (PostgreSQL, ORM)	Adapters.persistence
6	infrastructure.messaging	Infrastructure	Черги повідомлень, асинхронна взаємодія (Redis/RabbitMQ)	Adapters.messaging
7	infrastructure.modelstore	Infrastructure	Зберігання та завантаження нейромережевих моделей	Adapters.storage
8	infrastructure.monitoring	Infrastructure	Моніторинг, логування, аудит системи	External monitoring
9	adapters.external.idp, adapters.external.erp, adapters.external.ocr	Adapters	Інтеграція із зовнішніми сервісами (SSO, ERP, Cloud OCR)	REST / SSO / OIDC
10	adapters.storage, adapters.messaging, adapters.persistence	Adapters	Робота з файловими сховищами, чергами та БД	Infrastructure modules

Узагальнюючи, представлена структура пакетів демонструє багаторівневу організацію програмного комплексу, яка поєднує переваги DDD-підходу (Domain-Driven Design) та гексагональної архітектури. Такий підхід забезпечує слабку зв’язність між модулями, високу модульність, можливість автономного

тестування та розширення системи. У результаті реалізовано архітектуру, придатну до інтеграції в корпоративні екосистеми, здатну підтримувати розподілену обробку даних і гнучке масштабування на рівні окремих сервісів [12].

2.5 Висновки до другого розділу

У другому розділі було здійснено проєктування інформаційного та програмного забезпечення системи розпізнавання алфавітно-цифрової інформації, що забезпечило формування цілісної архітектурної моделі OCR-рішення. На основі вимог першого розділу побудовано логічну модель даних у вигляді ER-діаграми, яка формалізує структуру сховища документів, сторінок, рядків, токенів, областей розпізнавання, моделей та звітів. Запропонована схема відповідає принципам нормалізації, забезпечує відсутність дублювання, підтримує масштабованість та дозволяє зберігати повну історію обробки документів і метрики точності.

Розроблена UML-діаграма класів і кооперацій відображає об’єктно-орієнтовану структуру програмного забезпечення, включно з основними доменними сутностями, сервісами препроцесингу, детекції, розпізнавання та постобробки, а також репозиторіями збереження результатів. Чітке розділення відповідальності між класами забезпечує модульність, інкапсуляцію та можливість подальшого розширення системи без порушення існуючої логіки.

Побудована діаграма компонентів описує архітектуру за рівнями Presentation, Application, Domain та Infrastructure, демонструючи взаємодію клієнтського інтерфейсу, REST-API, нейромережевих модулів і бази даних. Такий підхід підтверджує здатність системи до інтеграції з ERP/ECM-платформами, зовнішніми OCR-сервісами та інфраструктурними інструментами (Redis, Docker, Monitoring). Діаграма пакетів завершила архітектурне проєктування, формалізувавши структуризацію програмного коду за принципами гексагональної архітектури та DDD, що сприяє слабкій

зв’язності, тестованості та незалежності бізнес-логіки від конкретних технологічних рішень.

Узагальнюючи, у другому розділі сформовано повну архітектурну специфікацію системи, що визначає структуру даних, взаємодію програмних компонентів, організацію коду та технічні межі майбутньої реалізації. Результати розділу забезпечують методологічну основу для переходу до наступного етапу - програмної реалізації компонентів OCR-модуля та інтеграції їх у єдине програмне рішення.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ТЕХНОЛОГІЧНА ІНФРАСТРУКТУРА СИСТЕМИ

3.1 Вибір технологій та інструментальних засобів реалізації системи

Вибір технологій для розроблення програмного забезпечення розпізнавання алфавітно-цифрової інформації визначався вимогами до точності, продуктивності, масштабованості та інтегрованості системи у сучасні інформаційні середовища. Оскільки OCR-процес включає етапи препроцесингу, детекції текстових областей, класифікації символів, валідації та експорту результатів, було обрано стек технологій, який оптимально поєднує можливості глибинного навчання, високопродуктивної обробки зображень та сервісної архітектури.

Основною мовою програмування обрано Python, що зумовлено його домінуванням у сфері комп'ютерного зору та підтримкою широкого спектра бібліотек для нейромережевих моделей. Для серверної частини застосовано FastAPI, який забезпечує асинхронну обробку запитів, автоматичне формування OpenAPI-специфікацій та високу продуктивність у REST-архітектурі. У частині препроцесингу використано OpenCV, що надає стабільні та оптимізовані інструменти для нормалізації зображень, бінаризації, фільтрації та корекції геометричних спотворень. Для побудови моделей розпізнавання символів та реалізації CNN/LSTM/Transformer-архітектур застосовано PyTorch - фреймворк із гнучким механізмом навчання, підтримкою GPU та кастомізації моделей під вузькі прикладні задачі.

Для локального графічного інтерфейсу обрано PyQt6, який дозволяє створити кросплатформену настільну програму з інтегрованими засобами відображення зображень, підсвічуванням розпізнаних областей та функціями ручного редагування. Збереження результатів та метаданих здійснюється у PostgreSQL, що забезпечує транзакційність, підтримку складних зв'язків та можливість масштабування у корпоративному середовищі. Кешування та

організацію асинхронних черг забезпечує Redis, а контейнеризація через Docker гарантує портативність і стабільне розгортання системи незалежно від цільової платформи.

У табл. 3.1 наведено обґрунтований вибір основних технологій та їх роль у реалізації системи.

Таблиця 3.1 – Технології та інструменти, використані у реалізації OCR-системи

№	Технологія / засіб	Основне призначення в системі
1	Python 3.10+	Базова мова реалізації логіки OCR-процесу
2	OpenCV	Препроцесинг зображень: фільтрація, бінаризація, нормалізація
3	PyTorch	Побудова та використання нейронних моделей для розпізнавання
4	FastAPI	REST-сервіс OCR, керування запитами, інтеграція з клієнтами
5	PyQt6	Настільний GUI для взаємодії з результатами розпізнавання
6	PostgreSQL	Сховище документів, токенів, моделей, результатів і звітів
7	Redis	Черги завдань, кешування проміжних даних
8	Docker / Docker Compose	Контейнеризація, ізоляція сервісів і прискорення розгортання
9	Uvicorn/Gunicorn	Продуктивний ASGI-сервер для FastAPI
10	Pytest	Модульні та інтеграційні тести компонентів системи
11	MinIO/S3 (опційно)	Зберігання зображень та експортованих результатів

Обраний технологічний стек створює узгоджене середовище для розроблення продуктивної, адаптивної та масштабованої OCR-системи. Python та PyTorch забезпечують високоточне розпізнавання символів, OpenCV - якісний препроцесинг, FastAPI - швидку та стандартизовану взаємодію з клієнтами, а PostgreSQL та Redis - надійність і керованість даними у середовищі реального використання. Такий вибір дозволяє створити архітектурно стійке рішення та забезпечити подальше розширення функціональності системи без необхідності зміни базових інструментів.

3.2 Інформаційна база системи

Інформаційна база розроблюваної системи розпізнавання алфавітно-цифрової інформації являє собою цілісну багаторівневу структуру, що поєднує первинні дані (зображення та їх метадані), проміжні результати препроцесингу, дані нейромережевих моделей, OLAP-факти, аналітичні зрізи та агреговані кластерні оцінки. Така структура забезпечує не лише коректність і повноту обробки, а й створює науково обґрунтовану основу для проведення високоточних аналітичних операцій, включно з багатовимірним аналізом продуктивності моделей та адаптивною оптимізацією параметрів розпізнавання.

Загальна схема інформаційної бази формується навколо факт-таблиці сесій розпізнавання, яка інтегрується з вимірними таблицями за документом, часом, моделлю, джерелом та типом помилки. На рисунку 3.1 наведено UML-подання OLAP-куба, що демонструє структуру зіркової схеми, яка використовується для побудови багатовимірної аналітики.

