БАКАЛАВРСЬКА РОБОТА

КРБ.СІ – 02.00.000 ПЗ

Група СІ-21-1

Роман Боярський

2025

Міністерство освіти і науки України

Івано-Франківський національний університет нафти і газу

Інститут інформаційних технологій

Кафедра інформаційно-телекомунікаційних технологій та систем

Боярський Роман Романович

(прізвище, ім'я, по-батькові)

УДК 004.8
(індекс)

БАКАЛАВРСЬКА РОБОТА

Розроблення системи розпізнавання тексту з використанням технологій машинного зору
(назва роботи)

Системна інженерія - Інтернет речей
(назва освітньої програми)

151 Автоматизація та комп'ютерно-інтегровані технології
(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня _____ Р. Р. Боярський
(підпис, ініціали та прізвище здобувача)

Науковий керівник _____ Паньків Х. В. к.т.н., доцент кафедри ІТТС
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

д.т.н., професор _____ Л. М. Заміховський
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

(повне найменування закладу вищої освіти)

Інститут інформаційних технологій

Кафедра інформаційно-телекомунікаційних технологій та систем

Освітній рівень бакалавр

Спеціальність 151 Автоматизація та комп'ютерно-інтегровані технології

(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри ІТТС

д.т.н., проф. Л. М. Заміховський

«___» _____2025 року

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Боярському Роману Романовичу

(прізвище, ім'я, по-батькові)

1. Тема роботи *Розроблення системи розпізнавання тексту з використанням технологій машинного зору*
   керівник роботи *Паньків Х. В. к.т.н., доцент кафедри ІТТС*,
   затверджені наказом закладу вищої освіти від «05» травня 2025 року №281/7
2. Строк подання студентом роботи *15.06.2025*
3. Вихідні дані до роботи: *мінімальна точність розпізнавання не менше 85%, дані зібрані під час проходження переддипломної практики*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   1. Аналіз сучасних методів та технологій розпізнавання тексту на основі машинного зору
   2. Розроблення методу розпізнавання тексту
   3. Розроблення програмного забезпечення системи та аналіз її роботи
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
   1. Система розпізнавання тексту. Схема структурна
   2. Розташування виводів ESP32-CAM. Схема функціональна
   3. Веб-інтерфейс системи розпізнавання тексту. Схема функціональна
6. Дата видачі завдання *01.03.2025*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Термін виконання етапів роботи	Примітка
1	*Аналіз сучасних методів та технологій розпізнавання тексту на основі машинного зору*	*1.04.2025*	*виконав*
2	*Розроблення методу розпізнавання тексту*	*1.05.2025*	*виконав*
3	*Розроблення програмного забезпечення системи та аналіз її роботи*	*1.06.2025*	*виконав*
4	*Оформлення пояснювальної записки*	*14.06.2025*	*виконав*

Студент _____ Боярський Р. Р.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Паньків Х. В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Боярський Р. Р. Розроблення системи розпізнавання тексту з використанням технологій машинного зору ІФНТУНГ, 2025. 64 с.

Бакалаврська робота на здобуття освітнього ступеня бакалавр за освітньо-професійною програмною «Системна інженерія – Інтернет речей», спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології». Івано-Франківський національний університет нафти і газу. Івано-Франківськ, 2025.

У дослідженні проаналізовано сучасні методи розпізнавання тексту на основі технологій машинного зору.

У роботі розроблено систему, що включає модуль зйомки на основі ESP32-CAM, серверну частину для обробки зображень із використанням FastAPI, OpenCV і Tesseract OCR, а також клієнтський веб-інтерфейс на Vue.js для взаємодії з користувачем. Систему протестовано, підтверджено її ефективність та надано рекомендації щодо покращень.

Ключові слова: МАШИННИЙ ЗІР, РОЗПІЗНАВАННЯ ТЕКСТУ, OCR, ESP32-CAM, OPENCV, TESSERACT OCR, FASTAPI, VUE.JS, ВЕБ-ІНТЕРФЕЙС, IOT

ANNOTATION

Boiarskyi R. R. Development of a text recognition system using machine vision technologies. IFNTUNG, 2025, 64 p.

Bachelor's thesis for obtaining a bachelor's degree under the educational-professional program «System Engineering – Internet of Things», specialty 151 «Automation and computer-integrated technologies». Ivano-Frankivsk National Technical University of Oil and Gas. Ivano-Frankivsk, 2025.

The study analyzed modern text recognition methods based on machine vision technologies.

The paper develops an integrated software and hardware system comprising an ESP32-CAM based image capture module, a backend server built with FastAPI, OpenCV and Tesseract OCR for image processing, and a user-facing web interface created with Vue.js. The system was tested, its performance was evaluated, and suggestions for further improvement were provided.

Keywords: MACHINE VISION, TEXT RECOGNITION, ESP32-CAM, OPENCV, TESSERACT OCR, FASTAPI, VUE.JS WEB INTERFACE, IOT

РЕФЕРАТ

Розрахункова-пояснювальна записка: 63 с., 48 рисунків, 7 табл., 1 додаток, 50 джерел.

Об’єкт дослідження – розпізнавання тексту з використанням технологій машинного зору.

Предмет дослідження – методи та засоби автоматичного розпізнавання тексту на основі апаратної платформи ESP32-CAM, програмних інструментів OpenCV та Tesseract OCR, засоби розробки серверної частини FastAPI та клієнтської частини Vue.js.

Мета роботи – розроблення системи розпізнавання тексту з використанням технологій машинного зору, веб технологій та інтернету речей.

Програмний продукт реалізовано за допомогою мови програмування Python, бібліотек OpenCV і Tesseract OCR, фреймворку FastAPI, засобів фронтенд-розробки на базі Vue.js та платформи ESP32 (ESP32-CAM).

У дослідженні проаналізовано сучасні методи та технології розпізнавання тексту, обґрунтовано вибір апаратної та програмної частини проєкту, розроблено програмний додаток, проведено його тестування та аналіз ефективності роботи.

МАШИННИЙ ЗІР, РОЗПІЗНАВАННЯ ТЕКСТУ, OCR, ESP32-CAM, OPENCV, TESSERACT OCR, FASTAPI, VUE.JS, ВЕБ-ІНТЕРФЕЙС, IOT

ЗМІСТ

с.

					КРБ.СІ-02.00.000 ПЗ		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Боярський Р. Р.			*Розроблення системи розпізнавання тексту з використанням технологій машинного зору*	*Літ.*	*Арк.* 7 / *Аркушів* 63
Перевірив		Паньків Х. В.					
Н. Контр.		Возний А. В.				ІФНТУНГ СІ-21-1	
Затвердив		Заміховський Л. М.					

ПЕРЕЛІК ОСНОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

OCR – Optical Character Recognition (Оптичне розпізнавання символів)

CNN – Convolution Neural Network (Згорткова нейронна мережа)

RNN – Recurrent Neural Network (Рекурентна нейронна мережа)

LSTM – Long Short-Term Memory (Довготривала короткочасна пам’ять)

API – Application Programming Interface (Інтерфейс прикладного програмування)

GPU – Graphics Processing Unit (Графічний процесор)

CPU – Central Processing Unit (Центральний процесор)

IoT – Internet of Things (Інтернет речей)

TTS – Text-To-Speech (Перетворення тексту в мовлення)

FastAPI – Fast Application Programming Interface (Фреймворк API)

OpenCV – Open Source Computer Vision Library (Бібліотека комп’ютерного зору з відкритим кодом)

ViT – Vision Transformer (Трансформер для обробки зображень)

ВСТУП

Сучасне суспільство генерує величезні обсяги текстової інформації, яка може існувати як у цифровій формі, так і в аналогових носіях – друкованих документах, сканах, фотографіях. Використання цих даних у цифрових системах вимагає попереднього перетворення в машиночитаний формат. Попри значний прогрес у розробці систем розпізнавання символів (OCR), розпізнавання тексту залишається складним завданням в умовах реального середовища – зашумлених зображень, неоднорідного фону, різноманіття шрифтів та освітлення.

В останні роки розвиток технологій машинного зору відкрив нові можливості для підвищення точності OCR-систем, особливо у випадках обробки зображень із нестандартними параметрами. Комбінація комп'ютерного зору та глибинного навчання дозволяє розробляти системи, які здатні ефективно інтерпретувати візуальну інформацію, адаптуючись до змін зовнішніх умов. На тлі цього все більшу увагу привертають легкі та недорогі апаратні рішення, зокрема платформа ESP-32CAM, яка, попри обмежені ресурси, демонструє високу придатність для задач машинного зору в умовах периферійної обробки.

Незважаючи на доступність окремих рішень, комплексні дослідження, які охоплюють повний цикл – від захоплення зображення до аналізу та інтеграції в клієнт-серверні системи – залишаються обмеженими. Особливо актуальними є створення відкритих, масштабованих і мобільних платформ для розпізнавання тексту, здатних працювати у змінних умовах, з мінімальними апаратними вимогами.

У цьому контексті дана робота спрямована на проєктування та реалізацію системи розпізнавання тексту з використанням ESP32-CAM, бібліотек OpenCV та Tesseract OCR, серверної частини на базі FastAPI і клієнтського інтерфейсу на Vue.js. Основне завдання – не лише ефективність розпізнавання, а й архітектурна цілісність системи, її модульність, швидкодія та зручність виконання.

Крім того, у роботі здійснено спробу оцінки потенціалу таких систем для застосування в реальних умовах – у сфері документообігу, освіти, логістики тощо. Особливу увагу приділено тестуванню на наборах зображень різної якості, аналізу продуктивності та побудові критеріїв ефективності.

Таким чином, метою роботи є створення доступної, функціональної системи розпізнавання текстової інформації з використанням сучасних методів машинного зору.

1 АНАЛІЗ СУЧАСНИХ МЕТОДІВ ТА ТЕХНОЛОГІЙ РОЗПІЗНАВАННЯ ТЕКСТУ НА ОСНОВНІ МАШИННОГО ЗОРУ

1.1 Огляд технологій машинного зору та їх застосування

Машинний зір як галузь штучного інтелекту активно розвивається з кінця XX століття. Сьогодні він охоплює широкий спектр теоретичних досліджень і практичних рішень, зокрема у сфері розпізнавання тексту. Вітчизняні та зарубіжні дослідження демонструють значний науковий інтерес до проблем аналізу та обробки візуальної інформації. Зокрема, машинний зір базується на досягненнях суміжних наук: математики, інформатики, оптики, електроніки, а також інформаційних технологій.

З огляду на стрімкий розвиток цифрових технологій та зростаючу потребу в автоматизації обробки зображень, машинний зір посідає важливе місце серед прикладних напрямків сучасного штучного інтелекту. Його впровадження дозволяє підвищити ефективність процесів у багатьох ключових галузях.

У сучасному світі обробка візуальної інформації займає важливе місце у багатьох сферах людської діяльності – від автоматизованого виробництва до автономного керування транспортом. Завдяки швидкому розвитку технологій та алгоритмів штучного інтелекту машини отримали можливість «бачити» та аналізувати зображення без людського втручання.

Одним із ключових напрямків у цьому процесі є машинний зір – технологія, що дозволяє системам отримувати, обробляти та інтерпретувати візуальні дані. Він поєднує методи комп’ютерного зору, цифрової обробки зображень та штучного інлекту, забезпечуючи машинну інспекцію, контроль, навігацію та прийняття рішень у реальному часі [1].

У типових системах машинного зору поєднуються цифрові камери, алгоритими обробки зображень, інтерфейси для взаємодії з користувачем. Така схема працює за принципом: зчитування – обробка – аналіз – дія. Схема технології машинного зору зображена на рисунку 1.1.

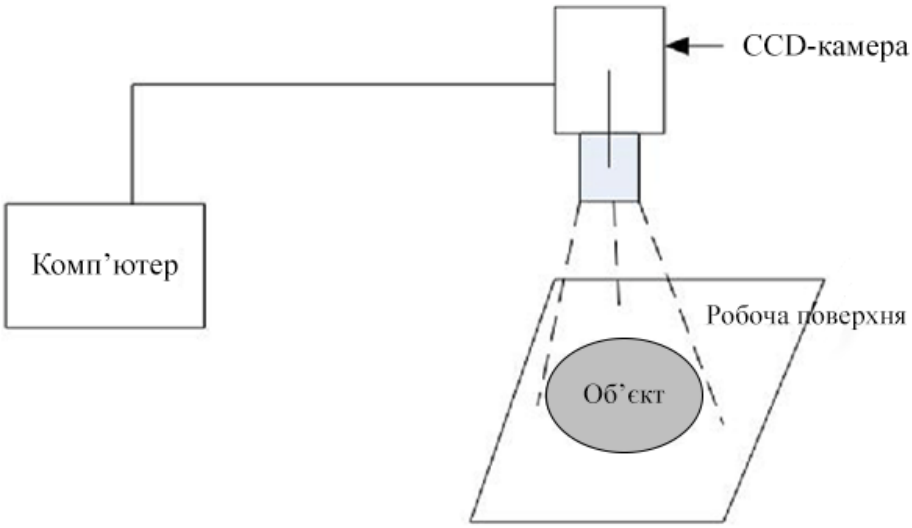


Рисунок 1.1 – Схема типової системи машинного зору

Наукові джерела аналізують питання алгоритмізації процесів машинного зору, детально описуючи моделі обробки зображень, методи попередньої обробки та виявлення об'єктів на зображенні. Велика увага приділяється методам оптичного розпізнавання символів (OCR), серед яких особливе місце займають нейронні мережі та методи глибокого навчання.

Становлення технологій машинного зору не було б можливим без технічних досягнень у галузі розробки камер високої роздільної здатності, засобі передачі даних, алгоритмів стиснення інформації. Важливим напрямом розвитку є оптимізація алгоритмів для використання в системах з обмеженими обчислювальними ресурсами, таких як ESP32-CAM.

Підсумовуючи огляд, можна стверджувати, що сфера машинного зору та розпізнавання тексту активно розвивається, але має низку нерозв'язаних питань, пов'язаних із точністю, швидкістю обробки та захистом даних, що й обґрунтовує необхідність подальших досліджень.

1.2 Аналіз методів оптичного розпізнавання символів (OCR)

Однією із центральних задач машинного зору є автоматичне розпізнавання тексту на зображеннях, відоме як Optical Character Recognition (OCR). OCR – технологія, яка конвертує зображення, які містять текст, у формат що може бути проаналізований і оброблений системою.

Основна перевага OCR полягає у можливості автоматичного пошуку, редагування та збереження тексту. Ця технологія значно оптимізує введення даних та покращує ефективність роботи з текстовою інформацією.

Процес оптичного розпізнавання символів включає кілька послідовних етапів, які дозволяють системі максимально точно ідентифікувати текстову інформацію. До них відносяться:

1. Попередня обробка – покращення якості зображення, усунення шумів, нормалізація кольорів для оптимальної роботи алгоритму.

2. Сегментація – поділ зображення на окремі символи, слова або рядки для детального аналізу тексту.

3. Витяг ознак – отримання ключових характеристик, які дозволяють розрізняти символи один від одного.

4. Класифікація – розпізнавання символів на основі алгоритмів машинного навчання або нейронних мереж.

5. Постобробка – перевірка і виправлення помилок на основі контексту для підвищення точності результату [14].

На рисунку 1.2 наведена загальна схема етапів процесу OCR, що демонструє взаємозв’язок між етапами обробки зображення та аналізу тексту.

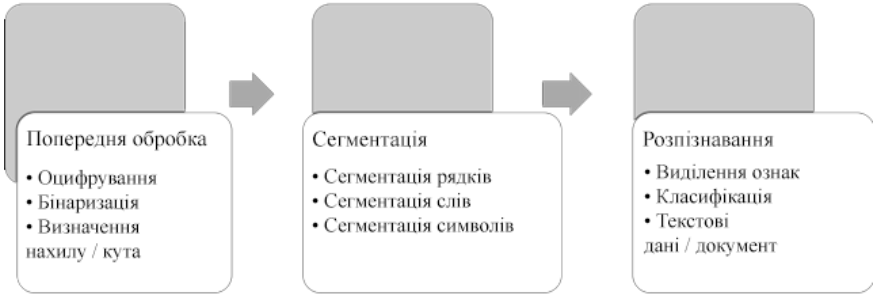


Рисунок 1.2 – Основні етапи OCR

Незважаючи на високу ефективність OCR, технологія має як переваги, так і певні обмеження [2]:

1. Швидкість обробки – OCR дозволяє швидко розпізнавати великі обсяги текстової інформації, що значно прискорює обробку.

