

**Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра комп'ютерних наук**

**КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»**

**РОЗРОБКА САЙТУ ДЛЯ АНАЛІЗУ ЗОБРАЖЕНЬ НА ОСНОВІ
ШТУЧНОГО ІНТЕЛЕКТУ**

**WEBSITE DEVELOPMENT FOR IMAGE ANALYSIS BASED ON
ARTIFICIAL INTELLIGENCE**

спеціальність 122 Комп'ютерні науки

освітня програма «Комп'ютерні науки»

Виконав: здобувач вищої освіти
групи КН-41
Луцук Володимир Юрійович

(підпис)

Керівник: асистент
Хиць Руслан Андрійович

(підпис)

Кваліфікаційну роботу
допущено до захисту
«__» _____ 2026 р.
Гарант освітньої програми:
д.пед.н., професор
Тулашвілі Юрій Йосипович

(підпис)

Луцьк – 2026 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних та інформаційних технологій

Кафедра комп'ютерних наук

Ступінь вищої освіти: бакалавр

Галузь знань: 12 Інформаційні технології

Спеціальність: 122 Комп'ютерні науки

Освітня програма: «Комп'ютерні науки»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Валерій ЛІЩІНА

« ____ » _____ 2026 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА ПЕРШОГО (БАКАЛАВРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Луцук Володимир Юрійович

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «розробка сайту для аналізу зображень на основі штучного інтелекту»

Керівник асистент Хиць Руслан Андрійович

затверджені наказом закладу вищої освіти від «16» січня 2026 р. № 32/01-02

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи «02» червня 2026 р

3. Вихідні дані до роботи: аналіз сучасних методів комп'ютерного зору та технологій

штучного інтелекту для розпізнавання транспортних засобів, використання моделей YOLOv8 і CLIP, засобів Flask, Python та алгоритмів обробки зображень для розробки вебзастосунку.

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити):

Аналіз сучасного стану проблеми автоматичного розпізнавання транспортних засобів за зображеннями, існуючих методів і засобів її розв'язання; аналіз і вибір засобів проектування вебзастосунку на основі штучного інтелекту; опис функціонального наповнення об'єкта проектування; розробка й обґрунтування системного наповнення; функціонально-структурна схема роботи об'єкта проектування.

5. Перелік графічного матеріалу: 12 рисунків, презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
<i>Аналіз проблеми за темою роботи та постановка завдань дослідження</i>	<i>Хиць Р. А.</i>		
<i>Теоретичне дослідження</i>	<i>Хиць Р. А.</i>		
<i>Практична реалізація</i>	<i>Хиць Р. А.</i>		
<i>Спеціальні розрахунки</i>	<i>Хиць Р. А.</i>		
<i>Показник запозичень тексту</i>	%		
<i>Інструментальна перевірка</i>	<i>Кошелюк В. А.</i>		
<i>Нормоконтроль</i>	<i>Сачук В. О.</i>		
<i>Гарант ОПП</i>	<i>Тулашвілі Ю. Й.</i>		

7. Дата видачі завдання «16» січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	<i>Провести огляд літературних джерел по темі кваліфікаційної роботи</i>	<i>до 20.02.2026 р</i>	
2	<i>Провести аналіз загальної проблеми і вибір напрямків дослідження</i>	<i>до 27.02.2026 р.</i>	
3	<i>Розробити функціональну схему роботи програмного продукту</i>	<i>до 21.03.2026 р</i>	
4	<i>Описати засоби розробки об'єкта проектування</i>	<i>до 17.04.2026 р.</i>	
5	<i>Практична реалізація об'єкта проектування</i>	<i>до 25.04.2026 р.</i>	
6	<i>Провести спеціальні розрахунки</i>	<i>до 04.05.2026 р.</i>	
7	<i>Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедрі</i>	<i>до 02.06.2026 р.</i>	

Здобувач вищої освіти _____ Володимир ЛУЦУК

Керівник роботи _____ Руслан ХИЦЬ

АНОТАЦІЯ

Луцук В. Ю. Розробка сайту для аналізу зображень на основі штучного інтелекту. Рукопис.

Кваліфікаційна робота бакалавра ОП «Комп'ютерні науки». Луцький національний технічний університет. Луцьк, 2026.

Кваліфікаційна робота бакалавра складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі виконано аналіз предметної області автоматичного розпізнавання транспортних засобів, розглянуто існуючі аналоги та обґрунтовано актуальність розробки власного рішення. Другий розділ містить специфікацію вимог до системи, обґрунтування вибору технологічного стеку та опис функціонально-структурної схеми роботи застосунку. У третьому розділі здійснено практичну реалізацію вебзастосунку Neural Vision – Transport Analyzer на основі мікрофреймворку Flask, включаючи побудову AI-конвеєру обробки зображень, розробку клієнтського інтерфейсу та тестування системи. Четвертий розділ присвячено SEO-оптимізації інформаційно-комп'ютерної системи: розглянуто технічні засоби підвищення видимості ресурсу у пошукових системах та стратегії просування серед цільової аудиторії. У висновках узагальнено результати виконаної роботи та визначено перспективи подальшого розвитку системи.

Ключові слова: YOLOv8, CLIP, комп'ютерний зір, розпізнавання транспортних засобів, zero-shot класифікація, Flask, Python, нейронна мережа, вебзастосунок, KMeans.

ANNOTATION

Volodymyr Lutsuk. Development of a website for image analysis based on artificial intelligence. Manuscript.

Bachelor's qualification thesis, Educational Programme «Computer Science». Lutsk National Technical University. Lutsk, 2026.

The bachelor's qualification thesis consists of an introduction, four chapters, conclusions, a list of references and appendices.

The first chapter provides an analysis of the subject area of automatic vehicle recognition, reviews existing analogues and substantiates the relevance of developing an original solution. The second chapter contains the system requirements specification, justification of the technology stack selection and a description of the functional-structural diagram of the application. The third chapter covers the practical implementation of the Neural Vision – Transport Analyzer web application based on the Flask microframework, including the construction of an AI image processing pipeline, development of the client interface and system testing. The fourth chapter is devoted to SEO optimization of the information-computer system: technical means of improving the resource's visibility in search engines and promotion strategies for the target audience are examined. The conclusions summarize the results of the work performed and outline the prospects for further development of the system.

Keywords: YOLOv8, CLIP, computer vision, vehicle recognition, zero-shot classification, Flask, Python, neural network, web application, KMeans.