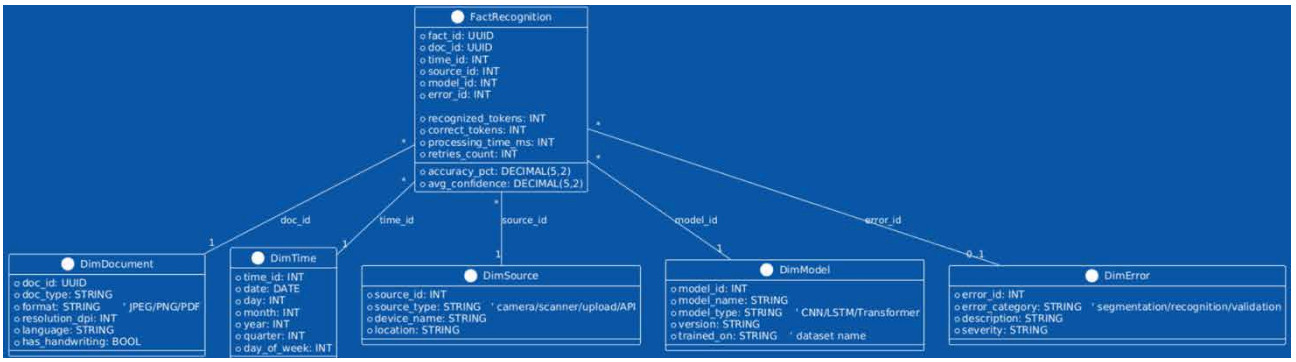

Рис. 3.1 – UML-схема OLAP-кубу системи розпізнавання алфавітно-цифрової інформації

Інформаційна база розширюється підсистемою аналітичного моделювання, у рамках якої зберігаються нормалізовані векторні представлення сесій розпізнавання, обчислені головні компоненти та результати кластерного аналізу. Це дозволяє системі не лише аналізувати показники продуктивності, а й виявляти закономірності у роботі моделей, групувати схожі сесії та оцінювати ризикові випадки. На рис. 3.2 наведено візуалізацію кластеризації сесій розпізнавання методом PCA·k-means (k=3), яка відображає результат обробки

даних OLAP-кубу та демонструє здатність системи до автоматизованого виокремлення стабільних, уповільнених/шумних та ризикових сценаріїв.

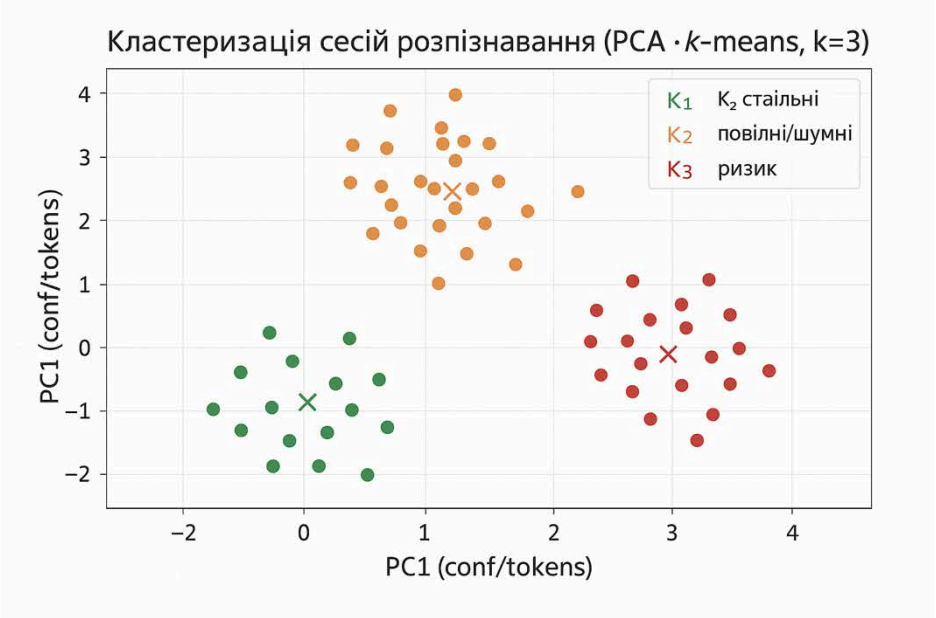

Рис. 3.2 – Кластеризація сесій розпізнавання (PCA·k-means, k=3)

Для визначення оптимальної кількості кластерів використано метод ліктя, що дозволяє ідентифікувати переломний момент суттєвого зниження сумарної внутрішньокластерної квадратичної помилки (SSE). На рис. 3.3 подано відповідну діаграму, яка демонструє, що оптимальним значенням є k = 3, що також збігається з реальними закономірностями групування за якістю та стабільністю результатів розпізнавання.

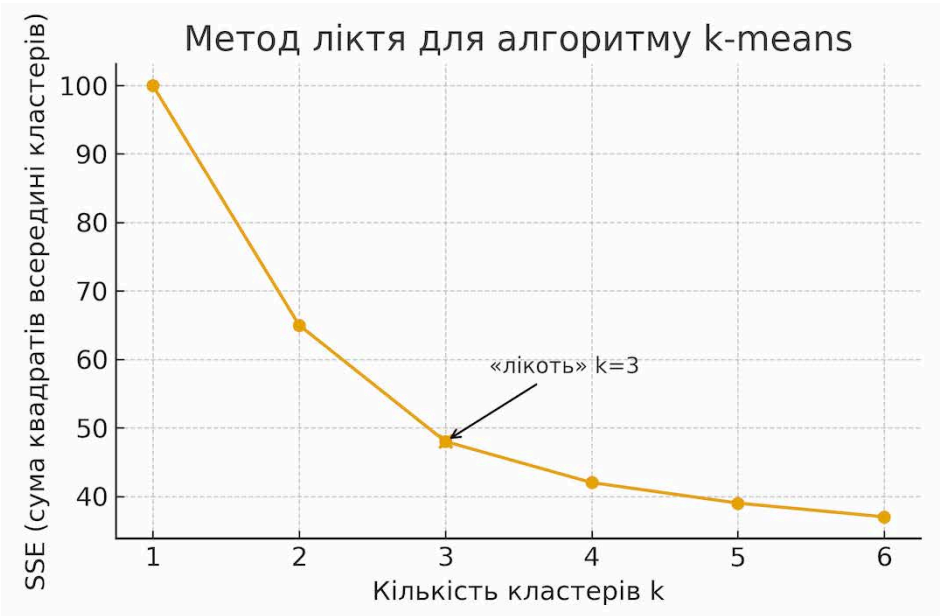

Рис. 3.3 – Метод ліктя для вибору параметра k у k-means

На наступному рівні інформаційної бази формується радіальна діаграма якості кластерів, що відображає порівняння груп за точністю, повнотою, стабільністю довірчих оцінок, швидкодією та надійністю. Така аналітична модель є критично важливою для прийняття рішень щодо перемикання моделей, адаптації параметрів препроцесингу та виявлення слабких місць у роботі всього OCR-конвеєра. На рис. 3.4 наведено відповідне порівняння.