2. Точність – автоматизоване розпізнавання знижує ризик виникнення певних помилок, характерних для ручного введення.

3. Гнучкість – OCR працює з різними типами документів, рукописним текстом та складними шрифтами.

4. Масштабованість – сучасні системи інтегруються у хмарні середовища, що дає змогу обробляти великі обсяги документі паралельно.

Водночас існують і обмеження:

1. Якість зображення – розмиті або низькоякісні зображення можуть значно погіршити точність розпізнавання.

2. Рукописний текст – хоча сучасні системи OCR працюють з рукописами, точність розпізнавання рукописного тексту залишається нижчою.

3. Мовні особливості – складні мовні конструкції або специфічні символи можуть потребувати додаткової обробки.

4. Обмеженість контексту – OCR працює із символами окремо, а іноді потрібне глибше розуміння текстового контексту.

Таким чином, незважаючи на зазначені обмеження, OCR залишається однією з ключових технологій для автоматизації текстових процесів.

Для подолання цих обмежень, розроблено низку удосконалених OCR-рішень, серед яких:

1. Tesseract OCR – один із найпопулярніших Open-Source OCR-двигунів, що підтримує понад 100 мов. Його використовують у багатьох дослідницьких та комерційний проєктах [3].

2. Google Cloud Vision API – хмарне рішення, що забезпечує точне розпізнавання тексту зображень, включно з рукописними документами, та підтримує понад 50 мов [4].

3. ABBYY FineReader – комерційне рішення, відоме своєю високою точністю при роботі з багатосторінковими та багатошаровими PDF-документами [5].

OCR забезпечує високу точність розпізнавання тексту та значно покращує процеси цифровізації документів, автоматизації введення даних.

Незважаючи на певні технічні обмеження, сучасні методи машинного навчання та нейромережеві алгоритми суттєво вдосконалюють OCR-системи, розширюючи їх можливість у різних сферах застосування.

1.3 Використання нейромереж та методів глибокого навчання для розпізнавання тексту

Застосування нейромереж та методів глибокого навчання суттєво підвищує точність OCR. Особливо ефективним є згорткові нейронні мережі (CNN), які використовують шари згортки для вилучення ознак зображення, та рекурентні нейронні мережі (RNN), що дозволяють враховувати послідовність символів при розпізнаванні [5].

CNN можуть ефективно виявляти особливості символів (краї, контури, форми), навіть коли зображення низької якості [5]. RNN, зокрема LTSM-мережі, ідеально підходять для аналізу послідовностей, дозволяючи враховувати контекст для коректного розпізнавання слів і речень [6].

На рисунку 1.3 наведено типову структуру CNN, що використовується для задач розпізнавання тексту [7].

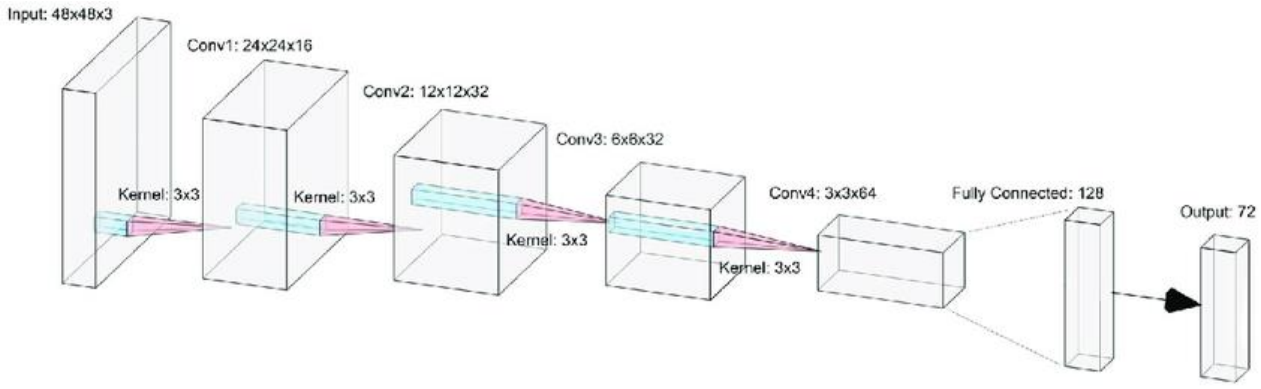


Рисунок 1.3 – Архітектура згорткової нейронної мережі CNN

Як було зазначено раніше, Tesseract OCR є одним із найпопулярніших інструментів для оптичного розпізнавання символів. Завдяки високій точності обробки тексту та відкритому коду, він широко використовується у наукових дослідженнях і комерційних проєктах. Це дозволяє розробникам легко адаптувати його до специфічних задач розпізнавання [7].

На рисунку 1.4 наведено приклад результату роботи Tesseract OCR, що демонструє процес розпізнавання тексту з цифрого зображення.

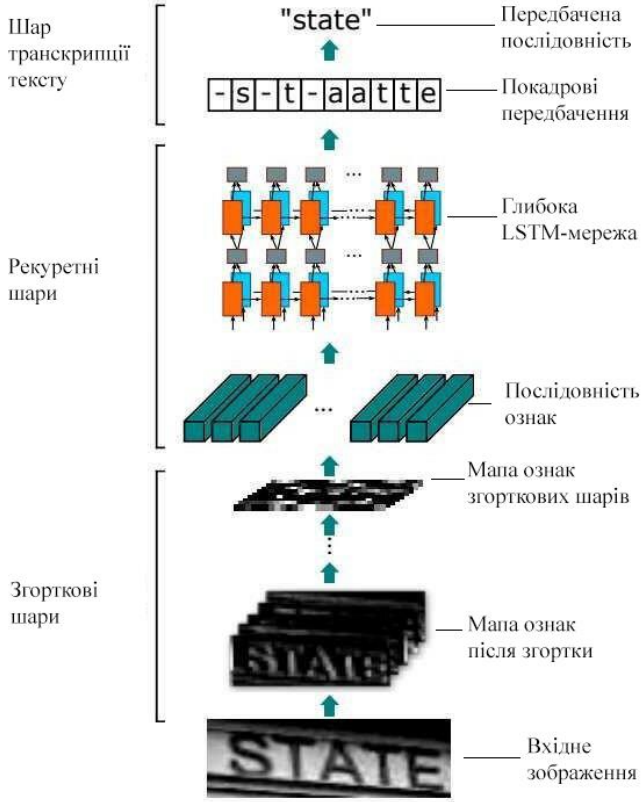


Рисунок 1.4 – Архітектура Tesseract OCR

Таким чином, використання сучасних нейромережевих технологій, зокрема згорткових (CNN) та рекурентних (LSTM) нейронних мереж, суттєво підвищує ефективність систем оптичного розпізнавання тексту. Завдяки здатності враховувати контекст, коригувати похибки, адаптуватися до змін освітлення, перекосів і спотворень, ці моделі значно покращують точність розпізнавання тексту навіть у складних умовах реального світу.

Незважаючи на можливі обмеження апаратного забезпечення, такі як ESP32-CAM, застосування оптимізованих моделей глибокого навчання дає змогу

створювати ефективні рішення машинного зору. У випадках нестачі якісних навчальних даних доцільно використовувати синтетичні або напівсинтетичні набори для попереднього навчання моделей, що покращує узагальнення та точність розпізнавання [11].

Ще одним важливим напрямом використання глибокого навчання у розпізнаванні тексту є методи самонавчання. Це особливо актуально в ситуаціях, коли доступ до великої кількості розмічених даних обмежений. Використовуючи такі методи, як псевдо-розмітка або напівконтрольоване навчання, можна навчати нейронні мережі на великих обсягах нерозмічених даних, що значно зменшує потребу в трудомісткому процесі ручного маркування.

Одним із потужних інструментів для OCR на основі нейромереж є платформа EasyOCR, яка використовує глибокі нейронні мережі для розпізнавання більше ніж 80 мов, включаючи українську. EasyOCR базується на поєднанні CNN для екстрації ознак і LSTM для послідовної обробки текстових рядків, що робить її дуже ефективною для задач багатомовного розпізнавання тексту.

У контексті розроблення власних систем розпізнавання тексту можна застосувати такі бібліотеки, як TensorFlow, PyTorch та Keras, які забезпечують потужні інструменти для побудови та навчання глибоких нейронних мереж. Для реалізації систем OCR необхідно створити ефективний пайплайн обробки даних, який включає етапи попередньої обробки зображення (наприклад, нормалізація яскравості, вирівнювання зображення, бінаризаця), побудову архітектури моделі, навчання на відповідній виібірці та подальше тестування на валідаційних даних [39].

Підсумовуючи, поєднання доступних апаратних рішень із сучасними методами глибокого навчання – включно з CNN, LSTM, трансформерами та самонавчанням – забезпечує побудову точних, гнучких і масштабованих OCR-систем, придатних для реального використання в IoT, автомомних модулях та веб-застосунках.

2 РОЗРОБЛЕННЯ МЕТОДУ РОЗПІЗНАВАННЯ ТЕКСТУ

2.1 Вибір апаратної платформи та аналіз її характеристик

Під час розроблення системи розпізнавання тексту вибір апаратної платформи є важливим етапом, оскільки він визначає її продуктивність, енергоефективність та можливості інтеграції з іншими технологіями. Від правильного вибору залежить точність обробки зображень, швидкість аналізу тексту та загальна ефективність роботи системи.

Вибір апаратного забезпечення здійснювався за такими критеріями:

1. Низьке енергоспоживання.

2. Компактність пристрою.

3. Наявність вбудованої камери високої чіткості.

4. Можливість передачі даних через Wi-Fi.

5. Підтримка доступних середовищ розробки.

Серед поширених апаратних рішень для реалізації подібних задач можна виділити наступні:

1. Raspberry Pi – потужний багатофункціональний міні комп’ютер, який підтримує повноцінну операційну систему та підтримує складні алгоритми машинного навчання. Водночас має великі розміри та вартість у порівнянні з мікроконтролерами.

2. Arduino з модулем камери – бюджетне рішення, яке дозволяє працювати з простими задачами розпізнавання тексту, але має обмежену продуктивність.

3. ESP32-CAM – компактний мікроконтролер з вбудованим модулем камери та бездротового зв’язку. Має досить низьке енергоспоживання, невеликі розміри, що робить його придатним для використання у вбудованих системах. Основна перевага – співвідношення функціональності до вартості.

Для вибору оптимальної платформи OCR важливо врахувати обчислювальну потужність, можливості підключення камер та інтеграцію бездротових технологій. У таблиці 2.1 наведено порівняння ключових параметрів.

Таблиця 2.1 – Порівняння апаратних платформ для реалізації OCR

Платформа	**Потужність**	**Підтримка камери**	**Wi-Fi, Bluetooth**	**Ціна**
Raspberry Pi	Висока	USB/CSI-камера	Через додаткові модулі	Від 4 000 до 6 000 грн.
Arduino	Низька	Через зовнішній модуль	Через додаткові модулі	Від 400 до 800 грн.
ESP32-CAM	Середня	Вбудована (OV2640)	Вбудовані	Від 200 до 500 грн.

З огляду на потреби, для реалізації даної системи обрано ESP32-CAM як найбільш збалансовану платформу для вбудованих систем оптичного розпізнавання тексту [9].

ESP32-CAM побудована на базі мікроконтролера ESP32, що має двоядерну архітектуру та підтримує ефективне управління живленням. Завдяки цьому вона активно використовується у системах відеоспостереження, аналізу зображень та автоматизації процесів. Основні характеристики ESP32-CAM наведені в таблиці 2.2.

Таблиця 2.2 – Основні характеристики платформи ESP32-CAM [8]

Параметр	**Значення**
Мікроконтролер	32-бітний двоядерний ESP32 (до 240 МГц)
Оперативна пам’ять	520 КБ SRAM + 4 МБ PSRAM
Камера	OV2640 (до 1600x1020 пікселів)
Підключення	Wi-Fi 802.11 b/g/n, Bluetooth 4.2
Порти введення/виведення	GPIO, UART, SPI, I2C, PWM, ADC, DAC

Оскільки ESP32-CAM є компактною платформою, важливо розглянути її фізичну структуру. На рисунку 2.1 зображено зовнішній вигляд модуля, а на рисунку 2.2 розташування його виводів.

Рисунок 2.1 – Зовнішній вигляд модуля ESP32-CAM

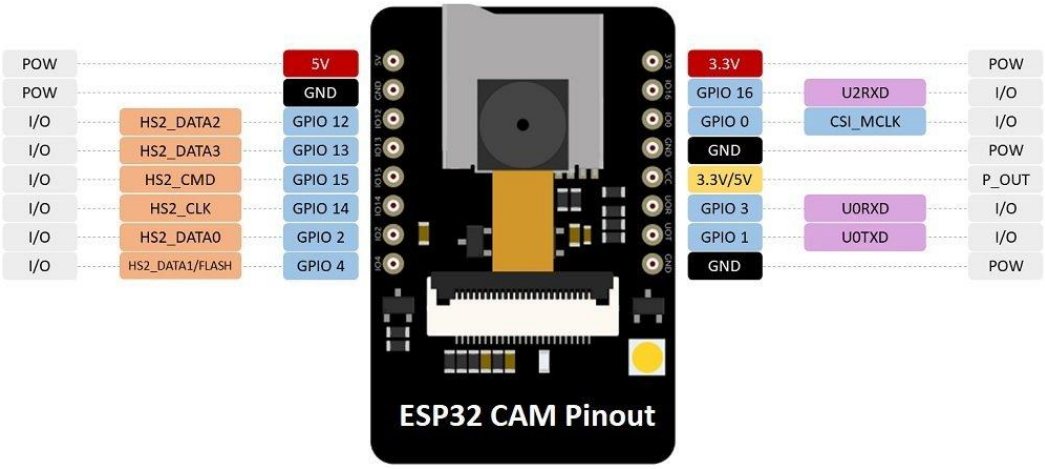


Рисунок 2.2 – Схема розташування виводів ESP32-CAM

Обґрунтування вибору ESP32-CAM базується на компромісі між вартістю та функціональністю. Цей модуль забезпечує базову локальну обробку зображення і передачу даних, знижуючи навантаження на серверні ресурси та інфраструктуру. Завдяки наявності Wi-Fi ESP32-CAM дозволяє легко інтегруватися в локальні або глобальні мережі без додаткового обладнання.

2.2 Вибір програмних технологій для реалізації системи

На відміну від апаратної частини, яка забезпечує збір та первинну обробку даних, програмне забезпечення відіграє ключову роль у аналізі зображень, розпізнаванні тексту та інтеграції із зовнішніми сервісами.

Розробка сучасних систем, зокрема машинного зору вимагає використання сучасних, продуктивних і масштабованих технологій. Важливими критеріями вибору є наявність бібліотек для обробки зображень, засобів інтеграції клієнтської та серверної частини, підтримка API, а також гнучність і швидкодія при роботі з різними форматами даних.

Оскільки система розпізнавання тексту повинна включати обробку зображень, аналіз символів, взаємодію з користувачем через веб-інтерфейс і передачу результатів, було сформовано технологічний стек, що охоплює як серверну, так і клієнтську частину системи розпізнавання тексту [42].

У межах цього дослідження для реалізації серверної частини використано мову програмування Python, фреймворк FastAPI, бібліотеки OpenCV та Tesseract OCR.

Мова Python була обрана завдяки простому синтаксису, широкому набору бібліотек для обробки зображень і машинного навчання. Серед найпопулярніших бібліотек, які підтримує Python можна виділити – OpenCV, NumPy, PyTorch, TensorFlow, Tesseract та інші.

Фреймворк FastAPI є сучасним рішенням для побудови високопродуктивних REST API. Завдяки асинхронному підходу він забезпечує швидкість обробки запитів, що робить його оптимальним вибором для розробки.

Для попередньої обробки зображень використовується OpenCV – одна з найпотужніших бібліотек у сфері машинного зору. Вона забезпечує широкий спектр функцій: масштабування, фільтрацію, бінаризацію, виявлення контурів тощо.

Для розпізнавання тексту використовується Tesseract OCR – оптичний рушій розпізнавання тексту з відкритим кодом, який підтримує понад 100 мов, можливість навчання на спеціальних шрифтах, роботу з кирилицею та адаптацію до спотворених зображень [43].