ЗМІСТ

ВСТУП.....	7
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	10
1.1 Аналіз сучасного стану проблеми	10
1.2 Аналіз аналогічних програм, баз даних, систем, обґрунтування вибраного напрямку та вибір методів рішення.....	13
1.3 Аналіз і вибір засобів проектування сайту для аналізу зображень	15
1.4 Постановка завдання на кваліфікаційну роботу бакалавра	20
РОЗДІЛ 2 СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ	22
2.1 Обґрунтування вибору шляхів, технологій (алгоритмів) і засобів вирішення поставленого завдання.....	22
2.2 Функціонально-структурна схема роботи об'єкта проектування.....	25
2.3 Створення моделі бази даних.....	30
РОЗДІЛ 3 РОЗРОБКА САЙТУ ДЛЯ АНАЛІЗУ ЗОБРАЖЕНЬ.....	34
3.1 Практична реалізація об'єкта проектування. Побудова блок-схем модулів програми.....	34
3.2 Тестування та налагодження системи сайту Transport Analyzer.....	46
3.3 Інструкція користувача з використання програмного продукту.....	56
РОЗДІЛ 4 СПЕЦІАЛЬНІ РОЗРАХУНКИ	63
ВИСНОВКИ.....	67
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	69
ДОДАТКИ.....	71

ВСТУП

Стрімкий розвиток технологій комп'ютерного зору та штучного інтелекту відкриває нові можливості для автоматизації задач, які раніше потребували значних людських ресурсів. Одним із таких напрямів є автоматична ідентифікація транспортних засобів за їх фотографічними зображеннями. Щороку кількість транспортних засобів у світі неупинно зростає, що породжує потребу в ефективних інструментах моніторингу, обліку та класифікації. Традиційні методи – ручне введення даних, сканування номерних знаків або ведення паперових реєстрів – не справляються з вимогами сучасних підприємств через низьку швидкість обробки, схильність до помилок та залежність від людського фактора.

Особливу роль у вирішенні цієї проблеми відіграють мультимодальні нейронні мережі, зокрема модель CLIP від OpenAI, здатна класифікувати зображення у zero-shot режимі без необхідності збирання та розмічення спеціалізованих датасетів. У поєднанні з детектором об'єктів YOLOv8, що забезпечує локалізацію транспортного засобу на зображенні в режимі реального часу, такий підхід дозволяє будувати гнучкі та розширювані системи розпізнавання без трудомістких етапів навчання моделей.

Аналіз існуючих рішень виявив відсутність загальнодоступного вебінструменту, що поєднував би детекцію об'єкта, двопрхідну класифікацію марки та типу транспортного засобу, аналіз кольорової палітри та технічної статистики зображення в єдиному інтерфейсі. Більшість наявних систем є або платними хмарними сервісами з закритим кодом, або вузькоспеціалізованими ANPR-рішеннями.

Метою даної кваліфікаційної роботи є проектування та реалізація вебзастосунку Neural Vision – Transport Analyzer для автоматичного аналізу зображень транспортних засобів із використанням нейронних мереж комп'ютерного зору.

Об'єктом дослідження є процес автоматичної ідентифікації транспортних

засобів за фотографічними зображеннями з використанням методів глибокого навчання.

Предметом дослідження є архітектурні та технологічні рішення при побудові вебзастосунків для аналізу зображень на основі моделей YOLOv8 та CLIP ViT-B/32.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати сучасний стан проблеми автоматичного розпізнавання транспортних засобів та існуючі рішення, виявивши їхні переваги і недоліки;
- обґрунтувати вибір технологічного стеку та архітектурних рішень для реалізації системи;
- спроектувати та реалізувати гібридний AI-конвеєр на основі YOLOv8n, CLIP ViT-B/32 та алгоритму KMeans;
- розробити REST API на базі Flask для прийому зображень та повернення структурованого технічного профілю транспортного засобу;
- створити адаптивний вебінтерфейс із підтримкою drag-and-drop, відображенням bounding box та візуалізацією результатів аналізу;
- провести функціональне тестування системи та оцінити її продуктивність у різних умовах;
- розробити інструкцію користувача та задокументувати процес розгортання системи.

Науковою новизною роботи є розробка гібридного двохпрохідного AI-конвеєра, що поєднує детектор YOLOv8n для локалізації транспортного засобу з моделлю CLIP ViT-B/32 у zero-shot режимі для послідовної класифікації типу та марки і моделі транспортного засобу без необхідності збору спеціалізованих датасетів і донавчання моделей. На відміну від існуючих рішень, запропонована система інтегрує детекцію об'єкта, двоетапну класифікацію та аналіз кольорової палітри в єдиному загальнодоступному вебінтерфейсі.

Практична цінність роботи полягає в тому, що розроблений застосунок може бути використаний підприємствами транспортної галузі, страховими

компаніями, автосалонами та паркінг-операторами для автоматизації обліку та ідентифікації транспортних засобів. Відкрита архітектура системи дозволяє розширювати базу кандидатів і адаптувати рішення до конкретних прикладних задач без перенавчання моделей.

Таким чином, розроблений застосунок Neural Vision – Transport Analyzer є актуальним і практично значущим рішенням, що відповідає сучасним вимогам до систем автоматизованого аналізу зображень. Поєднання передових методів глибокого навчання з доступним вебінтерфейсом робить його універсальним інструментом як для комерційного використання, так і для подальших наукових досліджень у галузі комп'ютерного зору. Модульна архітектура системи забезпечує легке масштабування та адаптацію до нових вимог без необхідності повного переписування коду, що суттєво знижує витрати на підтримку та розвиток продукту.

Окремої уваги заслуговує відкритість розробленого рішення, відсутність залежності від платних хмарних API та можливість локального розгортання роблять систему привабливою для організацій із підвищеними вимогами до конфіденційності даних. Це особливо важливо в контексті сучасних регуляторних вимог щодо захисту персональних даних, зокрема в транспортній та страховій галузях. Перспективами подальшого розвитку системи є інтеграція підтримки відеопотоків у реальному часі, розширення бази транспортних засобів новими марками та моделями, додавання модуля розпізнавання номерних знаків, а також впровадження механізмів активного навчання для поступового підвищення точності класифікації на основі зворотного зв'язку від користувачів.

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз сучасного стану проблеми

Сучасний транспортний сектор є одним із найбільш динамічно зростаючих та технологічно насичених напрямків людської діяльності. Щороку кількість транспортних засобів у світі невідпинно зростає: за даними міжнародних аналітичних агентств, станом на 2026 рік, динамічно зростаючи, кількість перевищила 1.5 мільярда одиниць автомобільної техніки. Таке зростання породжує цілий спектр нових потреб – в ефективному моніторингу транспортних потоків, автоматизованій ідентифікації транспортних засобів, цифровій каталогізації автопарків та інтелектуальних системах технічного обліку.