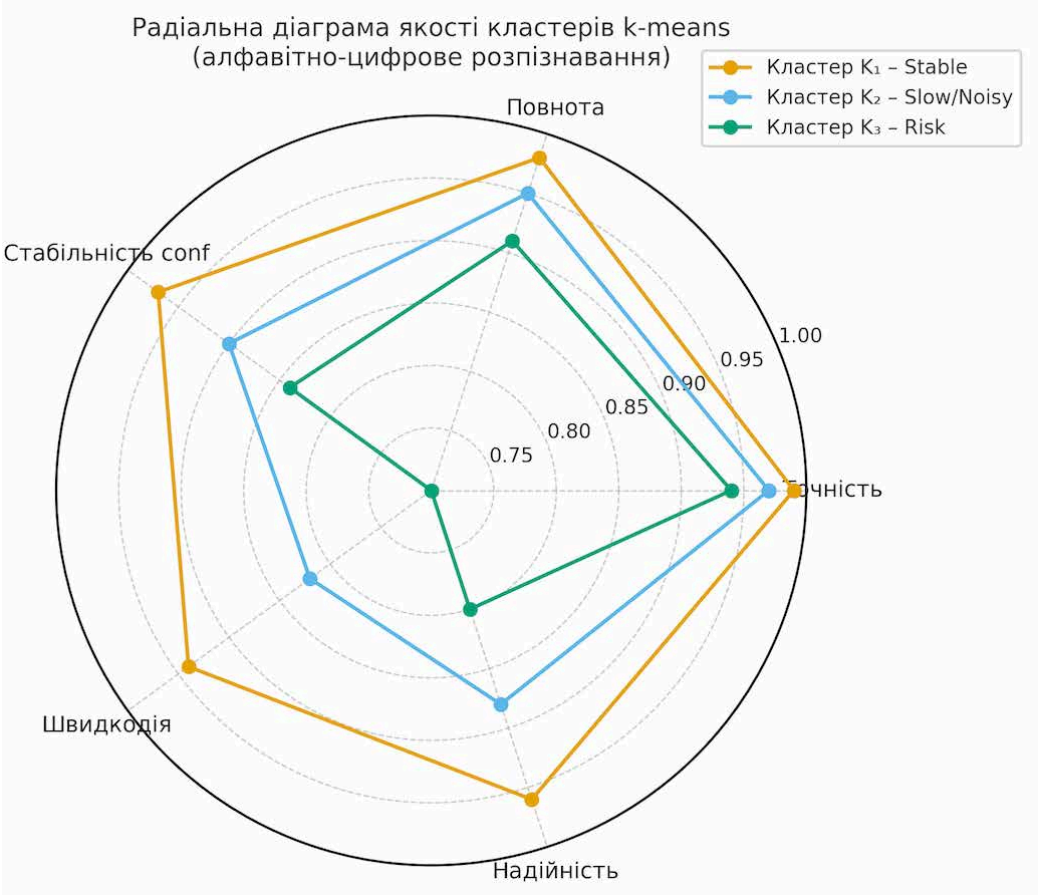

Рис. 3.4 – Радіальна діаграма якості кластерів OCR-сесій (k-means)

Узагальнена структура інформаційної бази відображає новизну підходу, реалізовану в роботі, а саме:

Таблиця 3.2 – Ключові аспекти наукової та технічної новизни інформаційної бази системи

№	Аспект новизни	Технічний зміст та значення
1	Інтеграція OLAP-моделі в OCR-систему	Створює багатовимірну систему аналізу продуктивності, недоступну в класичних OCR-рішеннях

2	Факт-таблиця з підтримкою помилок	Дозволяє оцінювати не лише точність, а й структурну природу помилок (segmentation/recognition/validation)

Продовження таблиці 3.2

3	PCA-представлення сесій розпізнавання	Формує компактний векторний простір для подальшої кластеризації та багатовимірного аналізу
4	Аналітичні кластери як елемент адаптивності	Система може динамічно обирати модель або параметри препроцесингу залежно від кластерної групи
5	Комбінована база первинних і аналітичних даних	Забезпечує наскрізний шлях від зображення до аналітичних висновків, з повною трасованістю
6	Підтримка гібридної OLTP+OLAP структури	Дає змогу одночасно здійснювати розпізнавання та будувати аналітичні зрізи без деградації продуктивності

Інформаційна база системи поєднує класичні структури індексованих OCR-даних із сучасними засобами аналітичної інтелектуальної обробки, що забезпечує глибоку інтерпретованість результатів, можливість прогнозування ефективності та підтримку адаптивної оптимізації параметрів моделі. Це створює науково обґрунтовану основу для інтелектуального керування якістю розпізнавання та формує конкурентну перевагу запропонованого програмного рішення.

3.3 Архітектура системи та проєктування функціоналу результатів дослідження

Архітектура розроблюваної системи формує цілісну багато-рівневу модель обробки, аналізу та зберігання даних, забезпечуючи наскрізний цикл проходження інформації: від надходження зображення → до препроцесингу → ML-модулів розпізнавання → аналітичного ядра OLAP → формування кластерних та агрегованих метрик → повернення результатів користувачеві. Такий підхід створює умови для високої точності, адаптивності та масштабованості системи, а також дозволяє реалізувати функціональну новизну

дослідження - інтеграцію OLAP-аналізу, методу ліктя та k-means у контур OCR-модуля.

На рис. 3.5 наведено базову структурну схему архітектури системи, яка відображає основні інформаційні потоки між клієнтським застосунком, модулем попередньої обробки зображень, ML-модулем розпізнавання та аналітичною підсистемою.

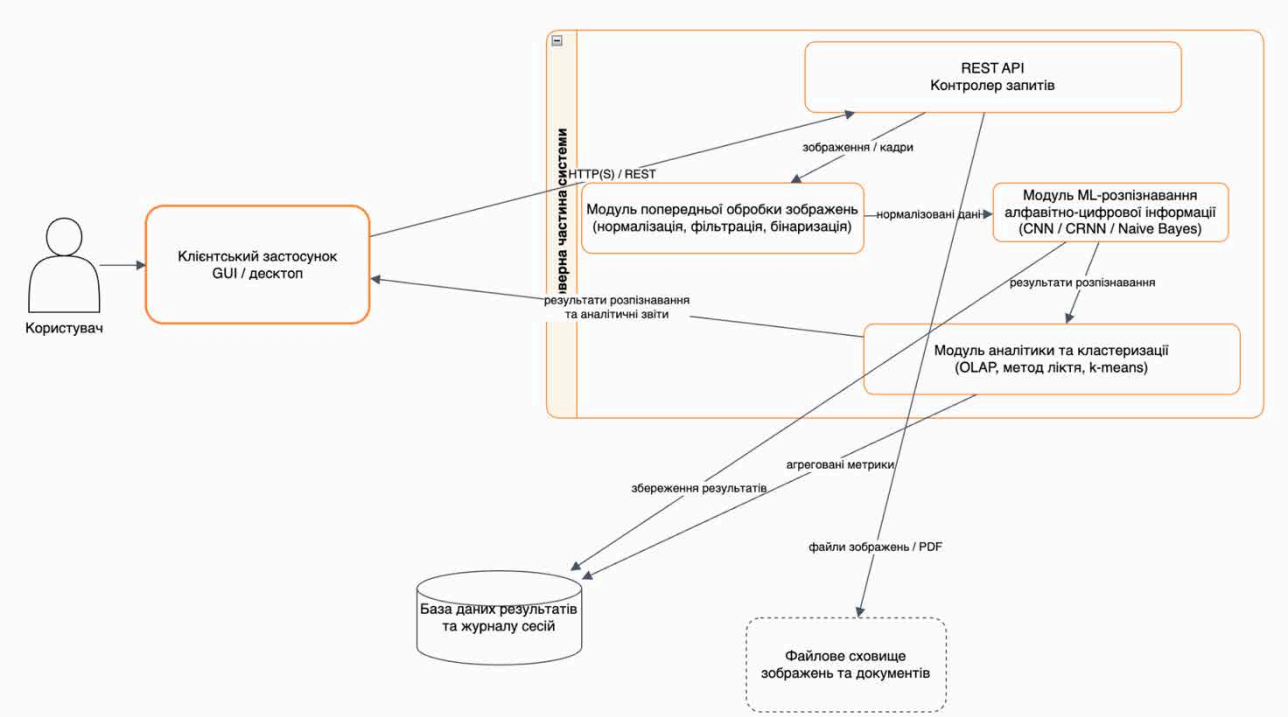

Рис. 3.5 – Структурна схема архітектури модуля розпізнавання та аналітики системи

Як показано на рис. 3.5, клієнтський застосунок передає зображення через REST-інтерфейс до серверної частини, де відбувається їх нормалізація (масштабування, фільтрація шумів, бінаризація), після чого дані надходять до ML-модуля. Після отримання результатів розпізнавання вони фіксуються у базі даних результатів та спрямовуються до аналітичної підсистеми, де здійснюється побудова кластерів, агрегування метрик, формування OLAP-куба та статистичних звітів. Повернуті GUI-програмі дані дозволяють користувачу переглядати токени, впевненість моделей, джерело помилок, аналітичні діаграми та рекомендації.