Клієнтська частина системи реалізована на основі Vue.js – реактивного фреймворку, який дозволяє створити адаптивний веб-інтерфейс, оптимізований для різних пристроїв і забезпечити користувачу зручну взаємодію.

Для більшої наочності вибору технологій на рисунку 2.3 наведено схему розподілу технологічного стеку за ролями у системі розпізнавання тексту. Умовні відсотки відображають внесок кожного інструменту в реалізацію загальної архітектури проєкту.

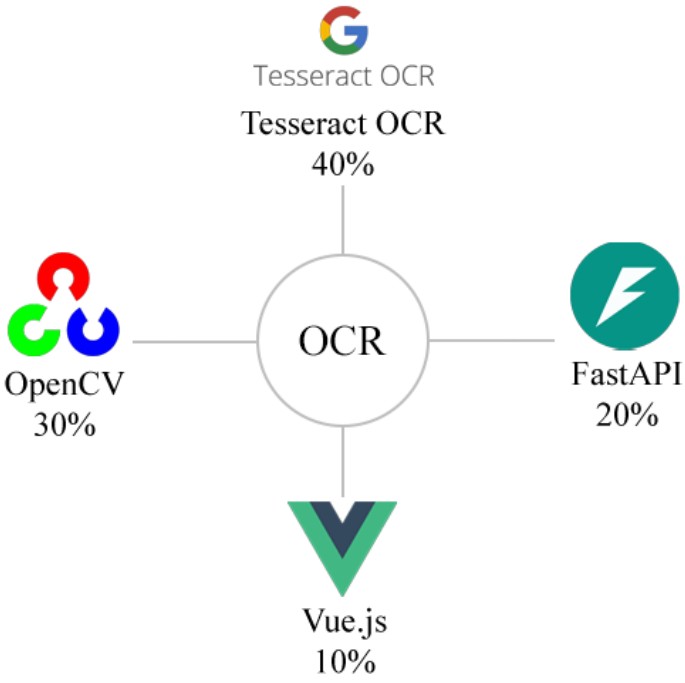


Рисунок 2.3 – Розподіл технологій у системі розпізнавання тексту

Потік процесів у системі OCR від захоплення зображення до збереження результатів зображено на рисунку 2.4.

Рисунок 2.4 – Потік процесів у системі від захоплення зображення до збереження результатів

Таким чином, побудова системи розпізнавання тексту базується на взаємодії між апаратною частиною, серверною логікою та клієнтським інтерфейсом. Поділ на окремі рівні забезпечує модульність архітектури, спрощує її розширення, а також гарантує ефективну обробку зображень у режимі реального часу.

2.3 Структурна та функціональна схема запропонованого методу

Для реалізації системи необхідно побудувати узгоджену програмно-апаратну архітектуру системи, яка забезпечує ефективний обмін даними між модулями, надійну обробку вхідної інформації та зручну взаємодію з користувачем. Запропонована система складається з трьох основних компонентів: апаратного модуля ESP32-CAM, серверної частини та клієнтського веб-інтерфейсу.

Структурна схема системи, представлена на рисунку 2.5, демонструє основні взаємозв'язки між компонентами та напрямки передачі даних. Основні складові:

1. Апаратний модуль ESP32-CAM, який формує JPEG-файл і передає його на сервер через Wi-Fi.

2. Сервер обробки зображень на основі FastAPI, який виконує попередню обробку через OpenCV та розпізнавання тексту за допомогою Tesseract OCR.

3. Клієнтський веб-інтерфейс на базі Vue.js для взаємодії користувача із системою.

Рисунок 2.5 – Структурна схема системи розпізнавання тексту

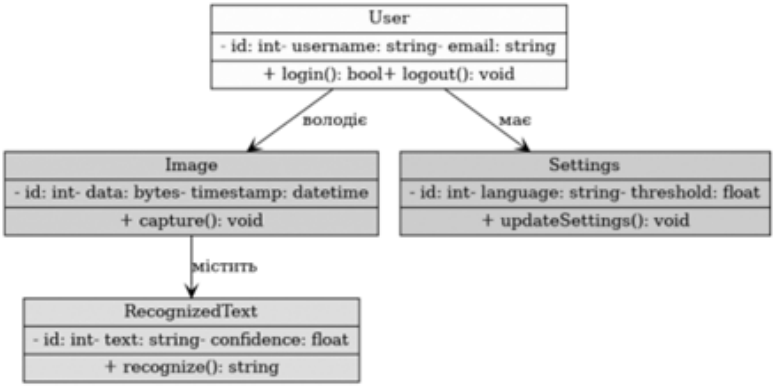


Рисунок 2.6 – Структура даних для обробки та збереження результатів

Рисунок 2.7 – Процес обробки зображення та розпізнавання тексту

Функціональна схема системи (рисунок 2.8) описує типовий сценарій обробки даних, що складається з шести ключових етапів:

1. Захоплення зображення камерою ESP32-CAM та форматування JPEG-файлу.

2. Передача зображення на серверну частину.

3. Попередня обробка зображення через OpenCV (фільтрація, нормалізація)

4. Розпізнавання тексту за допомогою бібліотеки Tesseract OCR.

5. Форматування та передача результату у клієнтський інтерфейс.

6. Відображення тексту у веб-інтерфейсі.

Рисунок 2.8 – Функціональна схема взаємодії компонентів

Апаратна частина базується на використанні модуля ESP32-CAM, який здіснює захоплення зображень із текстовою інформацією та забезпечує їх бездротову передачу на сервер.

Серверна частина реалізована на базі FastAPI та відповідає за обробку отриманих зображень, їх підготовку до розпізнавання та виконання OCR-процесу за допомогою бібліотеки Tesseract OCR. Сервер забезпечує асинхронну обробку запитів, що дозволяє досягти високої швидкодії та ефективного використання ресурсів, особливо при обробці великого обсягу інформації.

Попередня обробка зображень здійснюється за допомогою бібліотеки OpenCV, що дозволяє покращити якість вхідних даних за рахунок фільтрації шумів, корекції контрастності та бінаризації, що безпосередньо впливає на підвищення точності розпізнавання тексту [38].

Клієнтська частина побудована із застосуванням фреймворку Vue.js, що забезпечує створення сучасного, адаптивного, інтуїтивно зрозумілого інтерфейсу користувача. Завдяки SPA-архітектурі інтерфейс має високу швидкість завантаження та реагування, що покращує загальний користувацький досвід.

Усі компоненти системи взаємодіють через стандартизовані REST API, що полегшує інтеграцію, забезпечує прозорість обміну даними та дозволяє легко масштабувати рішення в майбутньому.

Розроблена архітектура системи відзначається балансом між складністю реалізації та функціональністю. Вона забезпечує ефективне розпізнавання тексту, мінімізацію затримок у передачі та обробці даних, високу надійність і безпеку, що в сукупності гарантує відповідність розробленої системи сучасним вимогам та очікуванням кінцевих користувачів. Отримані результати стали базисом для подальших етапів реалізації, інтеграції компонентів і тестування розробленої системи.

На першому етапі інформація у вигляді зображення передається із сенсора камери до модуля обробки. Далі проводиться попередня обробка зображення: фільтрація шумів, нормалізація освітлення, покращення контрасту за допомогою бібліотеки OpenCV. Після цього зображення передається до модуля розпізнавання символів на основі Tesseract OCR. Отриманий текст передається серверу для виведення його в інтерфейсі користувача.

Для моделювання потоків і структур системи використовуються різні типи наочних діаграм (рисунок 2.9, 2.10, 2.11), кожна з яких дозволяє візуалізувати певні аспекти взаємодії системи.

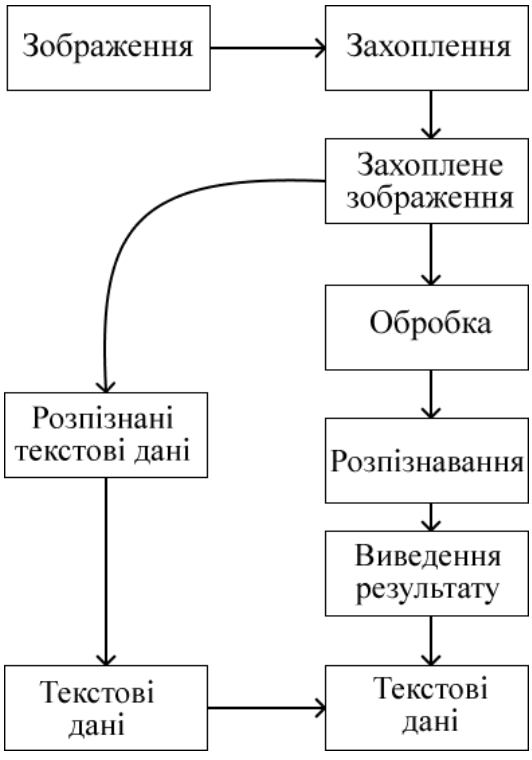


Рисунок 2.9 – Життєвий цикл обробки зображення

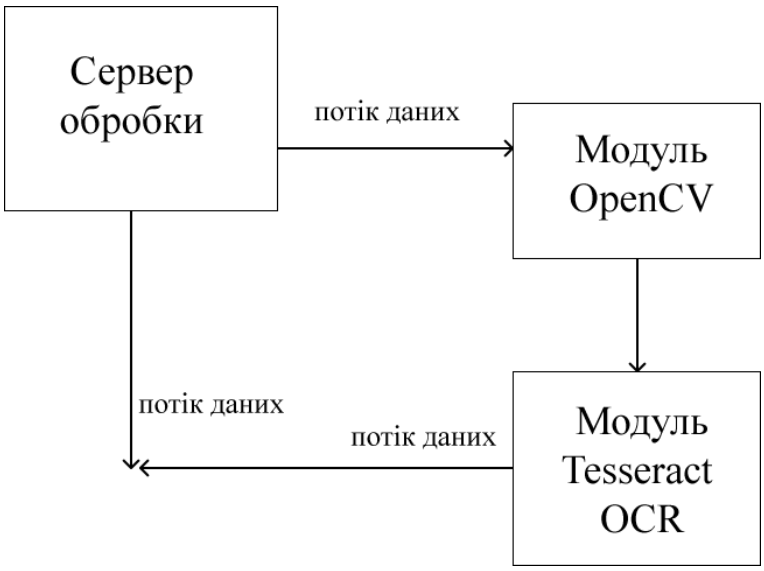


Рисунок 2.10 – Життєвий цикл обробки зображення

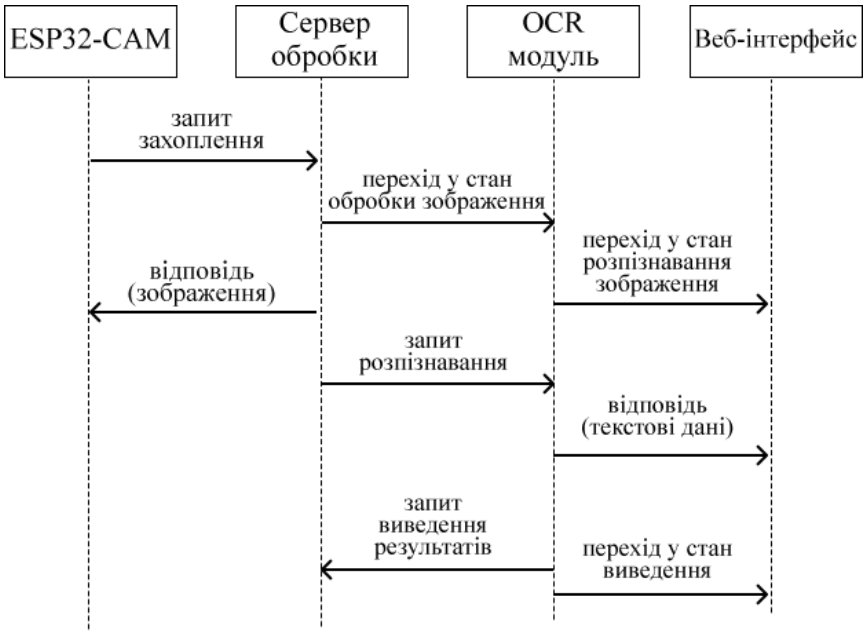


Рисунок 2.11 – Послідовність обробки запиту на розпізнавання тексту

Моделювання потоків інформації та структур дає змогу виявити потенційні точки оптимізації системи. Наприклад, буферизація зображень перед обробкою дозволяє уникнути втрат при навантаженнях, а використання асинхронної обробки запитів підвищує пропускну зданість сервера.

Таке детальне моделюваня також дозволяє ефективно планувати розгортання системи, вибирати оптимальні технічні засоби для її реалізації та забезпечувати гнучкість і масштабованість рішення відповідно до реальних умов експлуатації.

2.4 Алгоритм обробки та розпізнавання зображень

Розроблення алгоритму для обробки зображень і подальшого розпізнавання тексту є однією з ключових задач при створенні ефективної системи розпізнавання тексту. Від якості попередньої обробки залежить точність подальшого аналізу. Тому важливо побудувати послідовність алгоритмів, яка забезпечить оптимальну підготовку даних для текстового розпізнавання, мінімізуючи втрати інформації та зменшуючи рівень шумів.

Побудова алгоритмів включає кілька етапів:

1. Попередня обробка зображення для підвищення якості.
2. Локалізація текстових блоків.
3. Сегментація тексту на символи або слова.
4. Розпізнавання символів із використанням OCR-алгоритмів.
5. Постобробка результатів для підвищення точності.

На кожному етапі застосовуються спеціалізовані методи та технології, які забезпечують необхідні параметри обробки, враховуючи обмеження обчислювальних ресурсів ESP32-CAM та серверної інфраструктури.

Попередня обробка зображення починається з фільтрації шумів. Найчастіше для цієї задачі застосовується гаусова фільтрація, яка зменшує вплив дрібних артефактів і незначних спотворень. Далі виконується покращення контрасту за допомогою гістограмного вирівнювання, що дозволяє підвищити чіткість текстових символів (рисунок 2.12).

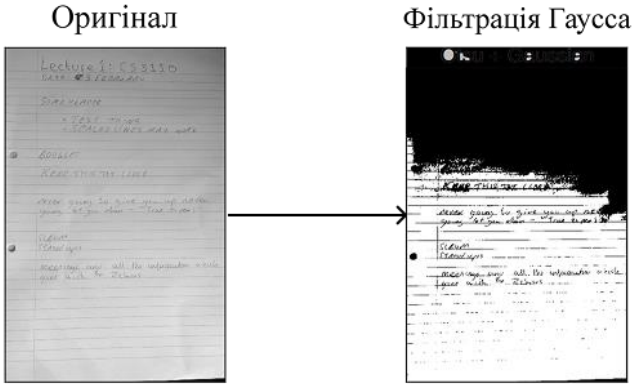


Рисунок 2.12 – Приклад попередньої обробки гаусовою фільтрацією

Для бінаризації зображення використовується метод Отсу, який автоматично визначає оптимальний поріг для розділення фону та текст (рисунок 2.13). Після бінаризації застосовується морфологічна обробка: операції розширення і ерозії дозволяють видалити незначні дефекти і об’єднати фрагментовані символи.

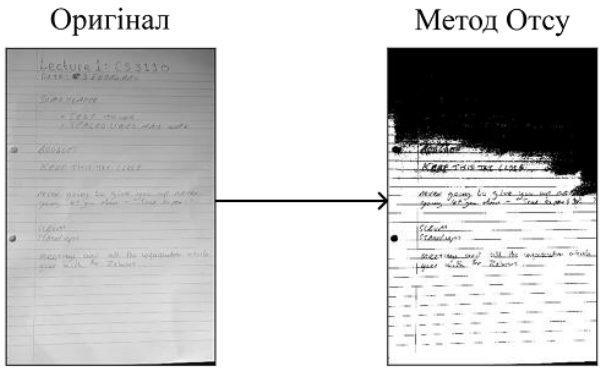


Рисунок 2.13 – Приклад попередньої обробки за методом Отсу

На етапі локалізації текстових блоків використовуються методи виявлення контурів та алгоритми на основі згорткових нейронних мереж (наприклад, EAST Detector), що забезпечує точне визначення меж текстових областей навіть при складних фонах або спотвореннях (рисунок 2.14) [49].