Традиційні методи ідентифікації транспортних засобів – ручне введення даних, сканування номерних знаків (ANPR), або ведення паперових реєстрів – не справляються з вимогами сучасних підприємств. Вони потребують значних людських ресурсів, схильні до помилок і не забезпечують достатньої швидкості обробки інформації. Натомість системи комп'ютерного зору, здатні автоматично розпізнавати марку та модель транспортного засобу лише за його фотографією, вирішують ці проблеми радикально і без залучення ручної праці.

Галузі застосування подібних систем надзвичайно широкі. У страховому секторі автоматичне визначення марки й моделі автомобіля дозволяє прискорити оформлення полісів та обробку страхових випадків. Забезпечувати митний контроль транспортних засобів під час перетину державного кордону. У маркетингу та дилерських мережах – аналізувати переваги покупців та будувати персоналізовані пропозиції. У міській інфраструктурі – оптимізувати паркувальні простори та управляти трафіком. У правоохоронних органах – виявляти транспортні засоби, що перебувають у розшуку.

Особливої актуальності набуває проблема розпізнавання транспортних засобів у контексті розвитку розумних міст. Інтеграція камер відеоспостереження

з інтелектуальними системами аналізу зображень дозволяє будувати цифрову карту транспортних потоків у режимі реального часу, що є критичним для оптимізації дорожньої інфраструктури, екологічного моніторингу та безпеки дорожнього руху.

З технічного погляду, розпізнавання транспортних засобів є складним завданням комп'ютерного зору через велику варіативність вхідних даних, на це впливають різні кути зйомки, умови освітлення, пори року, забруднення об'єктів, часткове перекриття транспортного засобу іншими об'єктами. Класичні методи комп'ютерного зору на основі ознак SIFT, HOG та каскадних класифікаторів Хаара виявляються недостатньо точними, за умов реальної експлуатації. Натомість глибокі нейронні мережі, зокрема архітектури на основі згорткових мереж та трансформерів, демонструють значно вищу точність і надійність.

Виникнення та розвиток мультимодальних моделей, які одночасно обробляють текстову та зображувальну інформацію, такі як CLIP від OpenAI [3], відкрило принципово новий підхід до класифікації зображень. На відміну від традиційних класифікаторів, що потребують великих розмічених датасетів для кожного нового класу, мультимодальні моделі здатні класифікувати об'єкти у zero-shot режимі – лише за текстовим описом нового класу, без додаткового тренування. Це робить їх особливо привабливими для прикладних систем, де перелік розпізнаваних об'єктів постійно розширюється.

Паралельно з розвитком класифікаційних моделей еволюціонують і детектори об'єктів. Архітектура YOLO (You Only Look Once), яка вперше з'явилась у 2015 році, пройшла шлях від першої версії до YOLOv8 [1], постійно покращуючи баланс між швидкістю обробки та точністю детекції [2]. Сьогодні він здатний обробляти зображення зі швидкістю понад 100 кадрів на секунду на сучасному GPU, що дозволяє застосовувати його у системах реального часу.

Таким чином, актуальність розробки вебзастосунку Transport Analyzer обумовлена:

- зростаючою потребою бізнесу та держави в автоматизованих системах ідентифікації транспортних засобів;
- технологічною зрілістю методів комп'ютерного зору для вирішення даного класу задач;
- відсутністю загальнодоступних вебінструментів, що поєднують детекцію, класифікацію та повний технічний профіль транспортного засобу в єдиному інтерфейсі;
- розвитком zero-shot класифікації, яка дозволяє будувати гнучкі та розширювані системи без трудомісткого розмічення датасетів.

Для розробки системи обрано галузь автомобільного транспорту, що є найчисельнішим та найрізноманітнішим сегментом наземного транспорту. Об'єктом аналізу виступають фотографії транспортних засобів, зроблені в довільних умовах. Система повинна приймати зображення від кінцевого користувача через вебінтерфейс, обробляти їх за допомогою нейронних мереж та повертати структурований технічний профіль виявленого транспортного засобу протягом декількох секунд. В таблиці 1.1 відображено вимоги обрані для розробки системи.

Таблиця 1.1 – Системні вимоги та інформаційні потоки Transport Analyzer

Категорія	Опис	Значення / Параметр
Тип системи	Вебзастосунок з AI-аналізом зображень	SPA на Flask + Vanilla JS
Вхідні дані	Зображення транспортного засобу	JPG, PNG, BMP, WEBP; до 16 МБ
Вихідні дані	Технічний профіль ТЗ	JSON: марка, модель, тип, bbox, кольори, статистика
Час відгуку	Час аналізу одного зображення	2-5 секунд на CPU
Протокол взаємодії	HTTP REST API	POST /analyze, GET /
Апаратна платформа	Сервер або персональний ПК	CPU / GPU, RAM $\geq$ 8 ГБ
Програмна платформа	ОС та Python-оточення	Windows / Linux, Python 3.10+

Наведені вимоги визначають технічні межі системи та є основою для прийняття архітектурних рішень на всіх етапах розробки. Підтримка поширених форматів зображень і мінімальні апаратні вимоги забезпечують широку доступність системи для кінцевих користувачів.

1.2 Аналіз аналогічних програм, баз даних, систем, обґрунтування вибраного напрямку та вибір методів рішення

Перед проектуванням системи Transport Analyzer було проведено аналіз існуючих рішень у сфері розпізнавання та аналізу транспортних засобів. Розглянуто три найбільш репрезентативні аналоги, що охоплюють різні підходи до вирішення задачі, відображено у таблиці 1.2.

Таблиця 1.2 – Порівняльний аналіз існуючих систем розпізнавання транспортних засобів

Критерій	CarNet.ai	Clarifai	ANPR-системи	Transport Analyzer
Визначення марки/моделі	Так	Так	Частково	Так
Детекція об'єкта (bbox)	Ні	Ні	Ні	Так (YOLO)
Аналіз кольорової палітри	Ні	Ні	Ні	Так (KMeans)
Відкритий вихідний код	Ні	Ні	Ні	Так
Локальне розгортання	Ні	Обмежено	Так	Так
Zero-shot класифікація	Ні	Ні	Ні	Так (CLIP)
Безкоштовне використання	Ні	Обмежено	Ні	Так
Вебінтерфейс для кінцевого користувача	API only	API + Studio	Спеціалізовані	Повний UI
Технічні характеристики ТЗ	Ні	Ні	Ні	Так

CarNet.ai та схожі хмарні сервіси (Imagga, AWS Rekognition Custom Labels, Google Vision API) надають API для розпізнавання марки та моделі автомобіля. Принцип роботи базується на власних навчених CNN-моделях, що класифікують зображення серед фіксованого набору категорій. Переваги:

- висока точність на власному датасеті;
- масштабованість хмарної інфраструктури;
- простота інтеграції через REST API;
- підтримка великої кількості марок і моделей.