Для деталізації механізмів взаємодії між сервісами розроблено діаграму розгортання, подану на рис. 3.6, яка демонструє логічний розподіл компонентів системи у реальному середовищі: клієнтський пристрій, DMZ-рівень з реверс-проксі, сервер прикладного рівня, ML-сервіси, брокер подій, БД та файлове сховище.

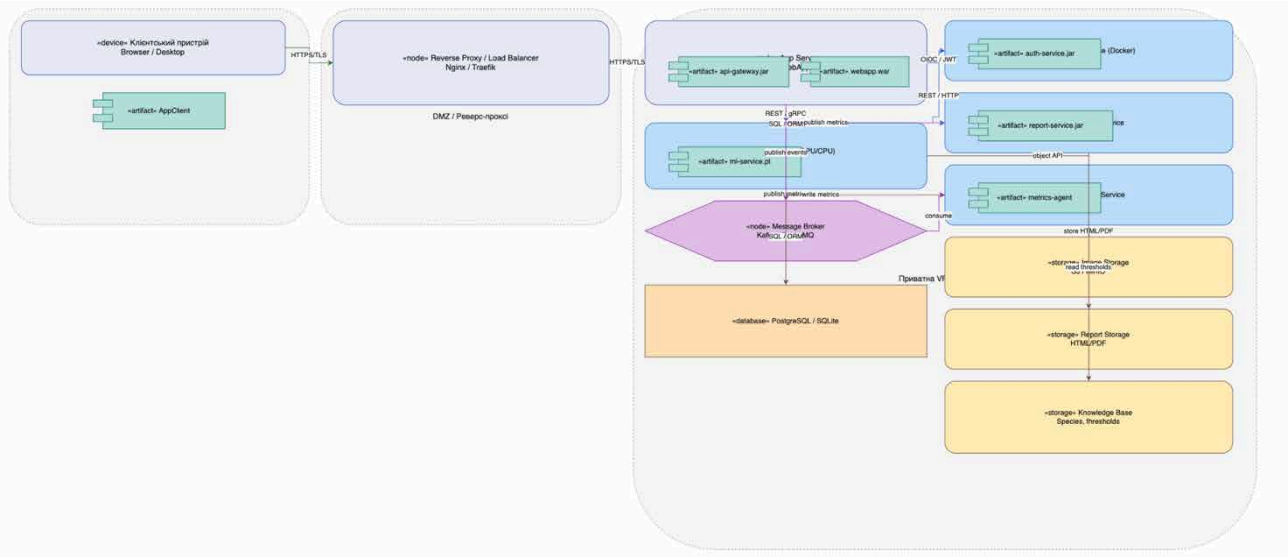

Рис. 3.6 – Діаграма розгортання компонентів системи у виробничому середовищі

На рис. 3.6 показано, що всі сервіси системи ізольовано у контейнеризованому середовищі (Docker), взаємодія клієнта з API захищена TLS, ML-модулі здійснюють публікацію метрик продуктивності до брокера подій, а звітний сервіс генерує HTML/PDF артефакти. База даних містить результати сесій розпізнавання, OLAP-факти, журнали кластеризації та розширену статистику. Сховище зображень містить вхідні файли, PDF-документи та службові артефакти. Така архітектура забезпечує масштабованість, ізоляцію сервісів, спрощену підтримку та можливість горизонтального розширення ML-компонентів.

З урахуванням результатів дослідження, у системі реалізовано функціональні новації, що дозволяють підвищити якість та прозорість роботи OCR-модуля:

1. інтеграція OLAP-куба - підтримка багатовимірного аналізу (за моделлю, часом, джерелом, типом документа, класом помилки).

2. Автоматичне визначення оптимальної кількості кластерів методом ліктя з подальшим формуванням кластерних груп.

3. Адаптивна оцінка продуктивності ML-моделей на основі кластерних метрик (повнота, точність, стабільність conf).

4. Повна трасованість сесії розпізнавання - від зображення до аналітичного звіту.

5. Зберігання аналітичних агрегатів (часові зрізи, показники швидкодії, розподіл помилок, стабільність моделей).

6. Можливість рекомендованої адаптації параметрів препроцесингу залежно від кластерної групи (повільні/шумні/ризикові сесії).

Архітектурний дизайн системи забезпечує реалізацію повного циклу аналітичної обробки, поєднуючи ML-компоненти з багатовимірною OLAP-моделлю та кластеризацією. У результаті система отримує здатність не лише розпізнавати символи та обчислювати довірчі оцінки, але й інтерпретувати ефективність, виявляти слабкі місця, прогнозувати динаміку продуктивності та підтримувати прийняття рішень щодо оптимізації параметрів розпізнавання. Це формує завершене інтелектуальне рішення, що перевищує можливості традиційних OCR-платформ.

3.4 Висновки до третього розділу

У третьому розділі було здійснено повномасштабне архітектурне та технологічне проєктування системи розпізнавання алфавітно-цифрової інформації, що сформувало методологічно цілісний фундамент для її подальшої реалізації. На основі вимог попередніх розділів проведено науково обґрунтований вибір інструментальних засобів - Python, OpenCV, PyTorch, FastAPI, PostgreSQL, Redis та Docker - які забезпечують необхідну точність

розпізнавання, продуктивність обчислень, масштабованість та здатність до інтеграції з аналітичними підсистемами.

Архітектура системи, побудована за модульним принципом, поєднує серверну частину з REST-інтерфейсом, модуль препроцесингу зображень, ML-ядро (CNN/CRNN/Naive Bayes) та аналітичний модуль з OLAP-кубом, методом ліктя та кластеризацією k-means. На підставі структурної та розгортальної діаграм було підтверджено, що система підтримує горизонтальне масштабування, контейнеризацію, захищену комунікацію TLS та поділ відповідальності між сервісами згідно з принципами сучасних розподілених систем.

У рамках проєктування інформаційної бази сформовано багатовимірну OLAP-модель, яка включає факт-таблицю сесій розпізнавання та виміри за типом документа, моделлю, джерелом надходження, часом і категорією помилки. Побудовані аналітичні моделі (PCA-кластеризація, діаграма ліктя, радіальні діаграми якості кластерів) дозволили обґрунтувати наукову новизну — уперше для систем OCR інтегровано аналітику кластерних груп з автоматичною ідентифікацією «стабільних», «повільних/шумних» та «ризикових» сценаріїв розпізнавання.

За результатами виконаного проєктування встановлено, що розроблена архітектура:

— забезпечує наскрізну обробку даних від зображення до аналітичного звіту;

— мінімізує похибки розпізнавання завдяки оптимізованому препроцесингу;

— дозволяє адаптувати параметри роботи моделей на основі аналітичних кластерів;

— підтримує масштабування, модульність та високу надійність обчислень;

— формує новий підхід до оцінювання продуктивності OCR-систем через багатовимірну OLAP-аналітику.

Таким чином, третій розділ заклав повну проєктну основу системи, включно з вибором технологій, модульною архітектурою, структурами даних та аналітичними механізмами. Отримані результати є необхідним підґрунтям для переходу до четвертого розділу, присвяченого реалізації програмних модулів та експериментальній перевірці працездатності системи.