Рисунок 2.14 – Приклад роботи EAST Detector

Сегментація тексту здійснюється за допомогою горизонтальних та вертикальних проєкцій, що дозволяє ефективно розділити текст на рядки, слова та символи.

Розпізнавання символів виконується OCR-движком Tesseract, який підтримує багатомовне розпізнавання, зокрема кирилицю, латиницю та інші алфавіти.

Для підвищення точності застосовується попереднє навчання моделі на специфічних шрифтах і форматах документів. В таблиці 2.9 представлено алгоритмічні етапи основних етапів.

Таблиця 2.9 Алгоритмічні етапи обробки зображень і розпізнавання тексту

Етап	**Алгоритм**	**Опис процесу**	**Інструменти**
Фільтрація шумів	Гаусова фільтрація	Зменшення шумів	OpenCV
Покращення контрасту	Гістограмне вирівнювання	Підвищення чіткості	OpenCV
Бінаризація	Метод Отсу	Розподілення фону і тексту	OpenCV
Морфологічна обробка	Розширення, ерозія	Усунення дефектів	OpenCV
Локалізація тексту	EAST Detector	Виявлення текстових блоків	TensorFlow, OpenCV
Сегментація символів	Горизонтальні/вертикальні проєкції	Розділення на слова/символи	OpenCV
Розпізнавання тексту	Tesseract OCR	Ідентифікація символів	Tesseract
Постобробка	Словникова перевірка	Виправлення помилок	NLTK, PySpellChecker

На рисунку 2.15 наочно представлено алгоритмічний процес обробки зображення, що включає всі основні етапи – від фільтрації шумів до постобробки розпізнаваного тексту.

Рисунок 2.15 – Алгоритмічний процес обробки зображення

Алгоритм обробки зображення передбачає певні оптимізації для зменшення обчислювальних витрат. Зокрема, на етапі обробки зображення розмір може бути зменшений для прискорення роботи без значної втрати якості тексту. Крім того, параметри гаусової фільтрації та морфологічних операцій можуть бути налаштовані динамічно залежно від властивостей зображення.

Модуль Tesseract працює на основі глибокого навчання та підтримує як розпізнавання окремих символів, так і цілих рядків тексту. Для підвищення точності використовуються попередньо навчені моделі, що дозволяє уникнути потреби у значних обчислювальних ресурсах для повторного навчання [28].

Постобробка тексту є важливим етапом для виправлення типових помилок розпізнавання, таких як плутанина між схожими символами («O» та «0», «l» та «1») та покращення структури тексту (розбиття на абзаци, відновлення форматування). Основні методи оптимизації обробки наведено в таблиці 2.10.

Таблиця 2.10 Основні методи оптимізації обробки зображень та OCR

Метод	**Призначення**	**Інструмент реалізації**	**Переваги**	**Недоліки**
Зменшення розміру зображення	Прискорення обробки	OpenCV resize	Зниження обчислювальних витрат	Можлива втрата дрібних деталей
Динамічна фільтрація	Оптимізація гаусової фільтрації	OpenCV adaptive methods	Автоматичне налаштування	Потреба в оцінці параметрів
Використання попередньо навчаних моделей	Підвищення точності	Tesseract pretrained models	Висока точність без навчання	Потреба в адаптації моделей
Адаптивна морфологія	Підвищення якості структури	OpenCV morphologyEx	Видалення шумів і артефактів	Потребує підбору параметрів
Словникова перевірка	Виправлення помилок	NLTK, PySpellChecker	Підвищення точності результату	Обмеження словинка

На рисунку 2.16 наочно представлено взаємозв’язок методів попередньої обробки та OCR.

Рисунок 2.16 – Взаємозв’язок методів попередньої обробки та OCR

Таким чином, застосування розроблених алгоритмічних рішень дозволяє створити ефективну систему обробки зображень та розпізнавання тексту, що забезпечує високу якість результату при оптимальному використанні ресурсів. Побудовані алгоритми можуть бути адаптовані під різні умови експлуатації, що є важливим для побудови універсальної платформи машинного зору на базі ESP32-CAM.

Для системи апаратна частина представлена модулем ESP32-CAM, який виконує функцію захоплення зображень, тоді як програмна частина реалізована на базі серверної частини (FastAPI, OpenCV, Tesseract OCR) та клієнтського веб-інтерфейсу (Vue.js) [32].

Апаратна частина відповідає за реєстрацію зображень об’єктів, конвертацію їх у формат JPEG, початкове стиснення та передачу даних до серверної частини через бездротову мережу Wi-Fi. Програмна частина отримує зображення, виконує попередню обробку для покращення якості, проводить розпізнавання тексту та передає результати на клієнтський інтерфейс.

Ключовими етапами взаємодії є:

1. Ініціація зйомки через ESP32-CAM.

2. Форматування HTTP-запиту зі вкладеним зображенням.

3. Обробка запиту сервером FastAPI.

4. Передача зображення для попередньої обробки OpenCV.

5.Передача обробленого зображення на модуль Tesseract для розпізнавання.

6. Формування відповіді у форматі JSON з текстовим результатом.

7. Виведення тексту на веб-інтерфейсі користувача.

Побудова схеми взаємодії дозволяє визначити вузькі місця системи, можливі затримки при передачі даних і забезпечити оптимізацію обміну даними. Зображення фізичного розгортання апаратної та програмної частини зображено на рисунку 2.17.

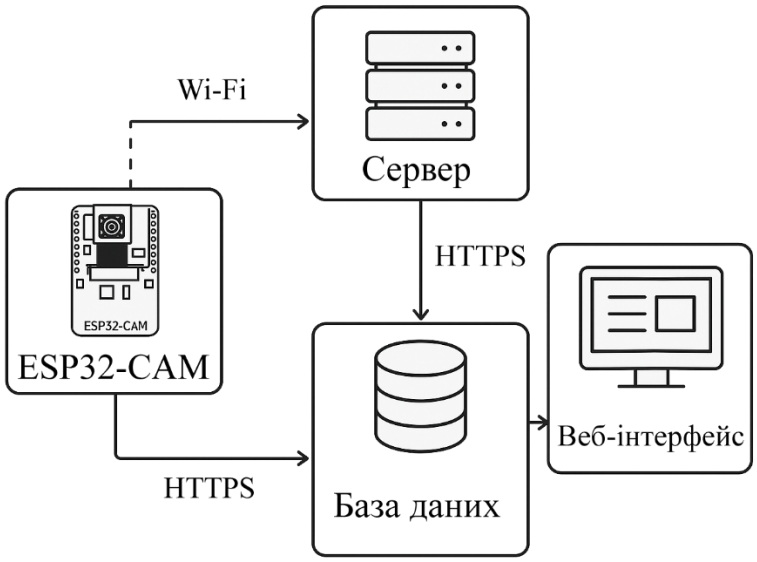


Рисунок 2.17 – Фізичне розташування апаратної та програмної частини

Архітектура взаємодії системи побудована на принципах асинхронності для забезпечення високої пропускної здатності та мінімізації затримок. Апаратна частина ESP32-CAM, будучи ресурсно обмеженою, виконує тільки базові функції передаючи обробку даних на локальний сервер.

Передача зображень здійснюється у форматі JPEG через HTTP POST-запит до API, що дозволяє знизити навантаження на мережу та сервер.

Після завершення обробки результати передаються клієнту у форматі JSON, що забезпечує гнучкість і зручність інтеграції з будь-якими веб-інтерфейсами.

Архітектура користувацького веб-інтерфейсу системи базується на сучасних підходах до розробки фронтенд-застосунків. У якості основного технологічного рішення обрано Vue.js – прогресивний JavaScript-фреймворк, що дозволяє реалізувати реактивний підхід до розробки, забезпечити високу швидкість взаємодії з користувачем і легкість масштабування інтерфейсу [50].

Основні принципи, закладені в архітектуру інтерфейсу:

1. Модульність. Інтерфейс поділений на окремі компоненти з чіткою визначеною відповідальністю.

2. Реактивність. Зміни у даних автоматично відображаються в інтерфейсі без потреби в оновленні сторінки.

3. Односторінкова архітектура (SPA). Забезпечує швидку навігацію без перезавантаження сторінок.

4. Відповідність принципам мобільної адаптивності.

На рисунку 2.18 зображено компонентну структуру веб-інтерфейсу, яка відповідає зазначеним принципам.

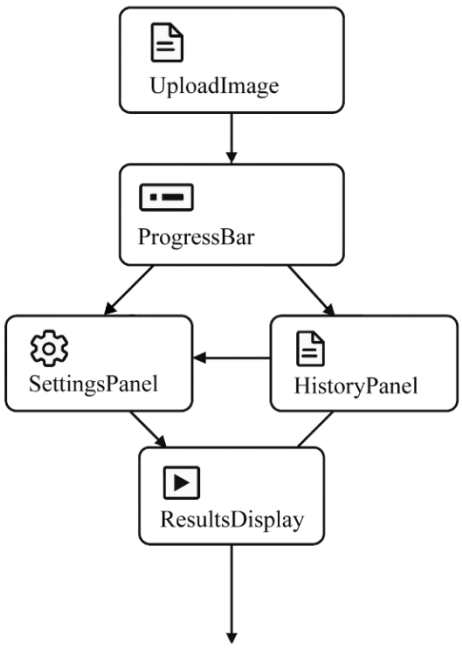


Рисунок 2.18 – Компонентна структура веб-інтерфейсу

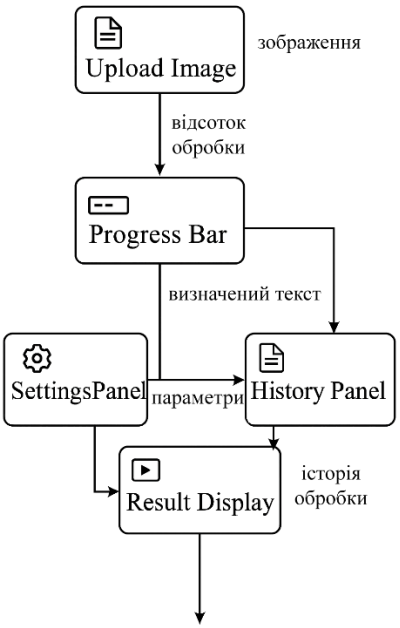


Рисунок 2.19 – Функціональні блоки інтерфейсу користувача

На рисунку 2.19 зображено внутрішню структуру функціональних блоків і потоки даних між ними під час роботи користувача з інтерфейсом.

На наступному рисунку 2.20 представлено детальний опис процесу взаємодії користувача з інтерфейсом, включаючи послідовність дій, комунікація між компонентами та обробку даних.

Рисунок 2.20 – Процес взаємодії з користувацьким інтерфейсом

Таким чином, архітектура користувацького веб-інтерфейсу розробленої системи базується на сучасних підходах побудови SPA-додатків з орієнтацію на високу реактивність, модульність та доступність. Обрані технології забезпечують масштабованість рішення, дозволяють швидко розширювати функціонал та підтримувати високий рівень користувацького досвіду навіть при збільшенні навантаження.

3 РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ ТА АНАЛІЗ РЕЗУЛЬТАТІВ ЇЇ РОБОТИ

3.1 Реалізація програмного забезпечення для модуля ESP32-CAM

Реалізація програмного забезпечення апаратної частини системи починається з налаштування основного елементу – модуля ESP32-CAM. Даний модуль поєднує в собі можливості роботи з камерою достатньо високої роздільної здатності та бездротової передачі даних через Wi-Fi.

Робота апаратної частини передбачає налаштування камери, захоплення зображення, обробку первинних даних та передачу зображень на серверну частину системи для подальшого розпізнавання тексту. Система також передбачає використання зовнішніх елементів, таких як стабілізоване джерело живлення та елементи кріплення камери для стабільної зйомки [12].

На рисунках 3.1, 3.2 відображено схему розгортання апаратної частини системи та структуру апаратної платформи ESP32-CAM.

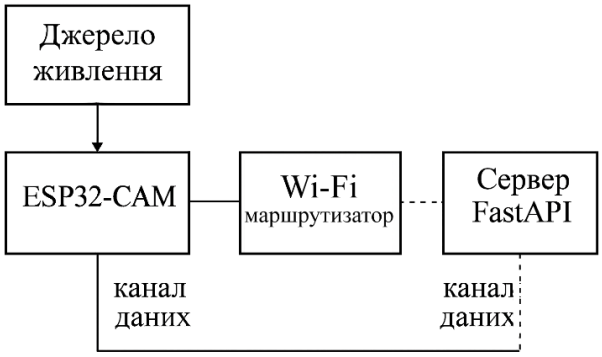


Рисунок 3.1 – Розгортання апаратної частини системи на базі ESP32-CAM

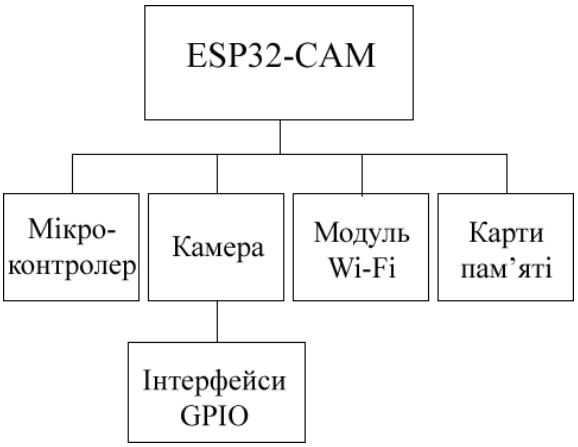


Рисунок 3.2 – Структура апаратної платформи ESP32-CAM

Для забезпечення надійності роботи апаратної частини були враховані наступні аспекти:

1. Використання стабілізованого джерела живлення 5В (USB підключення до комп’ютера).

2. Захист від перевантажень за допомогою запобіжників та конденсаторів згладжування.

3. Фіксація камери в нерухомому положенні для мінімізації спотворень.

Прошивка ESP32-CAM здійснюється через UART-з’єднання за допомогою зовнішнього програматора на базі USB-TTL конвертера. У якості середовища розробки обрано Arduino IDE через простоту налаштувань та підтримку великої кількості бібліотек.

Нижче наведено етапи процесу прошивки ESP32-CAM, які зображено на рисунках 3.3-3.7:

1. Підключення USB-TTL конвертера до ESP32-CAM.

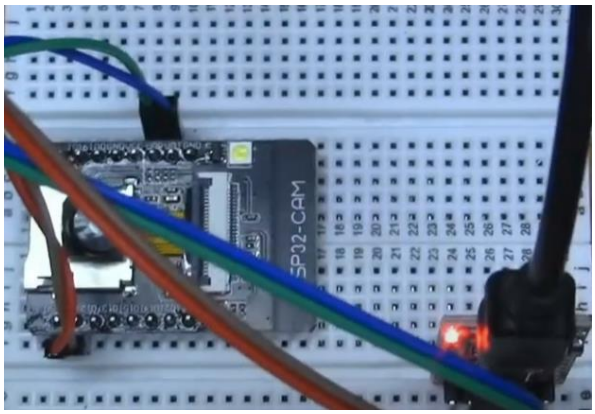


Рисунок 3.3 – Підключення USB-TTL конвертера до ESP32-CAM

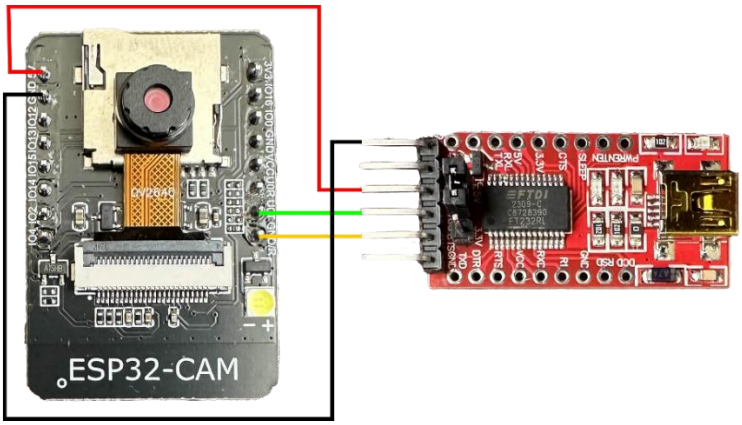


Рисунок 3.4 – Схема підключення ESP32-CAM до USB-TTL конвертера FTDI

2. Встановлення бібліотеку для роботи з ESP32.