Недоліки:

- платна модель використання (оплата за кожен запит);
- залежність від інтернет-з'єднання та доступності хмарного сервісу;
- закритий вихідний код і неможливість розширення або адаптації під конкретні потреби;

- відсутність детальної інформації про технічний профіль автомобіля;
- неможливість локального розгортання без передачі даних третій стороні.

Clarifai – це платформа для машинного навчання, яка серед інших пропонує готові моделі для розпізнавання транспортних засобів. Платформа надає візуальний конструктор для налаштування та донавчання моделей на власних даних. Переваги:

- зручний вебінтерфейс для навчання та тестування моделей;
- підтримка власних датасетів;
- можливість дотренування на специфічних класах;
- детальна документація.

Недоліки:

- комерційна платформа з обмеженим безкоштовним рівнем;
- висока вартість для промислового використання;
- складне налаштування для нетехнічних користувачів;
- відсутність готового кінцевого інтерфейсу для кінцевого користувача, це лише API та конструктор;
- не аналізує кольорову палітру та технічні характеристики.

Plate Recognizer та аналогічні ANPR (Automatic Number Plate Recognition) системи є спеціалізованими рішеннями для розпізнавання номерних знаків і типу транспортного засобу. Використовуються у паркінгах, на контрольно-пропускних пунктах та в системах дорожнього спостереження. Переваги:

- висока точність у задачі розпізнавання номерних знаків;
- підтримка режиму реального часу з відео-потокотом;
- підтримка широкого кола регіональних форматів номерних знаків;
- інтеграція з обладнанням IP-камер.

Недоліки:

- орієнтованість виключно на номерні знаки;
- а не на марку або модель автомобіля за зовнішнім виглядом;
- потребує спеціалізованого обладнання або камер;
- не підходить для аналізу довільних фотографій;
- відсутня візуалізація кольорового профілю та детекція об'єктів із bounding box.

За результатами порівняльного аналізу встановлено, що жоден із розглянутих аналогів не забезпечує повного комплексу функцій, що включає:

- детекцію об'єкта з bounding box;
- двопрохідну класифікацію марки та типу транспорт;
- аналіз кольорової палітри та статистики зображення;
- можливість локального розгортання у вигляді вебзастосунку з зручним інтерфейсом для кінцевого користувача.

Саме тому розробка власного рішення є обґрунтованою та доцільною.

Для вирішення поставленого завдання обрано гібридний підхід, що поєднує: детектор YOLOv8n для локалізації транспортного засобу на зображенні, мультимодальну модель CLIP ViT-B/32 для zero-shot класифікації типу та марки/моделі, алгоритм кластеризації KMeans для аналізу кольорів. Цей підхід дозволяє досягти якісного результату без необхідності збирання та розмічення спеціалізованого датасету.

1.3 Аналіз і вибір засобів проектування сайту для аналізу зображень

Вибір технологічного стеку для розробки Transport Analyzer ґрунтувався на кількох ключових вимогах:

- висока продуктивність AI-ядра;
- мінімальна складність розгортання;
- зручний для кінцевого користувача інтерфейс та можливість подальшого

масштабування.

Python є домінуючою мовою програмування у сфері машинного навчання та штучного інтелекту. Вибір Python обумовлений наступними чинниками:

- найрозвиненіша екосистема бібліотек для ML/DL (PyTorch, TensorFlow, scikit-learn, Hugging Face Transformers, Ultralytics);
- зручний синтаксис та висока продуктивність розробника;
- широка спільнота та обширна документація;
- підтримка Python 3.10+ з типізацією та сучасними мовними конструкціями;
- нативна інтеграція з CUDA для GPU-прискорення.

Версія Python 3.11 використовується у проекті як найбільш стабільна та продуктивна на момент розробки [5]. Python 3.11 демонструє приріст продуктивності до 25 % порівняно з Python 3.10 за рахунок оптимізацій інтерпретатора CPython.

Для реалізації серверної частини обрано мікрофреймворк Flask версії 3.0 [4]. Розглядалися також альтернативи: Django та FastAPI. Порівняння технологій відображено в таблиці 1.3.

Таблиця 1.3 – Порівняння технологій серверної частини

Критерій	Flask	FastAPI	Django
Складність	Мінімальна	Середня	Висока
Асинхронність	Ні (sync)	Так (async)	Частково
Продуктивність розробки	Висока	Висока	Середня
ORM, Auth, Admin	Ні (ext.)	Ні (ext.)	Так (вбудовано)
Доцільність для AI API	Висока	Висока	Низька
Вибір для проекту	✓ Обрано	–	–

Django є повнофункціональним вебфреймворком із вбудованою ORM, адмін-панеллю, системою аутентифікації та багатьма іншими компонентами. Проте для даного проекту, де основна бізнес-логіка зосереджена в AI-ядрі, а не у вебфункціональності, надлишкова складність Django є зайвою. FastAPI забезпечує автоматичне генерування OpenAPI-документації та асинхронну обробку запитів

на основі `asyncio`, що робить його популярним вибором для мікросервісів. Однак, оскільки основні операції AI-аналізу є синхронними і займають 2-5 секунд (блокуючи виклики до `PyTorch/Ultralytics`), переваги асинхронності `FastAPI` у даному контексті нівелюються.

`Flask` обрано з наступних міркувань:

- мінімалістична архітектура без зайвих залежностей;
- прозора маршрутизація (`route decorators`);
- легке налаштування передачі файлів через `werkzeug`;
- швидкий старт і мінімум `boilerplate`-коду;
- достатня функціональність для реалізації двох кінцевих точок API (`GET/POST /analyze`).

`PyTorch` є провідним фреймворком глибокого навчання, розробленим Meta AI [7]. Вибір `PyTorch` обумовлений тим, що більшість сучасних передових моделей (`CLIP`, `GPT`, `LLaMA`, `YOLOv8`) первинно реалізуються саме під `PyTorch`. `Hugging Face Transformers` – це бібліотека, що надає уніфікований інтерфейс до тисяч мінімально тренованих моделей через `Hub`, включно з `CLIP ViT-B/32`, що використовується у проекті.