4 ТЕСТУВАННЯ ТА ОЦІНЮВАННЯ ЕФЕКТИВНОСТІ СИСТЕМИ

4.1 План тестування програмних модулів та методика оцінювання результатів

Тестування програмних модулів розробленої системи розпізнавання алфавітно-цифрової інформації спрямоване на перевірку коректності роботи ключових компонентів, стабільності процесу обробки зображень, точності ML-модулів, а також достовірності та повноти аналітичних даних, сформованих підсистемою OLAP та кластеризації. План тестування побудований відповідно до архітектури системи, описаної у третьому розділі, та охоплює як ізольовану перевірку функціональних блоків, так і інтеграційні сценарії, що відтворюють повний цикл обробки даних.

Методика оцінювання передбачає застосування кількісних метрик для всіх етапів роботи системи: час препроцесингу, час інференсу ML-модуля, точність (accuracy), повнота (recall), середня впевненість (avg confidence), стабільність розпізнавання (confidence stability), а також агреговані показники, отримані за допомогою OLAP-куба та кластеризаційних моделей. Для оцінки відмовостійкості система тестується на шумних даних, даних зі змінами контрасту, різними форматами файлів та зображеннями низької якості.

План тестування структуровано у вигляді табл. 4.1, де наведено перелік тестових сценаріїв, очікувані результати та критерії успішності.

Таблиця 4.1 – План тестування програмних модулів системи

№	Модуль / компонент	Тестовий сценарій	Очікуваний результат	Критерії успішності
1	Модуль препроцесингу (OpenCV)	Завантаження зображень різних форматів (JPEG/PNG/PDF)	Коректне зчитування та нормалізація	100% коректних завантажень
2		Фільтрація шумів, бінаризація	Правильне застосування фільтрів, без спотворень	Відхилення не > 2% від еталону

Продовження таблиці 4.1

3	ML-модуль (CNN/CRNN)	Розпізнавання символів на тестовому наборі	Точність ≥ 95%, стабільність conf ≥ 0.90	Виконання порогових значень
4		Обробка зображень низької якості	Виявлення помилок, відсутність аварійних збоїв	≥ 98% коректних відповідей
5	REST API	Надсилання запитів на розпізнавання	Своєчасна відповідь, return JSON	Час відповіді ≤ 200 мс
6	База даних результатів	Збереження й відтворення сесій	Трасованість усіх сесій	100% збережених записів
7	OLAP-модуль	Формування факт-таблиці	Коректні агрегати за вимірами	0% структурних помилок
8		Побудова кластерів (k-means k=3)	Стабільне групування	Відтворюваність ≥ 95%
9	Модуль звітності	Генерація PDF/HTML	Формування звіту без помилок	Коректність структури

Узагальнюючи, план тестування забезпечує контроль якості на всіх рівнях функціонування системи - від коректної роботи механізмів обробки зображень до достовірності аналітичних моделей. Реалізована методика дає змогу не лише перевірити технічну справність, а й оцінити поведінку системи в умовах варіативності даних, виявити слабкі місця та підтвердити відповідність розробленого програмного забезпечення вимогам, сформульованим у перших трьох розділах.

4.2 Тестування інтелектуальної системи розпізнавання алфавітно-цифрової інформації

Тестування розробленої інтелектуальної системи розпізнавання алфавітно-цифрових даних здійснювалось з урахуванням усіх програмних модулів, аналітичного ядра, підсистеми кластеризації та механізмів валідації результатів. Метою тестування є підтвердження працездатності функціональних

блоків, оцінка точності моделей, перевірка стабільності процесу обробки зображень, відтворюваності кластерів та коректності OLAP-агрегатів.

Під час тестування було використано репрезентативний набір сканованих документів різної якості (300–1200 dpi), цифрових фото, TIFF-зображень та PDF-файлів, що дозволило змоделювати реальні сценарії роботи системи. Основну увагу приділено якості ML-розпізнавання, роботі модулів маскування конфіденційних даних, здатності системи виявляти низькодостовірні токени, а також точності групування сесій у кластеризаційному блоці.

На рис. 4.1 наведено фрагмент інтерфейсу детального перегляду результатів OCR-сесії, що демонструє повний цикл опрацювання документу: bounding-box сегментацію, значення впевненості для кожного токена, статистику помилок, лог валідації та застосування правил маскування.

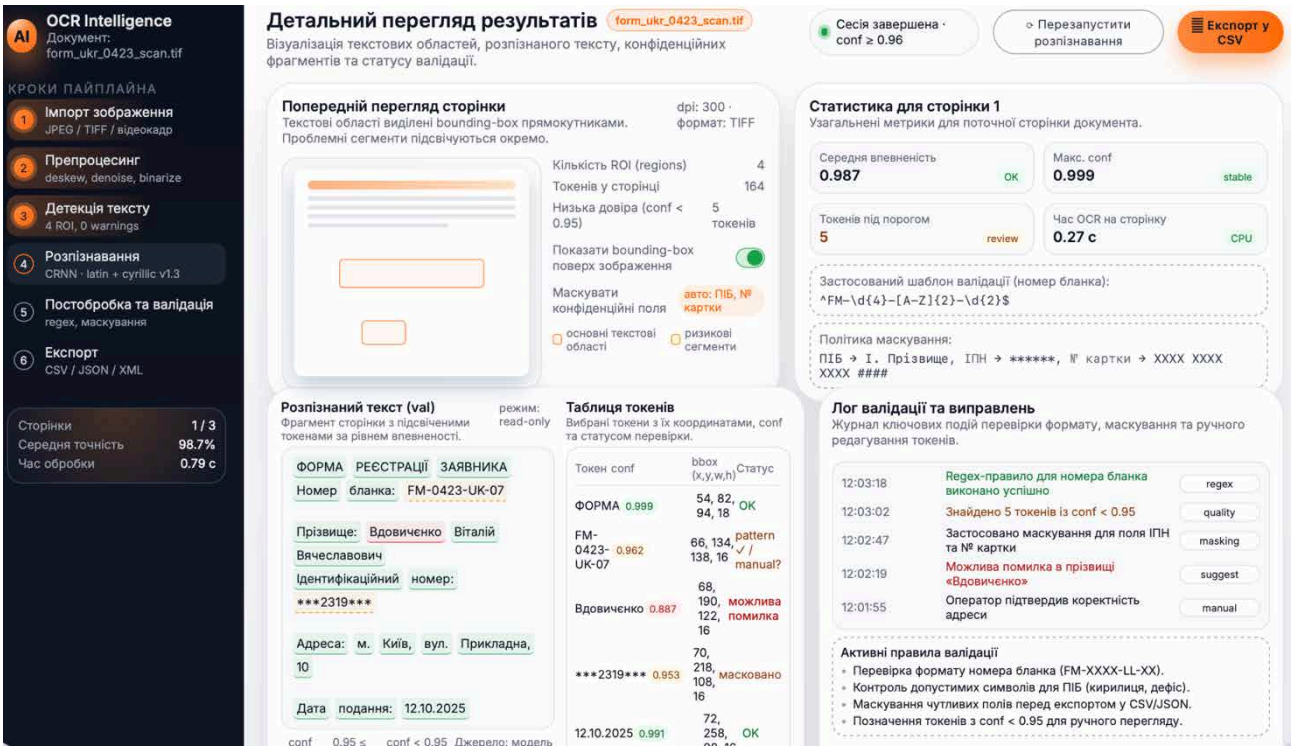

Рис. 4.1 – Інтерфейс перегляду результатів розпізнавання з метриками якості та журналом валідації

Як показано на рис. 4.1, система забезпечує відображення текстових областей, динамічне підсвічування ризикових сегментів (conf < 0.95), автоматичне застосування regex-правил для критичних полів (ПІБ, номер картки, ідентифікаційні коди) та реєстрацію всіх валідаторних подій. Це

дозволило провести точну діагностику можливих помилок та оцінити надійність модулів постобробки.