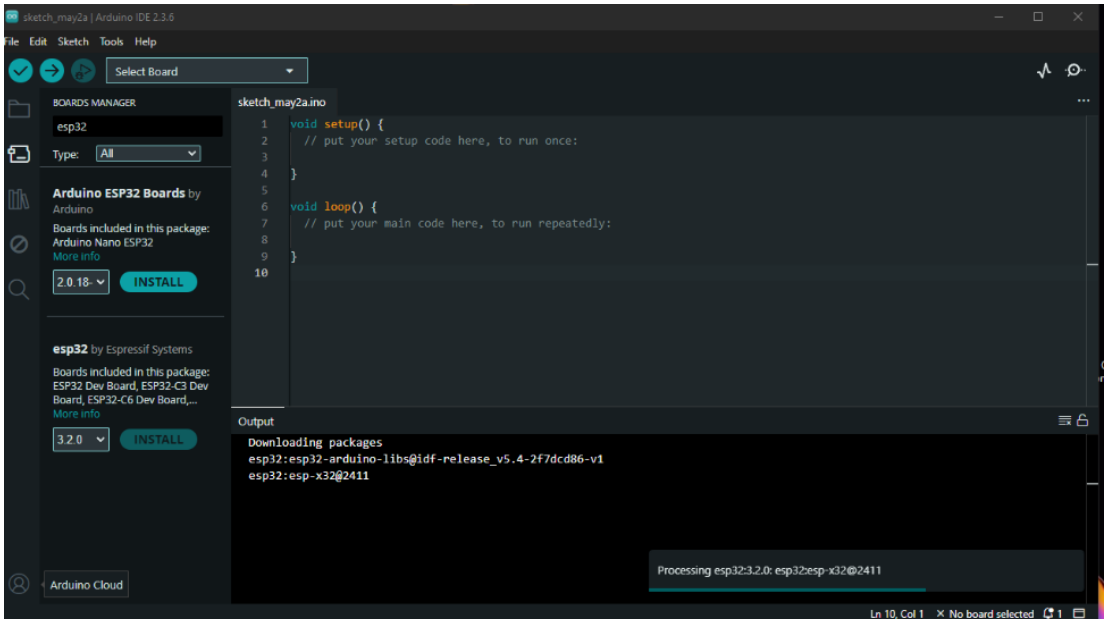


Рисунок 3.5 – Процес встановлення бібліотеки ESP32

3. Завантаження прошивки на ESP32-CAM.

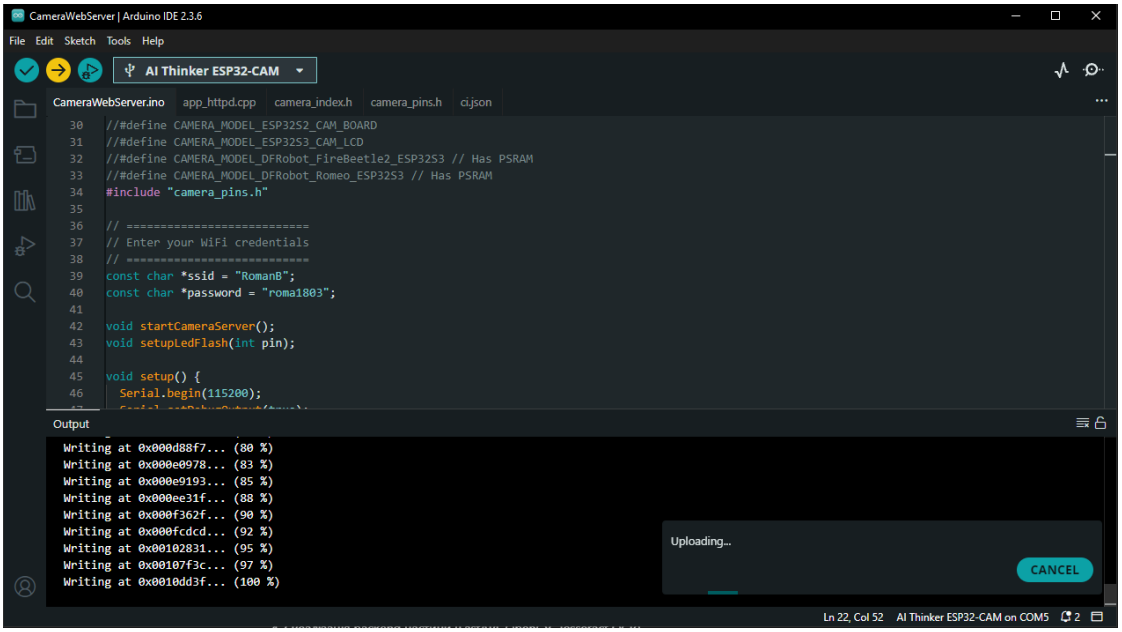


Рисунок 3.6 – Процес завантаження прошивки на ESP32-CAM

4. Відображення стандартного інтерфейсу після успішної прошивки на локальному веб-сервері.

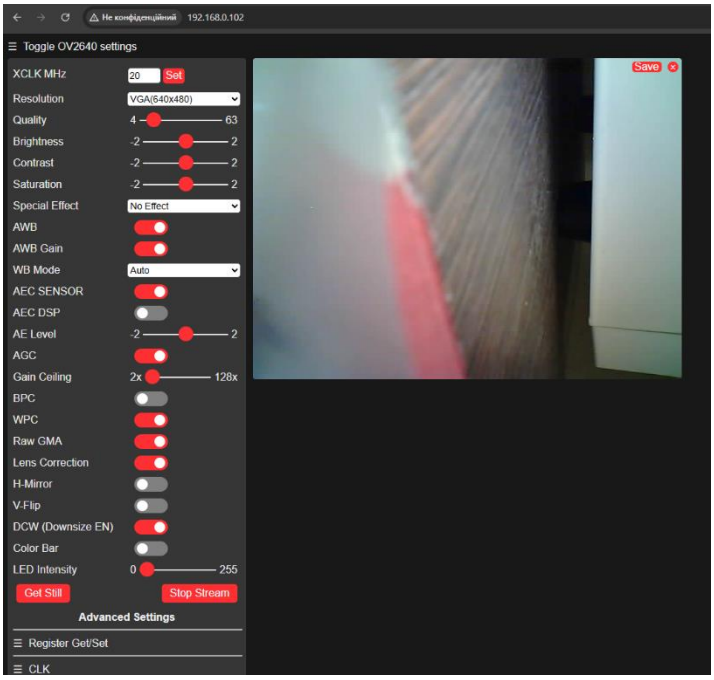


Рисунок 3.7 – Веб-інтерфейс із зображенням та налаштуваннями камери

Після успішної прошивки необхідно зробити програмну реалізацію, яка буде включати налаштування камери (якість зйомки, роздільна здатність), ініціалізацію Wi-Fi з’єднання, створення сервера на пристрої для надсилання знімків на віддалений сервер через POST-запити.

У рамках розробки було створено власний програмний код для роботи з камерою. Основні функції реалізованого коду:

1. Налаштування параметрів камери через API CameraWebServer.

2. Підключення до Wi-Fi з використанням попередньо визначених SSID і пароля.

3. Організація буферизації зображень для покращення якості передачі даних.

4. Підготовка зображень у форматі JPEG для передачі.

5. Передача даних через протокол HTTP.

Таким чином, апаратна частина системи забезпечує необхідний рівень функціональності для зйомки зображень, їх первинної обробки та бездротової передачі на серверну частину системи розпізнавання тексту.

3.2 Реалізація backend-частини системи

Backend частина системи відіграє ключову роль у забезпеченні функціональності, оскільки відповідає за отримання, обробку та аналіз зображень, а також передачу отриманих результатів до клієнта. Даний компонент реалізовано на Python із застосуванням FastAPI, що дозволяє створити ефективний REST API з високою продуктивністю.

1. FastAPI – забезпечує обробку запитів та взаємодію між компонентами.

2. OpenCV – виконує попередню обробку зображень для покращення якості розпізнавання.

3. Tesseract OCR – застосовується для виконання оптичного розпізнавання символів із отриманих зображень.

Процес обробки запитів на сервері відбувається наступним чином і зображений на рисунку 3.8:

1. Прийом та збереження зображення, отриманого з ESP32-CAM.

2. Попередня обробка (конвертація в градації сірого, бінаризація, фільтрація шумів).

3. Виконання OCR-аналізу за допомогою Tesseract.

4. Повернення результатів розпізнавання у вигляді текстових даних.

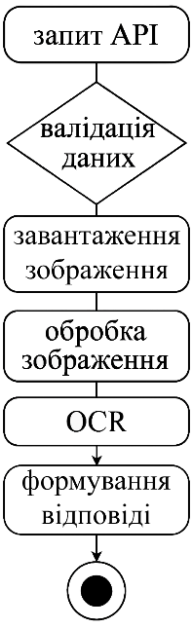


Рисунок 3.8 – Процес обробки запиту на сервері

Для обробки запитів використано FastAPI-роути з типами запитів POST і GET, основні з яких подано в таблиці 3.1.

Таблиця 3.1 – Основні роути API та їх функціональність

Метод	**Роут**	**Опис**	**Вхідні дані**	**Вихідні дані**
POST	/upload	Завантаження зображення на сервер	JPEG, PNG	Назва збереженого файлу
POST	/settings	Зміна налаштувань	JSON-параметри	Статус оновлення
GET	/capture	Запит на отримання зображення з ESP32-CAM	-	JSON-відповідь із зображенням
GET	/images	Отримання списку всіх збережених зображень	-	Список файлів
GET	/images/{filename}	Завантаження конкретного зображення	Назва файлу	Зображення у форматі JPEG
GET	/settings	Отримання налаштувань	-	Актуальні налаштування у форматі JSON

Особливістю реалізації є інтеграція модуля обробки зображень на базі OpenCV. Після прийому зображення сервер викликає серію фільтрів: шумозаглушення, нормалізацію контрасту, бінаризація методом Отсу.

Модуль Tesseract, який використовується у системі, має можливість роботи з кастомними моделями, що дозволяє налаштовувати розпізнавання під специфічні задачі.

Бібліотека Uvicorn була використана для створення локального сервера, який забезпечує роботу для FastAPI та всіх модулей.

У якості середовища для розробки Backend-частини було обрано Vistual Studio Code, яке є одним з найзручніших інструментів для написання коду завдяки широкій підтримці розширень та інтеграції із середовищем розробки Python.

Нижче наведено етапи створення Backend частини, які зображено на рисунках 3.9 – 3.13:

1. Створення папкової структури.

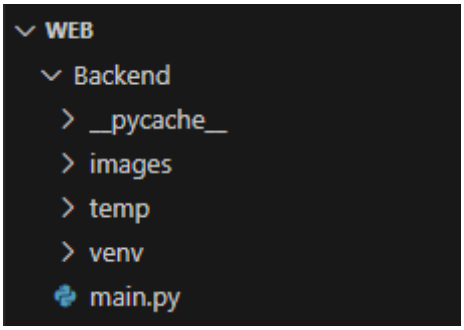


Рисунок 3.9 – Папкова структура Backend

2. Встановлення всіх необхідних компонентів за допомогою команди в терміналі (pip install fastapi uvicorn opencv-python pytesseract python-multipart).

```
PS C:\Users\boyar\Desktop\Дипломна\Web> pip install fastapi uvicorn opencv-python pytesseract python-multipart
Requirement already satisfied: fastapi in c:\users\boyar\desktop\дипломна\web\backend\venv\lib\site-packages (0.115.12)
Requirement already satisfied: uvicorn in c:\users\boyar\desktop\дипломна\web\backend\venv\lib\site-packages (0.34.2)
Requirement already satisfied: opencv-python in c:\users\boyar\desktop\дипломна\web\backend\venv\lib\site-packages (4.11.0.86)
Requirement already satisfied: pytesseract in c:\users\boyar\desktop\дипломна\web\backend\venv\lib\site-packages (0.3.13)
Requirement already satisfied: python-multipart in c:\users\boyar\desktop\дипломна\web\backend\venv\lib\site-packages (0.0.20)
Requirement already satisfied: starlette<0.47.0,>=0.40.0 in c:\users\boyar\desktop\дипломна\web\backend\venv\lib\site-packages (from fastapi) (0.46.2)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,!=2.0.0,!=2.0.1,!=2.1.0,<3.0.0,>=1.7.4 in c:\users\boyar\desktop\дипломна\web\backend\venv\lib\site
Requirement already satisfied: typing-extensions>=4.8.0 in c:\users\boyar\desktop\дипломна\web\backend\venv\lib\site-packages (from fastapi) (4.13.2)
Requirement already satisfied: annotated-types>=0.6.0 in c:\users\boyar\desktop\дипломна\web\backend\venv\lib\site-packages (from pydantic!=1.8,!=1.8.1,
Requirement already satisfied: pydantic-core==2.33.2 in c:\users\boyar\desktop\дипломна\web\backend\venv\lib\site-packages (from pydantic!=1.8,!=1.8.1,!
Requirement already satisfied: typing-inspection>=0.4.0 in c:\users\boyar\desktop\дипломна\web\backend\venv\lib\site-packages (from pydantic!=1.8,!=1.8.
Requirement already satisfied: anyio<5,>=3.6.2 in c:\users\boyar\desktop\дипломна\web\backend\venv\lib\site-packages (from starlette<0.47.0,>=0.40.0->fa
Requirement already satisfied: idna>=2.8 in c:\users\boyar\desktop\дипломна\web\backend\venv\lib\site-packages (from anyio<5,>=3.6.2->starlette<0.47.0,>
Requirement already satisfied: sniffio>=1.1 in c:\users\boyar\desktop\дипломна\web\backend\venv\lib\site-packages (from anyio<5,>=3.6.2->starlette<0.47.
Requirement already satisfied: click>=7.0 in c:\users\boyar\desktop\дипломна\web\backend\venv\lib\site-packages (from uvicorn) (8.1.8)
Requirement already satisfied: h11>=0.8 in c:\users\boyar\desktop\дипломна\web\backend\venv\lib\site-packages (from uvicorn) (0.16.0)
Requirement already satisfied: numpy>=1.21.2 in c:\users\boyar\desktop\дипломна\web\backend\venv\lib\site-packages (from opencv-python) (2.2.5)
Requirement already satisfied: packaging>=21.3 in c:\users\boyar\desktop\дипломна\web\backend\venv\lib\site-packages (from pytesseract) (25.0)
Requirement already satisfied: Pillow>=8.0.0 in c:\users\boyar\desktop\дипломна\web\backend\venv\lib\site-packages (from pytesseract) (11.2.1)
Requirement already satisfied: colorama in c:\users\boyar\desktop\дипломна\web\backend\venv\lib\site-packages (from click>=7.0->uvicorn) (0.4.6)
```

Рисунок 3.10 – Встановлення необхідних компонентів

3. Розробка основного коду FastAPI з ендпоінтами для роботи з модулем ESP32-CAM.

```python
app = FastAPI()

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_methods=["*"],
    allow_headers=["*"],
)

ESP32_URL = "http://192.168.0.102/capture"

@app.get("/capture")
def proxy_capture():
    try:
        response = requests.get(ESP32_URL, timeout=5)
        return JSONResponse(content=response.json())
    except Exception as e:
        return JSONResponse(status_code=500, content={"error": str(e)})

@app.post("/upload/")
async def upload_image(request: Request):
    os.makedirs("images", exist_ok=True)
    filename = datetime.now().strftime("%Y%m%d_%H%M%S") + ".jpg"
    with open(f"images/{filename}", "wb") as f:
        f.write(await request.body())
    return {"message": "Фото збережено", "filename": filename}

@app.get("/images/")
async def list_images():
```