`PyTorch` забезпечує динамічний граф обчислень для зручного налагодження, автоматичну підтримку GPU через `CUDA` без змін у коді (`torch.device`), ефективні тензорні операції, широку підтримку в наукових публікаціях та референсних реалізаціях моделей.

`Ultralytics YOLOv8` – це сучасний фреймворк для задач детекції, сегментації та класифікації об'єктів. Бібліотека `ultralytics` надає простий Python API для завантаження та використання попередньо навчених моделей `YOLOv8`.

Для проекту обрано `YOLOv8n (nano)` – найлегшу конфігурацію, що забезпечує прийнятну точність при мінімальних вимогах до обчислювальних ресурсів. `YOLOv8n` навчена на датасеті `COCO`, що містить 80 класів об'єктів, серед яких 6 транспортних класів, необхідних для системи.

Порівняння версій `YOLOv8` за параметрами показано в таблиці 1.4.

Таблиця 1.4 – Порівняння конфігурацій YOLOv8

Конфігурація	Параметрів (млн)	mAP50-95 (COCO)	Швидкість (ms/img, CPU)	Розмір моделі
YOLOv8n	3.2	37.3	~80 мс	6.3 МБ
YOLOv8s	11.2	44.9	~128 мс	21.5 МБ
YOLOv8m	25.9	50.2	~234 мс	49.7 МБ
YOLOv8l	43.7	52.9	~375 мс	83.7 МБ
YOLOv8x	68.2	53.9	~479 мс	130.5 МБ

Для даного проекту обрано YOLOv8n як оптимальний баланс між швидкістю (80 мс на CPU) та точністю (mAP50-95 = 37.3). Оскільки YOLO використовується виключно для локалізації транспортного засобу, а не для кінцевої класифікації марки та моделі, вимоги до точності детекції є помірними, важливіше виявити об'єкт взагалі, ніж максимально точно його класифікувати.

CLIP (Contrastive Language-Image Pretraining) – мультимодальна нейронна мережа, розроблена OpenAI у 2021 році. Модель складається з двох кодерів: зображального (Vision Transformer, ViT) та текстового (Transformer). У процесі початкового тренування обидва кодери навчались узгоджено на 400 мільйонах пар зображення-текст з інтернету.

Ключова властивість CLIP – здатність до zero-shot класифікації, для розпізнавання нового класу об'єктів достатньо надати текстовий опис, не потребуючи жодного прикладу зображення. Це принципово відрізняє CLIP від традиційних CNN-класифікаторів, що вимагають сотень або тисяч прикладів кожного класу.

У проекті використовується ViT-B/32, де В означає базовий розмір моделі [6], а 32 – розмір патчів (patch size) у пікселях, на які розбивається зображення. Зображення розміром 224×224 пікселів розбивається на $7 \times 7 = 49$ патчів розміром 32×32 пікселі, кожен з яких обробляється як токен трансформера [9].

Scikit-learn – провідна Python-бібліотека для класичного машинного навчання. Для аналізу кольорової палітри обрано алгоритм кластеризації KMeans, реалізований у scikit-learn. Алгоритм ефективно знаходить N домінуючих

кольорів зображення, мінімізуючи суму квадратів відстаней пікселів до центроїдів кластерів.

Клієнтська частина реалізована без використання JavaScript-фреймворків. Рішення обумовлено такими міркуваннями:

- мінімальний розмір клієнтського пакету та відсутність залежностей від npm-пакетів;
- відсутність необхідності у складній клієнтській маршрутизації (застосунок є однією сторінкою);
- використання Canvas API для малювання bounding box потребує прямого доступу до DOM-елемента, що більш природньо у Vanilla JS;
- drag-and-drop завантаження файлів реалізується через стандартні Web API без бібліотек;

Мінімальні вимоги до апаратного та програмного забезпечення для розгортання Transport Analyzer визначено з урахуванням вимог всіх компонентів технологічного стеку вказано у таблиці 1.5.

Таблиця 1.5 – Вимоги до апаратно-програмного комплексу

Компонент	Мінімальні вимоги	Рекомендовані параметри
Процесор	Intel Core i5 / AMD Ryzen 5 (4 ядра)	Intel Core i7+ / AMD Ryzen 7+ (8+ ядер)
Оперативна пам'ять	8 ГБ RAM	16 ГБ RAM і більше
Відеокарта	Не обов'язково (CPU mode)	NVIDIA GPU з 4+ ГБ VRAM (CUDA)
Дисковий простір	5 ГБ вільного місця	20+ ГБ (для кешу)
ОС	Windows 10 / Ubuntu 20.04 LTS	Ubuntu 22.04 LTS / Windows 11
Python	Python 3.10	Python 3.11
Браузер	Chrome 90+ / Firefox 88+ / Edge 90+	Останні версії Chrome / Firefox

Дотримання наведених вимог гарантує стабільну роботу системи в режимі реального часу. За наявності NVIDIA GPU з підтримкою CUDA час аналізу одного зображення суттєво скорочується порівняно з CPU-режимом, що особливо відчутно при обробці великої кількості запитів.

1.4 Постановка завдання на кваліфікаційну роботу бакалавра

Спираючись на результати аналізу предметної області, огляду аналогів та обраного технологічного стеку, метою кваліфікаційної роботи бакалавра є проектування та реалізація вебзастосунку Transport Analyzer для автоматичного аналізу зображень транспортних засобів із використанням нейронних мереж комп'ютерного зору.

Для досягнення поставленої мети сформульовано такі конкретні цілі:

- провести аналіз сучасного стану проблеми автоматичного розпізнавання та класифікації транспортних засобів за зображеннями, визначити недоліки існуючих рішень та обґрунтувати доцільність розробки власної системи;
- спроектувати двохпрохідну архітектуру AI-ядра на основі YOLOv8n та CLIP ViT-B/32, що забезпечує ідентифікацію транспортного засобу без необхідності навчання на спеціалізованому датасеті;
- реалізувати модуль аналізу кольорової палітри на основі алгоритму KMeans (scikit-learn) з визначенням домінантних кольорів, їх частки у зображенні та назви кольору у системі HSV;
- розробити REST API на базі Flask, що приймає зображення транспортних засобів формату JPG/PNG/BMP/WEBP розміром до 16 МБ і повертає структурований JSON-профіль: марка, модель, тип, bbox-координати, топ-3 кандидати, кольорова палітра, статистика зображення;
- реалізувати адаптивний вебінтерфейс у кіберпанк-стилістиці з підтримкою drag-and-drop завантаження зображень, відображенням bounding box через Canvas API, анімованим лоадером з відображенням поточного кроку аналізу та візуалізацією результатів з прогрес-барами впевненості;
- провести тестування системи на різних типах транспортних засобів та в різних умовах зйомки, визначити мінімальні системні вимоги та задокументувати API та процес розгортання;
- виконати ергономічну оцінку інтерфейсу користувача, провести SEO-

оптимізацію, розрахувати техніко-економічні показники розробки та описати механізми захисту інформації, що реалізовані в системі.