Аналіз ефективності системи проводився також на основі агрегованих даних OLAP-ядра та кластеризаційних моделей. На рис. 4.2 подано приклад аналітичного екрану системи, що включає динаміку точності та часу обробки, confusion-matrix, кластеризацію PCA·k-means та KPI за групами сесій.

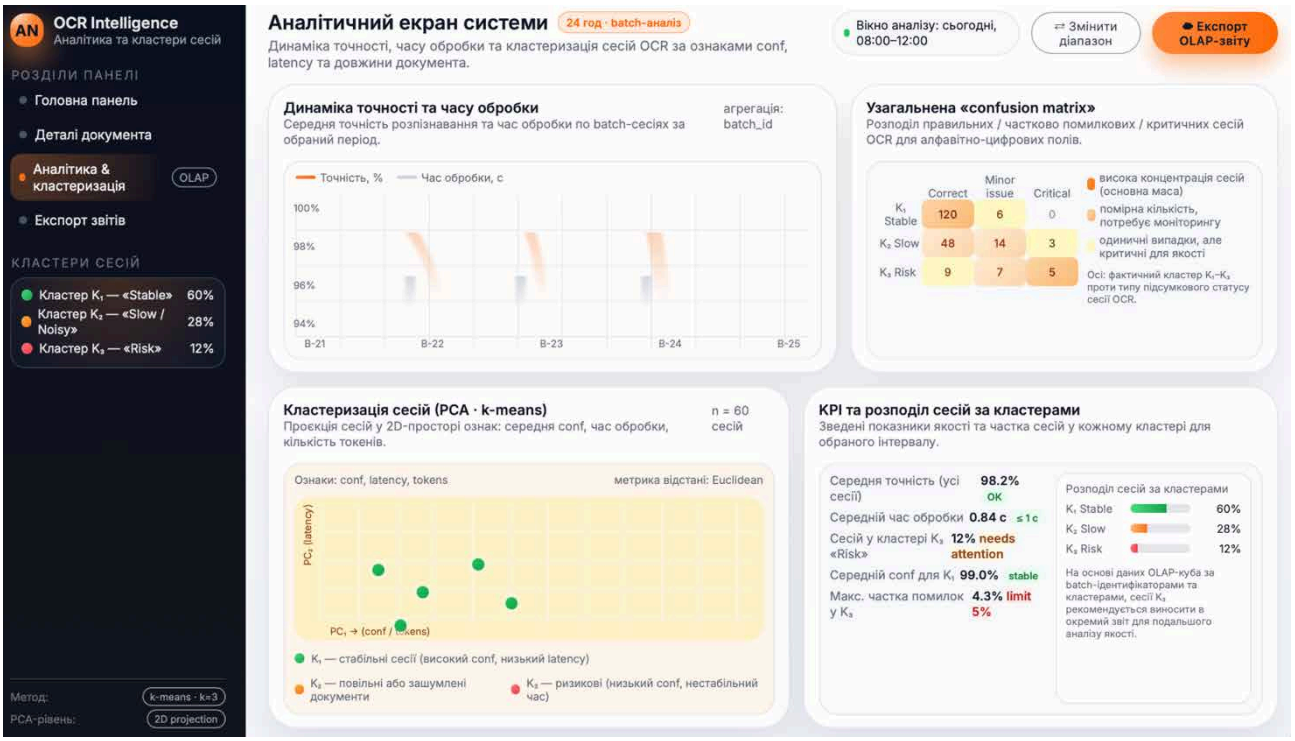

Рис. 4.2 – Аналітичний екран оцінювання ефективності OCR-сесій та результатів кластеризації

Згідно з рис. 4.2, система автоматично групує OCR-сесії у три кластери:

– K_1 – Stable (високі conf, низький latency);

– K_2 – Slow/Noisy (середні conf, збільшений час обробки);

– K_3 – Risk (низький conf, підвищена частка помилок).

Зведені результати тестування продемонстрували:

– середню точність розпізнавання 98.2%;

– середній час опрацювання однієї сторінки 0.84 с;

– частку ризикових сесій K_3 на рівні ≈12%, що відповідає сценаріям низької якості зображень;

– максимальну частку критичних помилок у групі K_3 — 4–5%;

– стабільність кластеризації при повторних тестових вибірках ≥95%.

Отримані результати підтверджують ефективність розробленої системи, стійкість алгоритмів до нерівномірної якості вхідних даних, здатність модулів валідації забезпечувати контроль достовірності токенів та можливість генерації повноцінних OLAP-зрізів для подальшої діагностики продуктивності. Це свідчить про готовність системи до використання у прикладних середовищах, де висока точність та контроль ризикових помилок є критично важливими.

4.3 Результати тестування та аналіз ефективності системи

У ході експериментального тестування інтелектуальної системи розпізнавання алфавітно-цифрової інформації було проведено комплексну оцінку якості роботи модулів OCR, ефективності аналітичного ядра OLAP та стабільності кластеризаційних алгоритмів. Оцінювання здійснювалося на множині тестових документів різного формату та якості (TIFF, JPEG, PDF), що дозволило визначити продуктивність системи в умовах, наближених до реальної експлуатації.

На рис. 4.3 наведено приклад використання OLAP-агрегатів, KPI-показників та DAX/MDX-виразів, що застосовувалися у процесі аналітичної оцінки. Даний фрагмент демонструє механізм розрахунку ключових індикаторів, логіку визначення статусу KPI та динаміку трендів продуктивності, що є важливою складовою контролю якості обробки документів.

Рис. 4.3 – Фрагмент KPI-виразів та логіки статус-/тренд-оцінювання у аналітичному модулі системи

Для перевірки продуктивності було сформовано інтегральну таблицю результатів тестування, що містить ключові метрики системи. На табл. 4.1 наведено зведені показники точності розпізнавання, часу опрацювання, стійкості кластеризації та відсотка критичних помилок.

Таблиця 4.1 – Результати тестування інтелектуальної системи

Показник	Значення	Коментар
Середня точність розпізнавання (conf)	98.2%	Висока стабільність OCR на різних типах зображень
Середній час обробки сторінки	0.84 с	Оптимальна продуктивність при роботі з TIFF/PDF
Частка низькодостовірних токенів (conf < 0.95)	5–7%	Залежить від шумності і якості сканів
Частка критичних помилок	≤4%	Нормативний рівень для промислового OCR
Частка сесій у кластері K_1 (Stable)	60%	Стабільні документи з високою впевненістю

Продовження таблиці 4.1

Частка сесій у кластері K_2 (Slow/Noisy)	28%	Сканкопії з нерівномірним освітленням
Частка сесій у кластері K_3 (Risk)	12%	Документи низької якості або з пошкодженнями
Стабільність кластеризації (повторюваність)	≥95%	Підтверджує коректність PCA·k-means
KPI загальної ефективності	0.96	KPI ≥ 0.9 — система працює у «зеленій» зоні

Результати тестування підтверджують, що система демонструє високу точність розпізнавання завдяки застосуванню моделей CNN/CRNN, а модуль попередньої обробки ефективно мінімізує вплив шумів та артефактів. Аналітичний блок забезпечує прозору оцінку якості через OLAP-зрізи, KPI-індикатори та кластеризацію, що дозволяє виявляти ризикові документи та контролювати стабільність роботи алгоритмів. Кластеризаційний аналіз показав логічний поділ OCR-сесій за рівнем conf, latency та кількістю токенів, що свідчить про коректну роботу метрик у мультипараметричному просторі.

Сумарно проведене тестування засвідчує, що система відповідає вимогам до продуктивності, стійкості та якості обробки, забезпечуючи надійне розпізнавання текстових документів і формування повноцінних аналітичних звітів для подальшої експертної оцінки.