Рисунок 3.11 – Вигляд вікна Visual Studio Code з кодом реалізації

4. Запуск сервера для перевірки його роботи.

```
PS C:\Users\boyar\Desktop\Дипломна\Web\Backend> uvicorn main:app --reload
INFO:     Will watch for changes in these directories: ['C:\\Users\\boyar\\Desktop\\Дипломна\\Web\\Backend']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [11844] using StatReload
INFO:     Started server process [13836]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

Рисунок 3.12 – Успішний старт серверу

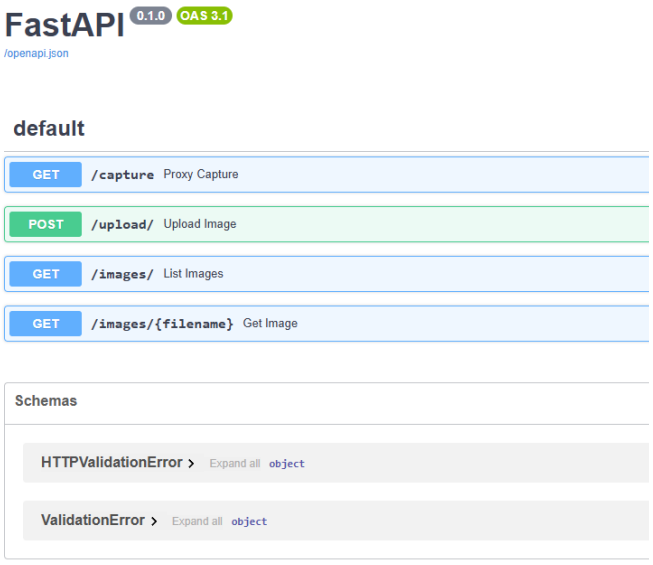


Рисунок 3.13 – Документація з наявними роутами

Отже, розроблена Backend-частина системи успішно забезпечує повний цикл обробки зображень, отриманих із модуля ESP32-CAM, включаючи їх прийом, попередню обробку, виконання OCR-аналізу та передачу результатів користувачу. Вибір технологій, таких як FastAPI, OpenCV та Tesseract, дозволив реалізувати надійне та продуктивне серверне середовище, готове до розширення функціоналу в майбутньому. Всі компоненти інтегровані у єдину систему, що дозволяє забезпечити ефективну та масштабовану обробку даних у реальному часі.

3.3 Реалізація веб-інтерфейсу для взаємодії з системою

Після створення Backend-частини, постала потреба у реалізації інтуїтивно зрозумілого та зручного веб-інтерфейсу для взаємодії з системою. Основною метою розробки є створення простого, інтуїтивного інтерфейсу, який дозволяє завантажувати зображення, переглядати процес розпізнавання тексту та отримувати результати в зручному вигляді.

Веб-інтерфейс реалізовано за допомогою фреймворку Vue.js – сучасного JavaScript-фреймворку для створення односторінкових застосунків. Серед основних компонентів інтерфейсу є:

1. Головна сторінка зі списком зображень, які були зроблені.
2. Сторінка результатів розпізнавання.

Для створення необхідно розробити прототипи взаємодії користувача з системою та візуалізувати основні компоненти і визначити стильові рішення у середовищі Figma.

Створені сторінки інтерфейсу, наведено на рисунках 3.14 – 3.16. Вони демонструють головну сторінку зі списком зображень, сторінку з прогресом обробки та сторінку результатів розпізнавання.

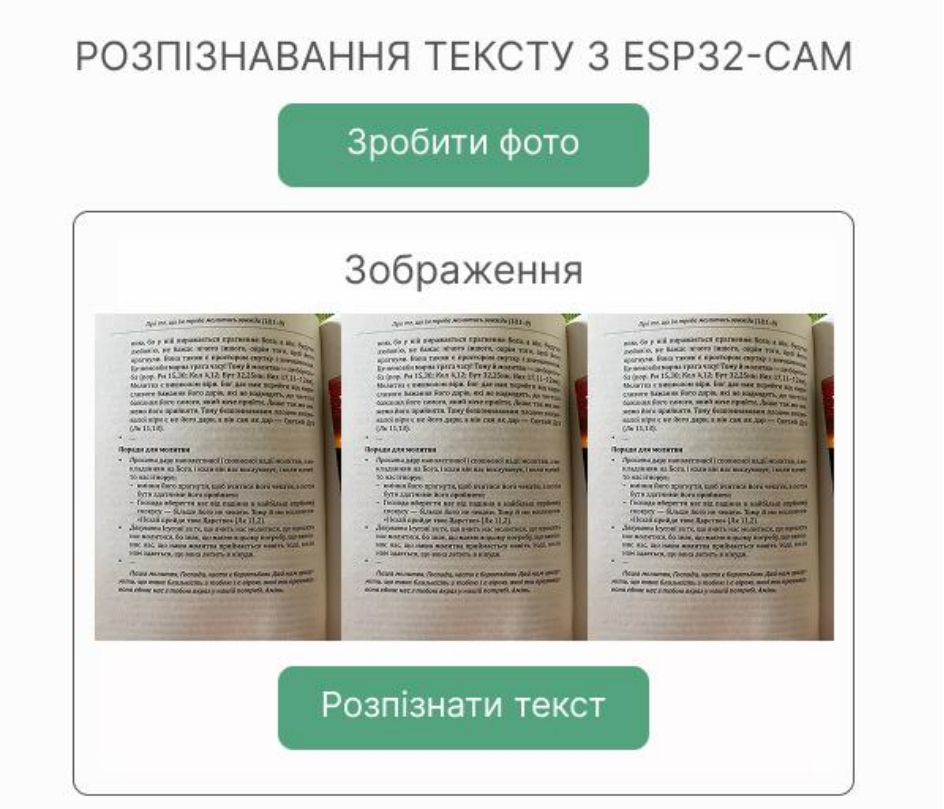


Рисунок 3.14 – Головна сторінка веб-інтерфейсу

Екран головної сторінки містить дві кнопки:

1. Кнопка «Зробити фото» - відповідає за активацію камери та збереження знімка, зробленого з модуля ESP32-CAM. Після натискання на цю кнопку система надсилає запит до пристрою, отримує зображення та відображає його на сторінці.

2. Кнопка «Розпізнати текст» - запускає процес обробки зображення за допомогою модуля розпізнавання тексту (OCR). Зображення передається на сервер, де за допомогою алгоритмів машинного зору виконується розпізнавання текстового вмісту. Результат обробки виводиться на окремій сторінці.

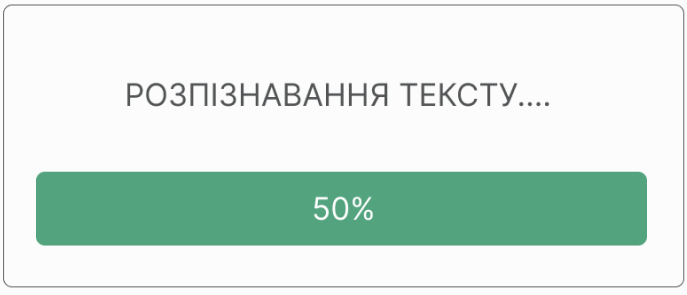


Рисунок 3.15 – Відображення прогресу обробки

Прогрес-бар використовується для відображення поточного статусу обробки, надаючи користувачу інформацію про хід розпізнавання, оскільки сам процес може тривати деякий час.

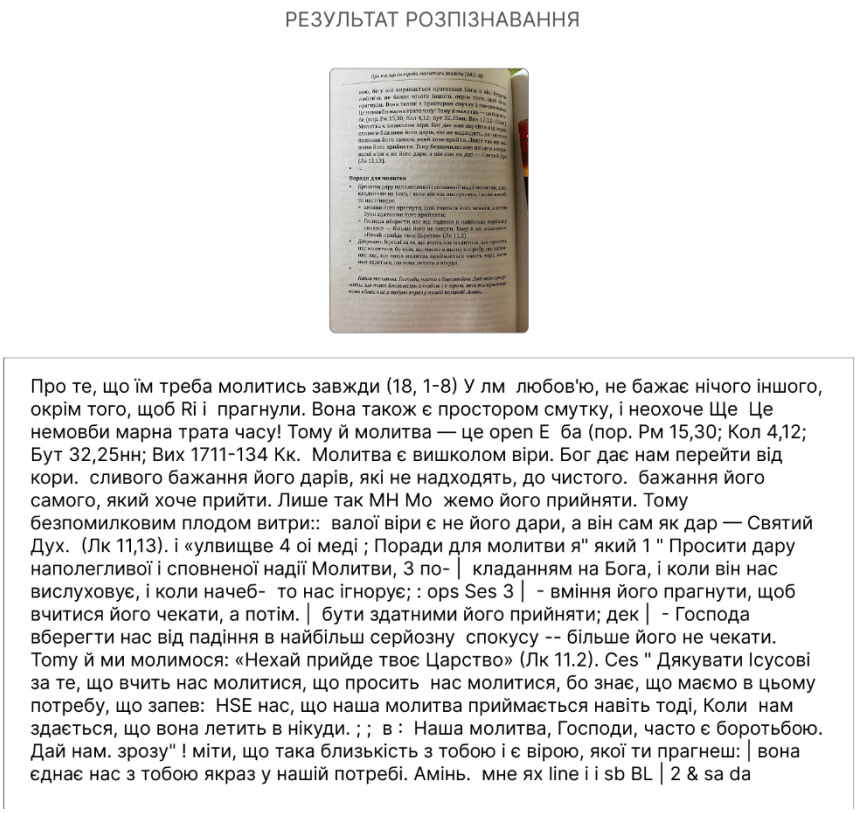


Рисунок 3.16 – Сторінка результату розпізнавання

Сторінка результатів містить розпізнаний текст, який є результатом розпізнавання.

Таким чином, веб-інтерфейс системи, реалізований з використанням Vue.js, який є зручним, адаптивним та доступним інструментом. Створені прототипи у Figma дозволили якісно спроєктувати інтерфейс, а реалізація за допомогою сучасних технологій забезпечила високу продуктивність і стабільність роботи.

Наступним етапом після розробки дизайну стала верстка та створення застосунку Vite + Vue + TypeScript. Нижче описано ключові етапи створення застосунку, які зображено на рисунках 3.17 – 3.21:

1. Ініціалізація проєкту.

Створення проєкту здійснювалося за допомогою команди:

npm create vue@latest

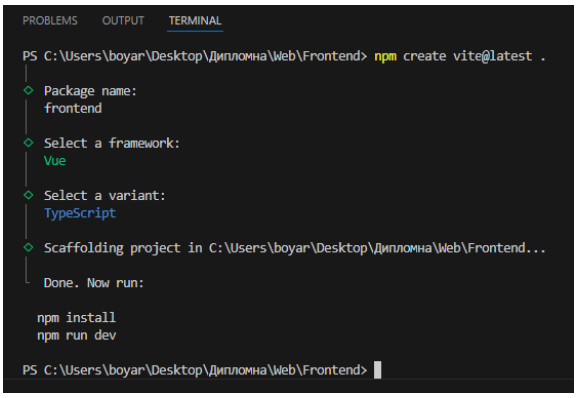


Рисунок 3.17 – Успішне створення проєкту Vite + Vue + Typescript

2. Структура проєкту.

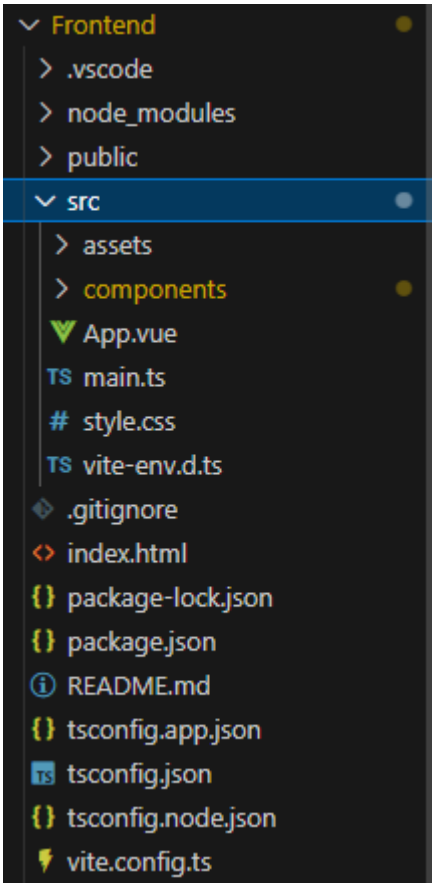


Рисунок 3.18 – Папкова структура проєкту Frontend

3. Встановлення необхідних бібліотек.

npm install axios

2. Встановлення необхідних бібліотек та пакетів.