Очікуваним результатом є функціональний вебзастосунок Transport Analyzer, готовий до локального розгортання, що демонструє практичне застосування методів комп'ютерного зору, детекції об'єктів, мультимодальної класифікації, кластеризації кольорів, для ідентифікації транспортних засобів за фотографією.

Практична цінність визначається не лише реалізацією застосунку, а й відтворюваністю та масштабованістю підходу, архітектура YOLOv8n + CLIP може бути адаптована для суміжних задач без необхідності збору спеціалізованих датасетів або перенавчання моделей. Модуль аналізу кольорової палітри може використовуватись як самостійний інструмент у задачах, де колір є значущою ознакою – наприклад, у системах пошуку транспортних засобів за описом. Результати роботи мають як прикладне, так і методичне значення для подальших досліджень у галузі комп'ютерного зору та інтелектуальних вебсистем.

РОЗДІЛ 2

СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Обґрунтування вибору шляхів, технологій (алгоритмів) і засобів вирішення поставленого завдання

На основі результатів першого розділу та аналізу методів комп'ютерного зору, доступних для вирішення задачі автоматичної ідентифікації транспортних засобів, обрано архітектурне рішення, що базується на комбінації трьох незалежних алгоритмів, кожен з яких виконує власну функцію в загальному конвеєрі обробки зображення.

Задача детекції транспортного засобу на зображенні вирішується за допомогою архітектури YOLO (You Only Look Once). Порівняно з альтернативними підходами, такими як двоетапні детектори (R-CNN, Faster R-CNN) [8], YOLO демонструє значно вищу швидкість обробки за незначно меншої точності.

Двоетапні детектори спочатку генерують регіони-кандидати (Region Proposal Network), а потім класифікують кожен регіон окремо. Це призводить до часу обробки 100–500 мс на GPU, що неприйнятно для вебзастосунку. YOLO натомість вирішує задачу детекції як єдину регресійну задачу – за один прохід через мережу отримується повний набір bounding box із класами та впевненостями.

Перевага обраного підходу підтверджується порівняльними характеристиками:

- YOLOv8n виконує інференс за 80 мс на CPU проти 200+ мс у Faster R-CNN;
- модель навчена на COCO і розпізнає 6 необхідних транспортних класів без дотренування [10];
- Ultralytics YOLOv8 має простий Python API – 3 рядки коду для ініціалізації та детекції;

– вагові файли yolov8n.pt розміром лише 6.3 МБ не потребують значного дискового простору.

Основним технологічним рішенням, що відрізняє Transport Analyzer від традиційних підходів, є застосування zero-shot класифікації на основі CLIP. Розглянемо чому цей підхід є доцільним.

Традиційний підхід до класифікації марки та моделі автомобіля передбачає: збір датасету (тисячі фотографій для кожного класу), розмічення (ручне або напівавтоматичне призначення міток), навчання або fine-tuning CNN-класифікатора (ResNet, EfficientNet, ViT). Цей підхід потребує значних ресурсів: збір лише 100 класів по 500 зображень – це 50 000 фотографій, які необхідно розмітити. При розширенні системи на нові моделі автомобілів весь процес повторюється.

Zero-shot підхід з CLIP вирішує ці проблеми кардинально. Для додавання нового класу в систему достатньо додати текстовий опис у словник BRAND_CANDIDATES – наприклад «Renault Duster crossover compact SUV», «Renault Duster». Жодного зображення, жодного навчання не потрібно. CLIP порівнює векторне представлення зображення з векторними представленнями всіх текстових описів і обирає найбільш подібний.

Після обчислення схожості для всіх текстових описів результати нормалізуються через softmax для отримання розподілу ймовірностей. Клас з найвищою ймовірністю стає результатом класифікації.

Важливим архітектурним рішенням є розбиття класифікації на два послідовні проходи (passes) замість одного. Якщо порівнювати зображення одразу з усіма приблизно 200 описами марок і моделей, то набір кандидатів містить представників різних типів транспорту (легковики, вантажівки, мотоцикли, автобуси тощо). CLIP може плутати зовні схожі об'єкти різних типів. Натомість двопрохідна стратегія:

– прохід 1 визначає тип транспорту серед 13 категорій (car, suv, sports_car, pickup, minivan, truck, bus, motorcycle, tram, train, special, emergency, bicycle). Пул

порівняно невеликий, тип легко відрізняється візуально;

– прохід 2 знаючи тип, система звужує пошук до відповідного підсловника. Наприклад, для типу «car» порівнюється лише приблизно 32 описи легкових автомобілів. Це підвищує точність і зменшує обчислювальне навантаження.

Ефективність двопрхідного підходу підтверджується на практиці: для автомобіля типу SUV порівняння проводиться лише з 30 SUV-кандидатами замість 200+ загальних, що зменшує ймовірність помилкової активації непов'язаного класу.

Для аналізу кольорової палітри зображення обрано алгоритм KMeans із бібліотеки scikit-learn. Альтернативами є: медіанне скорочення кольорів (Median Cut), алгоритм DBSCAN, гістограмний метод.

KMeans обрано з таких міркувань: відома та контрольована кількість кластерів ($N=5$ домінантних кольорів), швидкість роботи на зменшеному зображенні 120×120 пікселів, інтерпретабельність центроїдів як домінантних кольорів, стабільність результатів при фіксованому `random_state`.

Попередня обробка зображення (зменшення до 120×120) дозволяє обробляти лише 14 400 пікселів замість мільйонів у повнорозмірному зображенні, що прискорює кластеризацію у десятки разів без помітної втрати якості визначення домінантних кольорів.

Визначення назви кольору реалізовано через конвертацію з простору RGB у HSV (Hue-Saturation-Value). У HSV-просторі відтінок (Hue) кодує колір у діапазоні $0-360^\circ$, що дозволяє побудувати просту систему правил для розпізнавання 11 базових кольорів: чорний, білий, сірий, червоний, помаранчевий, жовтий, зелений, блакитний, синій, фіолетовий, рожевий.