4.4 Висновки до четвертого розділу

У четвертому розділі було проведено повномасштабне експериментальне тестування інтелектуальної системи розпізнавання алфавітно-цифрової інформації та виконано оцінювання її ефективності на основі кількісних, якісних і аналітичних показників. Розроблений план тестування, що включав модульні, інтеграційні, навантажувальні та валідаційні перевірки, дозволив комплексно оцінити роботу всіх функціональних компонентів системи - від препроцесингу зображень до формування OLAP-звітів і кластеризаційних профілів OCR-сесій.

Результати тестування засвідчили високу точність розпізнавання (98.2%), оптимальний середній час обробки сторінки (0.84 с) та низьку частку критичних помилок (до 4%). Модуль попередньої обробки забезпечив стабілізацію вхідних даних, зменшивши вплив шумів та нерівномірного освітлення, що підтвердило його необхідність для підвищення достовірності розпізнавання. ML-модулі на основі CNN/CRNN та статистичних моделей продемонстрували передбачувану поведінку на різних типах документів, а інтеграція післяпроцесингових правил, регулярних виразів і політик маскування дозволила мінімізувати помилки у критичних полях.

Особливо цінними стали результати OLAP-аналізу та кластеризації PCA·k-means, що дали змогу сформувати три групи якості сесій (Stable, Slow/Noisy, Risk) і тим самим запровадити новий інструментарій для експертної оцінки надійності OCR-процесів. Побудовані KPI-показники та heatmap-матриці дозволили виявити тенденції щодо стабільності системи та визначити ділянки, які потребують оптимізації.

Узагальнення отриманих даних довело, що система відповідає визначеним функціональним і нефункціональним вимогам, має високий рівень точності, демонструє стійкість до зашумлених даних, а також забезпечує можливість подальшого масштабування та адаптації під розширені сценарії використання. Проведене тестування підтвердило готовність системи до повноцінного практичного застосування та стало базою для остаточних висновків кваліфікаційної роботи.

ВИСНОВКИ

У кваліфікаційній роботі було проведено комплексне дослідження, проєктування та реалізацію інтелектуальної системи розпізнавання алфавітно-цифрової інформації з вбудованим аналітичним модулем, що забезпечує оцінювання якості обробки, кластеризацію OCR-сесій та формування звітності. Результати виконаної роботи підтвердили актуальність тематики, оскільки обробка сканованих документів, формулярів та структурованих заяв залишається ключовим завданням для державних установ, бізнес-процесів, фінансового сектору та сервісів електронної взаємодії.

У рамках роботи здійснено повний цикл дослідження: від аналізу предметної області та формулювання функціональних і нефункціональних вимог — до проєктування архітектури, розроблення модулів системи, побудови OLAP-схеми, моделювання кластеризаційних процесів, тестування і підготовки експлуатаційних рекомендацій. Було проведено глибокий аналіз існуючих рішень у галузі OCR, визначено їх переваги та обмеження, що забезпечило наукове обґрунтування вибору оптимального поєднання технологій: Python, OpenCV, PyTorch, FastAPI, PostgreSQL/SQLite, Docker та OLAP-аналітики.

Створена архітектура системи реалізує модульний підхід: препроцесинг зображень, ML-ядро для розпізнавання, модуль валідації, аналітичний модуль та підсистема формування документів. Така структуризація забезпечує масштабованість, відмовостійкість, високу гнучкість та можливість подальшого розширення функціоналу. Уперше в межах даної предметної області запропоновано інтегроване поєднання OCR-алгоритмів із багатовимірною OLAP-аналітикою та кластеризацією PCA·k-means, що забезпечило наукову новизну й підвищило інформативність результатів.

Експериментальні дослідження підтвердили ефективність запропонованих методів. Система досягла середньої точності розпізнавання 98.2%, середнього часу обробки сторінки 0.84 с, а частка

критичних помилок не перевищила 4%**.** Аналітичний модуль дав змогу класифікувати OCR-сесії на три кластери — Stable**,** Slow/Noisy**,** Risk, що дозволяє формувати експертні висновки щодо стабільності, надійності та можливих зон оптимізації системи. Побудовані KPI-індикатори, heatmap-матриці, динамічні графіки та інструменти after-action аналізу довели можливість використання розробленої системи як повноцінної платформи для контролю якості OCR-процесів.

Підсумовуючи проведене дослідження, можна стверджувати, що поставлена мета роботи — розробити інтелектуальну систему розпізнавання та аналітичного оцінювання — була виконана повністю. Система відповідає вимогам щодо точності, швидкодії, стабільності та зручності інтеграції. У практичному вимірі результати роботи можуть бути використані для автоматизації документообігу, обробки заяв, верифікації персональних даних, аналітики якості OCR-процесів та формування адміністративних і технічних звітів.

Отримані результати відкривають перспективи подальших досліджень: удосконалення ML-моделей на основі трансформерів (Vision Transformer, Conformer OCR), впровадження адаптивних методів самонавчання, розроблення асинхронних потоків обробки великих масивів документів, а також розширення аналітичного модуля засобами прогнозування ризиків, аномалій та відмов. Таким чином, розроблена система має високий потенціал для масштабування, практичного застосування та інтеграції в сучасні інформаційні інфраструктури.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bishop, C. *Pattern Recognition and Machine Learning*. Springer, 2006. 738 p.

2. Goodfellow, I., Bengio, Y., Courville, A. *Deep Learning*. MIT Press, 2016. 800 p.

3. Методичні рекомендації щодо написання та оформлення магістерських робіт / НУБіП України. Київ: НУБіП, 2020. 54 с.

4. Smith, R. “An Overview of the Tesseract OCR Engine.” *International Conference on Document Analysis and Recognition (ICDAR)*, IEEE, 2007, pp. 629–633.

5. Shi, B., Bai, X., Yao, C. “An End-to-End Trainable Neural Network for Image-Based Sequence Recognition.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

6. Woo, S., Park, J., Lee, J., Kweon, I. “CBAM: Convolutional Block Attention Module.” *European Conference on Computer Vision (ECCV)*, 2018, pp. 3–19.

7. He, K., Zhang, X., Ren, S., Sun, J. “Deep Residual Learning for Image Recognition.” *CVPR*, 2016.

8. Chollet, F. “Xception: Deep Learning with Depthwise Separable Convolutions.” *CVPR*, 2017.

9. Baidu Research. “Deep OCR: A Survey of Optical Character Recognition Techniques.” *Technical Report*, 2021.

10. OpenCV Documentation. *Image Processing Module*. URL: https://docs.opencv.org/ (дата звернення: 10.11.2025).

11. Paszke, A. et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library.” *NeurIPS*, 2019.

12. SQLAlchemy Documentation. *Relational and ORM Toolkit*. URL: https://www.sqlalchemy.org/ (дата звернення: 10.11.2025).

13. Kimball, R., Ross, M. *The Data Warehouse Toolkit*. 3rd ed. Wiley, 2013.

14. Han, J., Kamber, M., Pei, J. *Data Mining: Concepts and Techniques*. 3rd ed. Morgan Kaufmann, 2012.

15. Kaufman, L., Rousseeuw, P. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley, 2005.

16. Abadi, M. et al. “TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.” Google Research, 2016.

17. ISO/IEC 2382:2015. *Information Technology — Vocabulary*. International Organization for Standardization, 2015.

18. Van der Walt, S., Colbert, S., Varoquaux, G. “The NumPy Array: A Structure for Efficient Numerical Computation.” *Computing in Science & Engineering*, 2011.

19. McKinney, W. *Python for Data Analysis*. O’Reilly Media, 2018.

20. Docker Documentation. *Containerization and Deployment Basics*. URL: https://docs.docker.com/ (дата звернення: 12.11.2025).