```
PS C:\Users\boyar\Desktop\Дипломна\Web\test> npm install

added 50 packages, and audited 51 packages in 9s

7 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Users\boyar\Desktop\Дипломна\Web\test>
```

Рисунок 3.19 – Успішне встановлення

3. Перший запуск проєкту.

Для запуску проєкту необхідно виконати команду – npm run dev

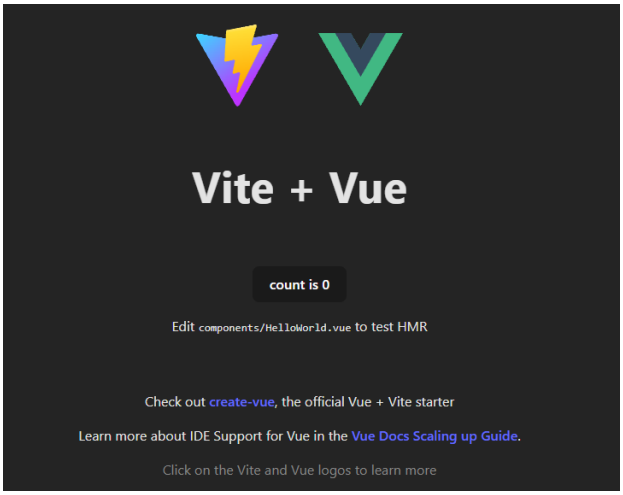


Рисунок 3.20 – Вигляд стандартної сторінки Vite + Vue після запуску

4. Розробка основного коду для веб-інтерфейсу.

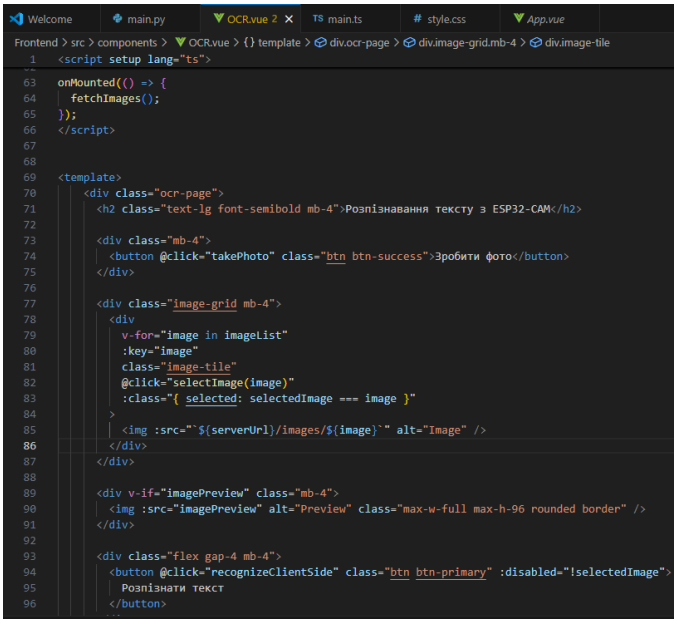


Рисунок 3.21 – Вигляд вікна Visual Studio Code з кодом реалізації

3.4 Тестування системи та аналіз результатів її роботи

З метою перевірки функціональності розробленої системи проведено тестування. Основною метою тестування є перевірка відповідності реалізованої системи технічним вимогам, визначеним на етапі проєктування, та оцінка її працездатності.

Процес тестування включає такі етапи:

1. Розробка тестових сценаріїв.
2. Визначення метрик оцінювання якості роботи системи.
3. Проведення функціонального тестування.
4. Аналіз результатів тестування.

Тестування виконувалось за допомогою контрольних наборів зображень різної складності, що містять текст різних шрифтів, розмірів і мов.

Рисунок 3.22 – Діаграма трасування вимог

На рисунку 3.22 зображено відповідність вимог до реалізованих компонентів системи. Вимоги: захоплення зображення, передача даних, обробка зображень, розпізнавання тексту, інтеграція інтерфейсу – зіставлені з відповідними модулями: ESP32-CAM, FastAPI, OpenCV, Tesseract OCR, Vue.js.

Тестування проводилося на основі розроблених тестових сценаріїв, кожен з яких відповідав певному функціональному модулю системи. Всього було підготовлено п’ять основних сценаріїв тестування.

Таблиця 3.2 – Основні тестові сценарії

№	Назва	Опис	Вхідні дані	Очікуваний результат	Критерій успіху
1	Завантаження зображення	Виконати функцію захоплення фото через інтерфейс	-	Зображення захоплене ESP32-CAM	Завантажене зображення в локальну папку
2	Обробка зображення	Виконати фільтрацію та бінаризацію	JPEG	Готове до OCR зображення	Оброблене зображення
3	Розпізнавання тексту	Розпізнати текст на зображенні	Передоброблене зображення	Витягнутий текст	> 90% правильних символів
4	Відображення результату	Отримати перелянутий текст	ID сесії	Відображення тексту у браузері	Текст на екрані

Для кожного сценарію визначено очікувані результати та критерії успішності, що дозволяє об’єктивно оцінити відповідність фактичної роботи системи заявленим характеристикам [19].

Під час тестування була проведена серія експериментів на контрольних зображеннях. Вхідний набір включав:

1. 5 зображень із машинописним текстом (рисунок 3.23).

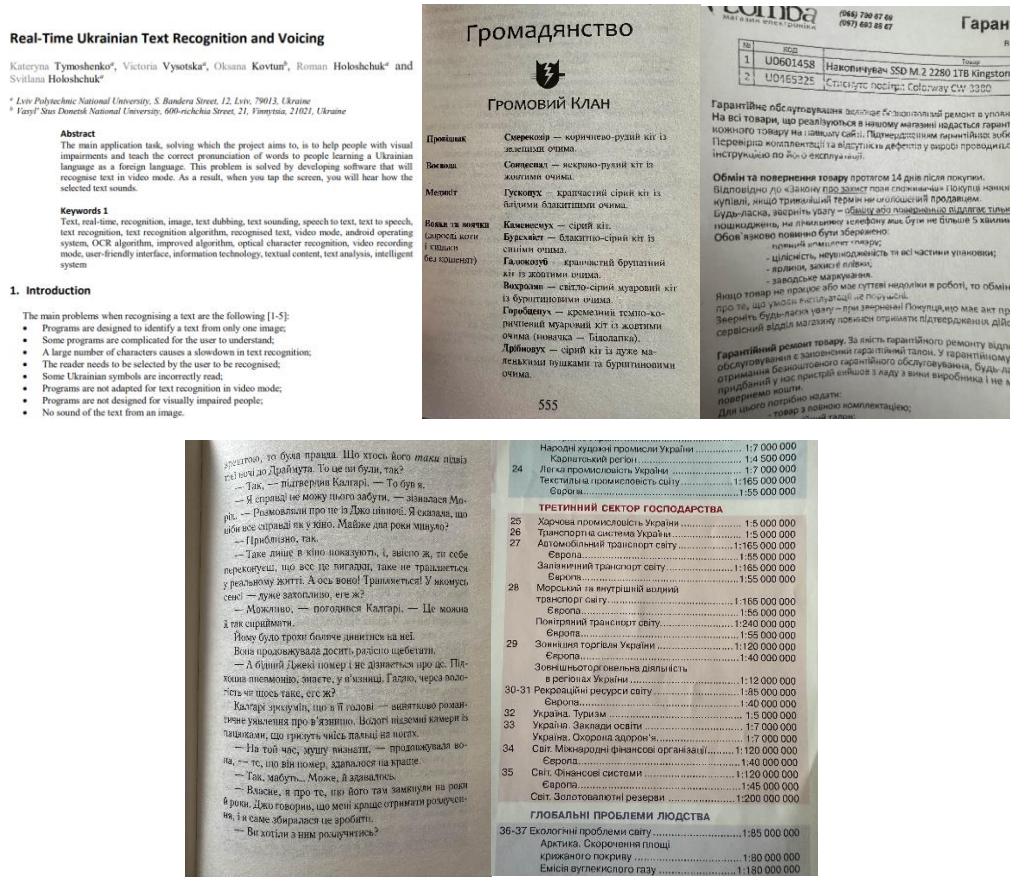


Рисунок 3.23 – Набір зображень із машинописним текстом

2. 5 зображень із рукописним текстом (рисунок 3.24).

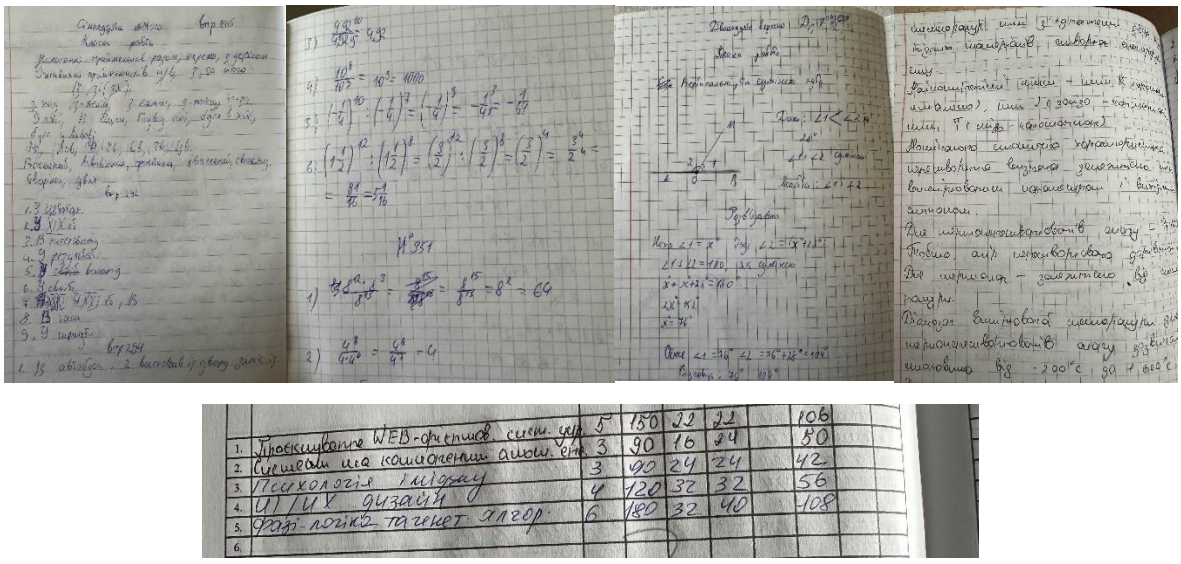


Рисунок 3.24 – Набір зображень із рукописним текстом

Таблиця 3.3 – Результати тестування системи

Тип	**Кількість тестів**	**Успішних тестів**	**Показник успішності**	**Середній час обробки**	**Точність розпізнавання**
Попередня обробка	10	10	100%	0,6	-
Розпізнавання тексту	10	9	90%	2	92%
Відображення результатів	10	10	100%	0,5	-

Результати тестування (таблиця 3.3) підтверджують працездатність розробленої системи, її відповідність технічним вимогам і здатність забезпечити досить високий рівень точності.

3.5 Аналіз точності, обмеження та перспективи покращення розробленої системи

Аналіз результатів тестування дозволяє визначити рівень відповідності створеної системи початковим вимогам, оцінити її продуктивність, надійність, функціональність та визначити напрямки для подальшого удосконалення.

Під час тестування система показала достатній рівень функціональності, а результати тестів свідчать про її здатність успішно виконувати поставлені завдання в різних умовах експлуатації.

Аналіз точності розпізнавання тексту показав, що середнє значення успішності складає 92%. Тобто система здатна правильно розпізнавати більше ніж 9 символів з 10, що є прийнятим рівнем для практичного застосування.

Швидкість обробки запиту є критично важливою для інтерактивних систем. Досягнутий середній час 2,6 секунди вказує на можливість обробки даних у режимі, близькому до реального часу, що забезпечує комфортну взаємодію з користувачем.

Варто зазначити, що система забезпечує стабільну передачу даних від модуля ESP32-CAM до серверної частини FastAPI. Незначна кількість втрат не вплинула на загальну якість роботи. Це свідчить про те, що система ефективно справляється з навантаженням середнього рівня і може бути масштабована для використання в умовах середньої інтенсивності.

Таким чином, аналіз результатів тестування показав, що розроблена система є надійною, ефективною та зручною для користувачів. Вона відповідає всім технічним вимогам і забезпечує високий рівень автоматизації процесу розпізнавання тексту з використанням технологій машинного зору.

ВИСНОВКИ

Системи машинного зору, які поєднують апаратні засоби з алгоритмами штучного інтелекту демонструють високий потенціал у вирішенні прикладних завдань, пов’язаних з автоматичним аналізом візуальної інформації. Одним із таких завдань є оптичне розпізнавання тексту, що дозволяє перетворювати зображення текстових документів у машиночитанний формат. У цій роботі було розроблено повноцінну систему для розпізнавання тексту з використанням ESP32-CAM як джерела зображень, що передаються на сервер для обробки та аналізу.

Наскільки нам відомо, ця робота є однією з перших, що використовує інтеграцію компактного мікроконтролера ESP32-CAM з програмними бібліотеками, такими як FastAPI, OpenCV та Tesseract OCR, для створення доступних і економічних рішень розпізнавання тексту. Важливим внеском цієї роботи є забезпечення повного циклу обробки – від отримання зображення до видачі результату користувачу через зручний та адаптивний веб-інтерфейс, реалізований на базі Vue.js. Ця інтеграція дозволяє максимально ефективно використовувати обмежені апаратні ресурси ESP32-CAM.

Запропонований метод забезпечує достатнє розпізнавання тексту навіть в умовах низької якості вихідних зображень завдяки застосуванню попередньої обробки засобами бібліотеки OpenCV, що включає фільтрацію шумів, бінаризацію та морфологічну обробку.

Ця робота також відкриває перспективи для подальших досліджень, зокрема застосування глибоких нейронних мереж, що можуть значно підвищити якість розпізнавання за рахунок адаптивного аналізу тексту в різноматнітних умовах. Крім того, розвиток алгоритмів OCR на базі нейронних мереж може допомогти ефективніше обробляти рукописні тексти та складні зображення зі змішаним фоном, які залишаються важкою задачею для традиційних методів.

Інтеграція систем машинного зору в Інтернет речей, автономні системи та мобільні додати додатково розширить сфери використання даного рішення.

Таким чином, отримані результати демонструють, що система, розроблена в цій роботі, може стати ефективним та економічно доступним інструментом для автоматичного розпізнавання тексту, значно полегшуючи процес цифровізації документів та розширюючи доступність до цієї технології для широкого кола користувачів і застосувань.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 5 Steps to OCR Training Data. AIMultiple: High Tech Use Cases & Tools to Grow Your Business. URL: https://research.aimultiple.com/ocr-training-data/ (дата звернення: 08.12.2024).
2. A Quick Introduction to Neural Networks. Ujjwal Karn's blog. URL: https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/ (дата звернення: 08.12.2024).
3. Advantages and disadvantages of the main programming languages - ITC Web Solutions. *ITC Web Solutions*. URL: https://itcwebsolutions.com/en/webdevelopment-and-support/programming-and-tools/programming-languages/advantagesand-disadvantages-of-the-main-programming-languages/ (дата звернення: 23.05.2025).
4. Ahmed M. M., Qays M. O., Abu-Siada A., Muyeen S. M., Hossain M. L. Cost-effective design of IoT-based smart household distribution system // Designs. – 2021. – Vol. 5, No. 3. – Article № 55. – DOI: 10.3390/designs5030055.
5. Axelrod A. Complete Guide to Test Automation: Techniques, Practices, and Patterns for Building and Maintaining Effective Software Projects / A. Axelrod. – NY: Apress, 2018. – 588 p.
6. Backpropagation. Strona główna -- Serwis Akademii Górniczo-Hutniczej. URL: https://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html (дата звернення: 23.05.2025).
7. Balaji D. Handling Multiple File Uploads with Multer in Node.js. *Medium*. URL: https://dvmhn07.medium.com/handling-multiple-file-uploads-with-multer-innode-js-c1901a51f8b8 (дата звернення: 23.05.2025).
8. Espressif Systems. ESP32-CAM Datasheet. Espressif. URL: https://www.espressif.com/sites/default/files/documentation/esp32cam_datasheet_en.pdf (дата звернення: 23.05.2025).

9. DrMax. How Does Optical Character Recognition Work | Baeldung on Computer Science. *Baeldung on Computer Science*. URL: https://www.baeldung.com/cs/ocr#1-useful-text-from-an-image (дата звернення: 23.05.2025).
10. EAST: An Efficient and Accurate Scene Text Detector. arXiv.org. URL: https://arxiv.org/abs/1704.03155 (дата звернення: 23.05.2025).
11. Elkholy M. H., Senjyu T., Lotfy M. E., Elgarhy A., Ali N. S., Gaafar T. S. Design and implementation of a real-time smart home management system considering energy saving // Sustainability. – 2022. – Vol. 14. – Article № 13840. – DOI: 10.3390/su142113840.
12. Free PDF Reader & Viewer | FineReader PDF. *FineReader PDF*. URL: https://pdf.abbyy.com/finereader-pdf-viewer/ (дата звернення: 18.05.2024).
13. GeeksforGeeks. Activation functions in Neural Networks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/activation-functions-neural-networks/ (дата звернення: 23.05.2025).
14. GitHub - tesseract-ocr/tesseract: Tesseract Open Source OCR Engine (main repository). *GitHub*. URL: https://github.com/tesseract-ocr/tesseract (дата звернення: 23.05.2025).
15. Hakimi S. M., Saadatmandi M., Shafie-khah M., Catalão J. P. S. Smart household management systems with renewable generation to increase the operation profit of a microgrid // IET Smart Grid. – 2019. – Vol. 2, No. 4. – P. 522–528. – DOI: 10.1049/iet-stg.2018.0299.
16. Hoang C. Monolith Architecture. *Medium*. URL: https://tech.tamara.co/monolith-architecture-5f00270f384e (дата звернення: 23.05.2025).
17. Hoffer, J. A., George, J. F., & Valacich, J. S. Modern Systems Analysis and Design. 9th ed. Pearson. 2020. – p. 528
18. How OCR Works: An In-Depth Explanation of Optical Character Recognition. *We Label Data*.

19. How to use OCR software for PDFs in 4 easy steps | Adobe Acrobat. *Adobe Acrobat.* URL: https://www.adobe.com/acrobat/how-to/ocr-software-convert-pdf-totext.html (дата звернення: 23.05.2025).
20. IBM. What is Computer Vision? | IBM. IBM - United States. URL: https://www.ibm.com/topics/computer-vision (дата звернення: 23.05.2025).
21. IBM. What Is Machine Learning (ML)? | IBM. IBM - United States. URL: https://www.ibm.com/topics/machine-learning (дата звернення: 23.05.2025).
22. Into Enterprise Systems with your Data. URL: https://www.xenonstack.com/insights/crnn-for-text-recognition (дата звернення: 23.05.2025).
23. Isbarov J. Going deeper with convolutions: The Inception paper, explained. Medium. URL: https://medium.com/aiguys/going-deeper-with-convolutions-theinception-paper-explained-841a0c661fd3 (дата звернення: 23.05.2025).
24. Khalid. What Is OCR And What Is It Used For?. *Docparser*. URL: https://docparser.com/blog/what-is-ocr/ (дата звернення: 30.05.2024).
25. Machine learning, explained | MIT Sloan. MIT Sloan. URL: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained (дата звернення: 23.05.2025).
26. Manok A. All You Need to Know about Machine Learning OCR | Affinda. *Affinda | Become An AI-First Company*. URL: https://www.affinda.com/techai/machine-learning-ocr (дата звернення: 23.05.2025).
27. OCR Software, Data Extraction Tool - Amazon Textract - AWS. *Amazon*
28. Pandey P. An Efficient and Accurate Scene Text Detector [EAST]. Medium. URL: https://poshan0126.medium.com/an-efficient-and-accurate-scene-text-detectoreast-973df9dfdd55 (дата звернення: 23.05.2025).
29. Precious L. Building and structuring a Node.js MVC application - LogRocket Blog. *LogRocket Blog*. URL: https://blog.logrocket.com/buildingstructuring-node-js-mvc-application/ (дата звернення: 23.05.2025).

30. How to use OCR software for PDFs in 4 easy steps | Adobe Acrobat. *Adobe Acrobat.* URL: https://www.adobe.com/acrobat/how-to/ocr-software-convert-pdf-totext.html (дата звернення: 23.05.2025).
31. IBM. What is Computer Vision? | IBM. IBM - United States. URL: https://www.ibm.com/topics/computer-vision (дата звернення: 23.05.2025).
32. IBM. What Is Machine Learning (ML)? | IBM. IBM - United States. URL: https://www.ibm.com/topics/machine-learning (дата звернення: 23.05.2025).
33. Into Enterprise Systems with your Data. URL: https://www.xenonstack.com/insights/crnn-for-text-recognition (дата звернення: 23.05.2025).
34. Isbarov J. Going deeper with convolutions: The Inception paper, explained. Medium. URL: https://medium.com/aiguys/going-deeper-with-convolutions-theinception-paper-explained-841a0c661fd3 (дата звернення: 23.05.2025).
35. Khalid. What Is OCR And What Is It Used For?. *Docparser*. URL: https://docparser.com/blog/what-is-ocr/ (дата звернення: 30.05.2024).
36. Machine learning, explained | MIT Sloan. MIT Sloan. URL: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained (дата звернення: 23.05.2025).
37. Manok A. All You Need to Know about Machine Learning OCR | Affinda. *Affinda | Become An AI-First Company*. URL: https://www.affinda.com/techai/machine-learning-ocr (дата звернення: 23.05.2025).
38. OCR Software, Data Extraction Tool - Amazon Textract - AWS. *Amazon*
39. Pandey P. An Efficient and Accurate Scene Text Detector [EAST]. Medium. URL: https://poshan0126.medium.com/an-efficient-and-accurate-scene-text-detectoreast-973df9dfdd55 (дата звернення: 23.05.2025).
40. Precious L. Building and structuring a Node.js MVC application - LogRocket Blog. *LogRocket Blog*. URL: https://blog.logrocket.com/buildingstructuring-node-js-mvc-application/ (дата звернення: 23.05.2025).

41. PVANET: Deep but Lightweight Neural Networks for Real-time Object Detection. arXiv.org. URL: https://arxiv.org/abs/1608.08021 (дата звернення: 23.05.2025).
42. Smith R. An Overview of the Tesseract OCR Engine. URL: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/ 33418.pdf (дата звернення: 23.05.2025).
43. Solution G. T. A Comprehensive List of OCR Datasets for Machine Learning. Medium. URL: https://medium.com/@shalinigts16/a-comprehensive-list-of-ocrdatasets-for-machine-learning-d3c29ed9983a (дата звернення: 23.05.2025).
44. Tesseract.js | Pure Javascript OCR for 100 Languages!. *Tesseract.js | Pure Javascript OCR for 100 Languages!*. URL: https://tesseract.projectnaptha.c om/ (дата звернення: 23.05.2025).
45. Tiwari N. [ML Story] Computer Vision made easy with Google Cloud Vision API. *Medium*. URL: https://dvmhn07.medium.com/handling-multiple-fileuploads-with-multer-in-node-js-c1901a51f8b8 (дата звернення: 23.05.2025).
46. Tripathi P. 10 text recognition algorithms. *Docsumo - Document AI Platform Built for Scale & Efficiency*. URL: https://www.docsumo.com/blog/text-recognitionalgorithms (дата звернення: 23.05.2025).
47. Vision AI: Image & Visual AI Tools. *Google Cloud*. URL: https://cloud.google.com/vision?hl=ru (дата звернення: 23.05.2025).
48. *Web Services, Inc.* URL: https://aws.amazon.com/textract/?nc1=h_ls (дата звернення: 23.05.2025).
49. What Is Optical Character Recognition (OCR)? - IBM Blog. *IBM Blog*. URL: https://www.ibm.com/blog/optical-character-recognition/ (дата звернення: 23.05.2025).

ДОДАТОК А

Лістинг програмного коду

ESP32-CAM