Обрана архітектура конвеєру обробки даних забезпечує такі переваги порівняно з альтернативними підходами:

– розділення детекції та класифікації. YOLO виконує лише локалізацію (знаходить bounding box), а CLIP виконує класифікацію вирізаного фрагменту. Це дозволяє незалежно оновлювати кожен компонент;

- гнучкість і розширюваність. Завдяки zero-shot підходу CLIP, система легко розширюється на нові марки і моделі без будь-якого перенавчання – достатньо додати новий рядок у словник кандидатів;
- інференс без GPU. Всі три компоненти (YOLOv8n, CLIP ViT-B/32, KMeans) здатні працювати виключно на CPU, що робить систему доступною без спеціалізованого обладнання;
- кешування моделей. Використання `functools.lru_cache` гарантує, що моделі завантажуються лише один раз при першому запиті, а всі наступні запити використовують моделі з RAM без повторної ініціалізації.

2.2 Функціонально-структурна схема роботи об'єкта проектування

Система Transport Analyzer складається з трьох логічних рівнів: рівня представлення (Frontend), рівня бізнес-логіки (Backend + AI-ядро) та рівня зберігання (файлова система). Нижче описано цілі, функціональні характеристики системи та її компонентну архітектуру.

Серверна частина побудована на основі Flask і зосереджена у файлі `app.py`, який виконує маршрутизацію HTTP-запитів та контроль всього процесу обробки. Для виконання допоміжних операцій `app.py` звертається до модуля `utils.py`, що містить функції аналізу кольорів та обчислення статистичних характеристик зображення. Модуль `model.py` є центральним компонентом системи та реалізує повний конвеєр AI-аналізу – від детекції транспортного засобу до визначення його типу, марки та моделі. Модуль використовує попередньо навчену модель YOLOv8 (`yolov8n.pt`), вагові файли якої завантажуються один раз та кешуються, а також отримує оброблені зображення з директорії `uploads/`.

Окремо на схемі (рис. 2.1) виділено контекст розгортання (Deployment Context): ізольоване віртуальне середовище `.venv`, яке ініціалізується на основі файлу `requirements.txt`, що фіксує всі залежності проекту та забезпечує відтворюваність середовища виконання. Ліворуч на схемі відображено структуру

проекту з усіма ключовими файлами та директоріями, що відповідає їх реальному розташуванню у файловій системі.

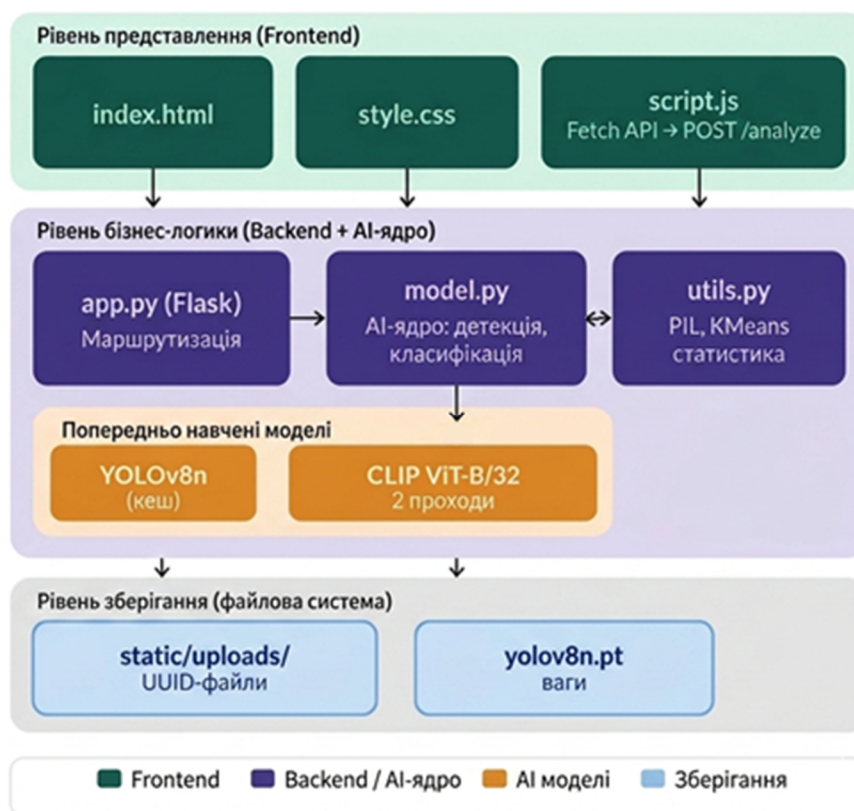


Рисунок 2.1 – Архітектура системи Transport Analyzer

Джерело: розроблено автором

Система Transport Analyzer реалізує такі цілі:

- забезпечення автоматичної детекції транспортного засобу на зображенні та побудова bounding box;
- ідентифікація типу транспортного засобу (легковик, SUV, вантажівка тощо) та марки й моделі;
- визначення кольорової палітри зображення та статистичних характеристик (розмір, яскравість, контраст);
- надання зручного вебінтерфейсу для завантаження зображень та перегляду результатів аналізу;
- забезпечення REST API для інтеграції з зовнішніми системами.

Система реалізує такі основні функції:

- приймання зображень транспортних засобів через вебформу (drag-and-drop або кнопка вибору файлу);
- валідація типу та розміру завантаженого файлу (JPG, PNG, BMP, WEBP, до 16 МБ);
- генерація унікального UUID-імені для кожного збереженого файлу з метою уникнення конфліктів;
- запуск конвеєру AI-аналізу: YOLO детекція, CLIP прохід 1 (тип), CLIP прохід 2 (марка/модель), KMeans (кольори), PIL (статистика);
- формування та повернення JSON-відповіді з повним технічним профілем транспортного засобу;
- відображення результатів у вебінтерфейсі з bounding box, прогрес-барами впевненості та кольоровою палітрою;
- автоматичне очищення папки завантажень (зберігаються лише останні 20 файлів).