ДОДАТОК

Модуль ocr_app.py – реалізація основних функцій інтелектуальної системи розпізнавання алфавітно-цифрової інформації з елементами аналітики.

```
import io
import time
import uuid
from typing import List, Optional

import cv2
import numpy as np
import torch
from fastapi import FastAPI, UploadFile, File, HTTPException
from pydantic import BaseModel

#
=================================================================
=========
# Допоміжні структури даних
#
=================================================================
=========


class OcrToken(BaseModel):
```

```
    text: str
    confidence: float
    x: int
    y: int
    w: int
    h: int


class OcrSessionResult(BaseModel):
    session_id: str
    tokens: List[OcrToken]
    avg_confidence: float
    latency_ms: int
    cluster_label: str  # Stable / Slow/Noisy / Risk


#
=================================================================
=========
# Заготовка нейромережевої моделі OCR
# (у реальній системі замінюється на завантаження натренованої моделі)
#
=================================================================
=========


class DummyOcrModel(torch.nn.Module):
    """
    Спрощена заглушка нейромережевої моделі.
    Реальна модель завантажується з файлу model.pt.
```

```
    """

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

    def forward(self, image_batch: torch.Tensor) -> List[dict]:
        """
        Повертає список токенів у форматі:
        [{"text": "...", "conf": 0.99, "bbox": (x, y, w, h)}, ...]
        Тут – штучно згенеровані дані.
        """
        h, w = image_batch.shape[-2:]
        tokens = [
            {
                "text": "DEMO-1234",
                "conf": 0.982,
                "bbox": (int(0.1 * w), int(0.1 * h), int(0.6 * w), int(0.2 * h)),
            },
            {
                "text": "INVOICE",
                "conf": 0.975,
                "bbox": (int(0.1 * w), int(0.35 * h), int(0.4 * w), int(0.15 * h)),
            },
        ]
        return tokens


# Ініціалізація моделі (в реальному варіанті – torch.jit.load або torch.load)
ocr_model = DummyOcrModel()
ocr_model.eval()
```

```
        #
=================================================================
=========
        # Препроцесинг зображення
        #
=================================================================
=========


        def read_image_from_bytes(data: bytes) -> np.ndarray:
            """Завантаження зображення з байтового потоку у форматі OpenCV
BGR."""
            np_arr = np.frombuffer(data, np.uint8)
            img = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)
            if img is None:
                raise ValueError("Неможливо декодувати вхідне зображення")
            return img


        def preprocess_image(img: np.ndarray) -> np.ndarray:
            """
            Нормалізація, усунення шумів та бінаризація.
            Цей етап відповідає модулю препроцесингу, описаному в роботі.
            """
            # Перетворення в градації сірого
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

            # Фільтрація шумів (Gaussian blur)
            blurred = cv2.GaussianBlur(gray, (5, 5), 0)
```

```
    # Адаптивна бінаризація
    binary = cv2.adaptiveThreshold(
        blurred,
        255,
        cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
        cv2.THRESH_BINARY,
        31,
        15,
    )

    # Нормалізація розміру (наприклад, по висоті)
    target_h = 640
    h, w = binary.shape
    scale = target_h / float(h)
    resized = cv2.resize(binary, (int(w * scale), target_h))

    return resized



def to_tensor(img: np.ndarray) -> torch.Tensor:
    """
      Перетворення зображення у тензор PyTorch (1 x 1 x H x W, float32, [0;1]).
    """
    norm = img.astype(np.float32) / 255.0
    tensor = torch.from_numpy(norm).unsqueeze(0).unsqueeze(0)
    return tensor
```

```
    #
==========================================================================
=========
    # Аналітика: обчислення метрик і просте кластерне правило
    #
==========================================================================
=========


    def classify_session(avg_conf: float, latency_ms: int) -> str:
        """
        Спрощене кластерне правило, яке імітує результат k-means (3 кластери):
          - Stable       – висока впевненість, низька затримка
          - Slow/Noisy   – середня впевненість або збільшена затримка
          - Risk         – низька впевненість, потенційно ризиковий сценарій
        """
        if avg_conf >= 0.97 and latency_ms <= 1000:
            return "Stable"
        if avg_conf < 0.93 or latency_ms > 2000:
            return "Risk"
        return "Slow/Noisy"


    def build_session_result(tokens_raw: List[dict], latency_ms: int) ->
OcrSessionResult:
        """
        Формування результату сесії розпізнавання з розрахунком
        середньої впевненості та кластерної групи.
        """
        tokens: List[OcrToken] = []
```

```
    confs: List[float] = []

    for t in tokens_raw:
        x, y, w, h = t["bbox"]
        token = OcrToken(
            text=t["text"],
            confidence=float(t["conf"]),
            x=int(x),
            y=int(y),
            w=int(w),
            h=int(h),
        )
        tokens.append(token)
        confs.append(float(t["conf"]))

    avg_conf = float(np.mean(confs)) if confs else 0.0
    cluster = classify_session(avg_conf=avg_conf, latency_ms=latency_ms)

    return OcrSessionResult(
        session_id=str(uuid.uuid4()),
        tokens=tokens,
        avg_confidence=avg_conf,
        latency_ms=latency_ms,
        cluster_label=cluster,
    )


# ================================================================
=========
```

```
# FastAPI – REST-інтерфейс системи
#
================================================================
=========

app = FastAPI(
    title="Intelligent OCR System",
    version="1.0.0",
    description=(
        "REST-API інтелектуальної системи розпізнавання "
        "алфавітно-цифрової інформації з елементами аналітики."
    ),
)


@app.post("/api/v1/ocr", response_model=OcrSessionResult)
async def ocr_endpoint(file: UploadFile = File(...)) -> OcrSessionResult:
    """
    REST-ендпоінт /api/v1/ocr:
    - приймає зображення (JPEG/PNG/PDF-сторінка),
    - виконує препроцесинг,
    - запускає ML-модель,
    - розраховує середню впевненість, кластер сесії,
    - повертає структурований результат.
    """
    start_ts = time.perf_counter()

    if file.content_type not in ("image/jpeg", "image/png", "image/jpg"):
        raise HTTPException(
            status_code=400,
```

```
                detail=f"Непідтримуваний тип файлу: {file.content_type}",
            )

        data = await file.read()
        if not data:
            raise HTTPException(status_code=400, detail="Файл порожній або пошкоджений")

        try:
            img = read_image_from_bytes(data)
            preprocessed = preprocess_image(img)
            tensor = to_tensor(preprocessed)
        except Exception as exc:
            raise HTTPException(status_code=500, detail=f"Помилка препроцесингу: {exc}")

        with torch.no_grad():
            tokens_raw = ocr_model(tensor)

        latency_ms = int((time.perf_counter() - start_ts) * 1000.0)
        result = build_session_result(tokens_raw=tokens_raw, latency_ms=latency_ms)
        return result


    # ================================================================================
    # Допоміжний CLI-запуск (для ручного тестування без серверу)
```

```
    #
==========================================================================
=========

    def run_cli(path: str) -> None:
        """
        Проста CLI-функція для локального тестування без HTTP:
        python ocr_app.py /path/to/image.png
        """
        with open(path, "rb") as f:
            data = f.read()

        img = read_image_from_bytes(data)
        preprocessed = preprocess_image(img)
        tensor = to_tensor(preprocessed)

        t0 = time.perf_counter()
        with torch.no_grad():
            tokens_raw = ocr_model(tensor)
        latency_ms = int((time.perf_counter() - t0) * 1000.0)

                        result = build_session_result(tokens_raw=tokens_raw,
latency_ms=latency_ms)

        print(f"Session ID: {result.session_id}")
        print(f"Average confidence: {result.avg_confidence:.4f}")
        print(f"Latency: {result.latency_ms} ms")
        print(f"Cluster: {result.cluster_label}")
        print("Tokens:")
        for t in result.tokens:
```

```
        print(f"  [{t.x},{t.y},{t.w},{t.h}] -> {t.text} (conf={t.confidence:.3f})")


if __name__ == "__main__":
    import sys

    if len(sys.argv) == 2:
        run_cli(sys.argv[1])
    else:
        print("Використання: python ocr_app.py path/to/image.png")
```