```
#include "esp_camera.h"
#include <WiFi.h>
#include <WebServer.h>
#include <HTTPClient.h>
#define CAMERA_MODEL_AI_THINKER
#include "camera_pins.h"
const char *ssid = "RomanB";
const char *password = "romanb";
const char *serverUrl = "http://192.168.0.108:8000/upload/";
WebServer server(80);
bool isBusy = false;
String captureAndSend() {
  digitalWrite(4, HIGH);
  delay(400);
  camera_fb_t *fb = esp_camera_fb_get();
  if (fb) esp_camera_fb_return(fb);
  delay(150);
  fb = esp_camera_fb_get();
  digitalWrite(4, LOW);
  if (!fb) {
    Serial.println("Не вдалося отримати кадр з камери");
    return "";
  }
  HTTPClient http;
  http.begin(serverUrl);
  http.addHeader("Content-Type", "image/jpeg");
  int res = http.POST(fb->buf, fb->len);
  String response = "";
  if (res > 0) {
    response = http.getString();
    Serial.println("Фото надіслано: " + response);
  } else {
    Serial.printf("POST помилка: %d\n", res);
  }
  http.end();
  esp_camera_fb_return(fb);
  delay(300);
```

```
  return response;
}
void handleCapture() {
  if (isBusy) {
    server.send(503, "application/json", "{\"error\":\"Камера зайнята. Спробуйте
пізніше.\"}");
    return;
  }
  isBusy = true;
  String json = captureAndSend();
  delay(500);
  isBusy = false;
  if (json.length() > 0) {
    server.send(200, "application/json", json);
  } else {
    server.send(500, "application/json", "{\"error\":\"Не вдалося зробити
фото\"}");
  }
}
void setup() {
  Serial.begin(115200);
  pinMode(4, OUTPUT); digitalWrite(4, LOW);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("WiFi підключено");
  if (!psramFound()) {
    Serial.println("PSRAM не знайдено");
    return;
  }
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
```

```
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sccb_sda = SIOD_GPIO_NUM;
  config.pin_sccb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;
  config.frame_size = FRAMESIZE_SVGA;
  config.jpeg_quality = 6;
  config.fb_count = 1;
  config.fb_location = CAMERA_FB_IN_PSRAM;
  config.grab_mode = CAMERA_GRAB_WHEN_EMPTY;
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Ініціалізація камери не вдалася. Код помилки: 0x%x\n", err);
    return;
  }
  server.on("/capture", handleCapture);
  server.begin();
  Serial.print("ESP32-CAM запущено: http://");
  Serial.println(WiFi.localIP());
}
void loop() {
  server.handleClient();
}
```

Backend

```
import requests
from fastapi import FastAPI, Request
from fastapi.responses import StreamingResponse, JSONResponse
from fastapi.middleware.cors import CORSMiddleware
from datetime import datetime
import os
app = FastAPI()
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
```

```
    allow_methods=["*"],
    allow_headers=["*"],
)
ESP32_URL = "http://192.168.0.102/capture"
@app.get("/capture")
def proxy_capture():
    try:
        response = requests.get(ESP32_URL, timeout=5)
        return JSONResponse(content=response.json())
    except Exception as e:
        return JSONResponse(status_code=500, content={"error": str(e)})
@app.post("/upload/")
async def upload_image(request: Request):
    os.makedirs("images", exist_ok=True)
    filename = datetime.now().strftime("%Y%m%d_%H%M%S") + ".jpg"
    with open(f"images/{filename}", "wb") as f:
        f.write(await request.body())
    return {"message": "Фото збережено", "filename": filename}
@app.get("/images/")
async def list_images():
    os.makedirs("images", exist_ok=True)
    files = sorted(os.listdir("images"), reverse=True)
    return {"images": files}
@app.get("/images/{filename}")
async def get_image(filename: str):
    path = f"images/{filename}"
    if not os.path.exists(path):
        return {"error": "File not found"}
    return StreamingResponse(open(path, "rb"), media_type="image/jpeg")
```

Frontend

```
<script setup lang="ts">
import { ref, onMounted } from 'vue';
import Tesseract from 'tesseract.js';
import axios from 'axios';
const serverUrl = 'http://127.0.0.1:8000';
const esp32Url = 'http://192.168.0.102/capture';
type ImageFilename = string;
const imageList = ref<ImageFilename[]>([]);
const selectedImage = ref<ImageFilename | null>(null);
const imagePreview = ref<string | null>(null);
const recognizedText = ref<string>('');
```

```
const loading = ref(false);
async function fetchImages() {
  try {
    const res = await axios.get(`${serverUrl}/images/`);
    imageList.value = res.data.images || [];
  } catch (err) {
    console.error('Помилка при отриманні списку зображень:', err);
  }
}

async function takePhoto() {
  try {
    const response = await axios.get(`${serverUrl}/esp32/capture`);
    const filename = response.data?.filename as ImageFilename;
    if (!filename) throw new Error('Не отримано назву зображення');
    imageList.value.unshift(filename);
    selectImage(filename);
  } catch (err) {
    console.error('Не вдалося зробити фото:', err);
    alert('Не вдалося зробити фото. Перевір ESP32-CAM або сервер.');
  }
}
function selectImage(imageName: ImageFilename) {
  selectedImage.value = imageName;
  imagePreview.value = `${serverUrl}/images/${imageName}`;
  recognizedText.value = '';
}
async function recognizeClientSide() {
  if (!imagePreview.value) return;

  loading.value = true;
  try {
    const result = await Tesseract.recognize(imagePreview.value, 'ukr+eng', {
      logger: (m) => console.log(m),
    });
    recognizedText.value = result.data.text;
  } catch (err) {
    recognizedText.value = 'Помилка при розпізнаванні.';
  } finally {
    loading.value = false;
  }
}
```

```
onMounted(() => {
  fetchImages();
});
</script>
<template>
    <div class="ocr-page">
      <h2 class="text-lg font-semibold mb-4">Розпізнавання тексту з ESP32-
CAM</h2>
      <div class="mb-4">
        <button @click="takePhoto" class="btn btn-success">Зробити фото</button>
      </div>
      <div class="image-grid mb-4">
        <div
          v-for="image in imageList"
          :key="image"
          class="image-tile"
          @click="selectImage(image)"
          :class="{ selected: selectedImage === image }"
        >
          <img :src="`${serverUrl}/images/${image}`" alt="Image" />
        </div>
      </div>
      <div v-if="imagePreview" class="mb-4">
        <img :src="imagePreview" alt="Preview" class="max-w-full max-h-96
rounded border" />
      </div>
      <div class="flex gap-4 mb-4">
        <button @click="recognizeClientSide" class="btn btn-primary"
:disabled="!selectedImage">
          Розпізнати текст
        </button>
      </div>
      <div v-if="loading" class="text-blue-600">Розпізнавання триває...</div>
      <div style="display: flex; justify-content: center; flex-direction:
column; align-items: center; margin-top: 350px; gap: 15px;">
        <div style="font-size: 25px; color: #525456; text-transform:
uppercase;">Результат розпізнавання</div>
        <div v-if="imagePreview" class="mb-4">
          <img :src="imagePreview" alt="Preview" class="max-w-full max-h-96
rounded border" style="width: 50%; border: 1px solid; border-color: #525456;" />
        </div>
```

```
        <pre v-if="recognizedText" class="bg-gray-100 p-4 rounded whitespace-
pre-wrap" style="width: fit-content; padding: 35px; color: black; border: 1px
solid; border-color: #525456;">
          {{ recognizedText }}
        </pre>
      </div>
    </div>
  </template>
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи – «Розроблення системи розпізнавання тексту з використанням технологій машинного зору».

Обсяг пояснювальної записки в аркушах – 63.

Перелік графічного матеріалу:

КРБ.СІ-02.00.001 Е1 – Система розпізнавання тексту. Схема структурна (аркушів – 1).

КРБ.СІ-02.00.002 Е2 – Розташування виводів ESP32-CAM. Схема функціональна (аркушів – 1).

КРБ.СІ-02.00.003 Е2 – Веб-інтерфейс системи розпізнавання тексту. Схема функціональна (аркушів – 1).

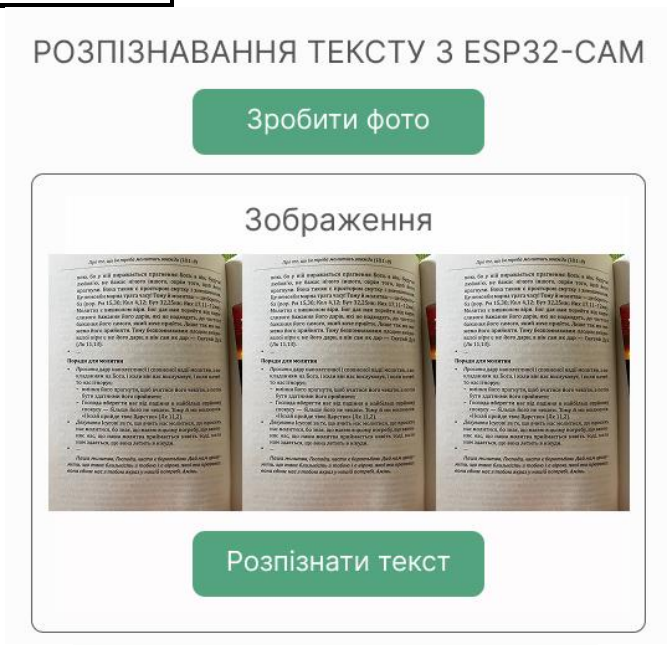
Дата закінчення бакалаврської роботи «14» червня 2025 р.

Студент _____ Боярський Р. Р.

КРБ.СІ-02.00.001 Е1

					КРБ.СІ-02.00.001 Е1			
Змн.	Лист	№ докум.	Підпис	Дата		Літ.	Маса	Масштаб
Розроб.		Боярський Р. Р.			*Система розпізнавання тексту. Схема структурна*			
Перевір.		Паньків Х. В.						
Т. Контр.								
Реценз.						Арк. 1	Аркушів 1	
Н. Контр.		Возний А. В.				ІФНТУНГ СІ-21-1		
Затверд.		Заміховський Л. М.						

					КРБ.СІ-02.00.002 Е2			
Змн.	Лист	№ докум.	Підпис	Дата		Літ.	Маса	Масштаб
Розроб.		Боярський Р. Р.			*Розташування виводів ESP32-CAM. Схема функціональна*			
Перевір.		Паньків Х. В.						
Т. Контр.								
Реценз.						Арк. 1	Аркушів 1	
Н. Контр.		Возний А. В.				ІФНТУНГ СІ-21-1		
Затверд.		Заміховський Л. М.						



Про те, що їм треба молитись завжди (18, 1-8) У лм любов'ю, не бажає нічого іншого, окрім того, щоб Ri і прагнули. Вона також є простором смутку, і неохоче Ще Це немовби марна трата часу! Тому й молитва — це open E ба (пор. Рм 15,30; Кол 4,12; Бут 32,25нн; Вих 1711-134 Кк. Молитва є вишколом віри. Бог дає нам перейти від кори. сливого бажання його дарів, які не надходять, до чистого. бажання його самого, який хоче прийти. Лише так МН Мо жемо його прийняти. Тому безпомилковим плодом витри:: валої віри є не його дари, а він сам як дар — Святий Дух. (Лк 11,13). і «улвищве 4 ої меді ; Поради для молитви я" який 1 " Просити дару наполегливої і сповненої надії Молитви, З по- | кладанням на Бога, і коли він нас вислуховує, і коли начеб- то нас ігнорує; : ops Ses 3 | - вміння його прагнути, щоб вчитися його чекати, а потім. | бути здатними його прийняти; дек | - Господа вберегти нас від падіння в найбільш серйозну спокусу -- більше його не чекати. Тому й ми молимося: «Нехай прийде твоє Царство» (Лк 11.2). Ces " Дякувати Ісусові за те, що вчить нас молитися, що просить нас молитися, бо знає, що маємо в цьому потребу, що запев: HSE нас, що наша молитва приймається навіть тоді, Коли нам здається, що вона летить в нікуди. ; ; в : Наша молитва, Господи, часто є боротьбою. Дай нам. зрозу" ! міти, що така близькість з тобою і є вірою, якої ти прагнеш: | вона єднає нас з тобою якраз у нашій потребі. Амінь. мне ях line і і sb BL | 2 & sa da

					КРБ.СІ-02.00.003 Е2			
Змн.	Лист	№ докум.	Підпис	Дата		Літ.	Маса	Масштаб
Розроб.		Боярський Р. Р.			*Веб-інтерфейс системи розпізнавання тексту. Схема функціональна*			
Перевір.		Паньків Х. В.				Арк. 1	Аркушів 1	
Т. Контр.								
Реценз.								
Н. Контр.		Возний А. В.				ІФНТУНГ СІ-21-1		
Затверд.		Заміховський Л. М.						