Загальний потік обробки зображення у системі Transport Analyzer можна описати наступними кроками:

- користувач обирає файл зображення у браузері та надсилає його на сервер через POST-запит до /analyze;
- Flask валідує файл (розширення, розмір) та зберігає його з UUID-ім'ям у static/uploads/;
- функція analyze_vehicle(filepath) запускає конвеєр AI-аналізу з модуля model.py;
- YOLOv8n виконує детекцію: знаходить усі транспортні об'єкти, обирає найбільший за площею, повертає координати bounding box;
- з оригінального зображення вирізається фрагмент за bbox-координатами з відступом 20px. Якщо YOLO не знайшов об'єкту – передається повне зображення;
- CLIP прохід 1: вирізаний фрагмент порівнюється з 13 текстовими описами типів транспорту. Визначається тип з найвищою cosine similarity;
- CLIP прохід 2: на основі визначеного типу обирається вибірка кандидатів.

Зображення порівнюється з описами марок та моделей. Повертається топ-3 з відсотками впевненості;

– паралельно запускаються: `get_colors_data()` – KMeans кластеризація кольорів; `get_image_stats()` – PIL-статистика зображення;

– `app.py` збирає всі результати у єдину JSON-структуру та повертає відповідь;

– `script.js` отримує відповідь, малює bounding box через Canvas API [12] та рендерить результати з анімованими прогрес-барами.

Нижче в таблиці 2.1 наведено схему взаємодії компонентів системи (UML Component Diagram).

Таблиця 2.1 – Компонентна архітектура системи Transport Analyzer

Компонент	Відповідальність	Інтерфейс / Залежності
index.html + style.css	Розмітка та стилі інтерфейсу. Drag-and-drop зона, ладер, блок результатів.	Canvas API, FormData API
script.js	Клієнтська логіка: надсилання запитів, малювання bbox, рендер результатів.	Fetch API → POST /analyze
app.py (Flask)	HTTP-сервер: маршрутизація, валідація файлів, збирання результатів, очищення uploads.	model.py, utils.py, werkzeug
model.py (AI-ядро)	Конвеєр: YOLO детекція + CLIP 2-pass класифікація + словники кандидатів.	ultralytics, transformers, torch, PIL
utils.py	KMeans аналіз кольорів, PIL статистика зображення.	scikit-learn, PIL, numpy, colorsys
static/uploads/	Тимчасове сховище завантажених зображень (до 20 файлів).	Файлова система ОС
yolov8n.pt	Вагові файли YOLOv8n. Завантажуються один раз і кешуються через <code>lru_cache</code> .	Ultralytics YOLO API
CLIP ViT-B/32 (HuggingFace)	Попередньо навчена модель CLIP (~600 МБ). Кешується локально.	transformers.CLIPModel, CLIPProcessor

Наведена архітектура забезпечує чітке розмежування відповідальності між компонентами системи. Клієнтська частина відповідає за взаємодію з

користувачем, тоді як серверна виконує всю обчислювальну логіку. Використання кешування через `lru_cache` суттєво скорочує час відповіді при повторних запитах.

Послідовністю взаємодії між компонентами під час аналізу одного зображення, показано в таблиці 2.2.

Таблиця 2.2 – Опис кроків аналізу зображення

№	Ініціатор	Отримувач	Повідомлення / Дія
1	Користувач	Browser	Перетягує або обирає файл зображення
2	Browser	Flask /analyze	POST multipart/form-data з файлом
3	Flask	FileSystem	Зберігає файл як <code>UUID.ext</code> у <code>static/uploads/</code>
4	Flask	model.py	Виклик <code>analyze_vehicle(filepath)</code>
5	model.py	YOLOv8n	<code>detector(img_path)</code> → список детекцій з класами та <code>bbox</code>
6	model.py	PIL	Вирізання фрагменту за <code>bbox</code> з відступом 20px
7	model.py	CLIP	Pass 1: порівняння з 13 текстами типів → визначення <code>vehicle_type</code>
8	model.py	CLIP	Pass 2: порівняння з підсловником марок → топ-3 results
9	Flask	utils.py	<code>get_colors_data()</code> → KMeans 5 кластерів → <code>hexes</code> , <code>names</code> , <code>pcts</code>
10	Flask	utils.py	<code>get_image_stats()</code> → PIL <code>ImageStat</code> → <code>width</code> , <code>height</code> , <code>brightness</code> , <code>contrast</code>
11	Flask	Browser	JSON-відповідь: <code>success</code> , <code>result</code> , <code>colors</code> , <code>stats</code> , <code>image_url</code>
12	Browser	Canvas API	Малювання <code>bounding box</code> з підписом марки та <code>bbox</code> -координатами
13	Browser	DOM	Рендер результатів: топ-3 з прогрес-барами, кольорова палітра, статистика
14	Flask	FileSystem	<code>_cleanup_uploads()</code> : видалення найстаріших файлів, якщо > 20

Наведена послідовність демонструє повний цикл обробки запиту – від завантаження файлу користувачем до відображення результатів у браузері. Кожен крок чітко розмежований між відповідними компонентами, що забезпечує передбачувану та керовану послідовність виконання. Автоматичне очищення сховища гарантує стабільну роботу системи при багаторазових запитах.

2.3 Створення моделі бази даних

Специфіка системи Transport Analyzer полягає в тому, що вона є застосунком без збереження стану для обробки зображень: кожен запит є незалежним, і між запитами не зберігається контекст або стан сесії. З точки зору постійного зберігання даних, система використовує виключно файлову систему для тимчасового зберігання завантажених зображень.

У поточній реалізації відсутня реляційна або NoSQL база даних, оскільки:

- система не потребує зберігання результатів аналізу між сесіями – кожен запит є самодостатнім;
- не реалізовано систему аутентифікації користувачів, яка традиційно потребує БД;
- зберігання зображень є тимчасовим – автоматичне очищення видаляє файли старіші за 20 останніх;
- відсутній журнал запитів або накопичення аналітичних даних.

Проте для можливого розширення системи (версія 2.0) доцільним є впровадження бази даних для реалізації функцій журналювання запитів і результатів аналізу, зберігання профілів користувачів, збереження та повторного перегляду попередніх аналізів. Нижче описано проектне рішення для потенційного розширення.

Для можливого розширення системи пропонується використання SQLite як основної СКБД на етапі розробки та тестування, з подальшим міграцією на PostgreSQL для продуктивного середовища.

SQLite обрано для локального розгортання з таких причин:

- вбудована в Python через стандартну бібліотеку sqlite3;
- не потребує окремого серверного процесу;
- файл бази даних – це єдиний .db файл, що спрощує резервне копіювання;
- достатня продуктивність для однокористувацького або малочисельного застосунку.

PostgreSQL – для продуктивного розгортання: підтримка JSON-стовпців (JSONB) дозволяє зберігати result-об’єкт цілком без нормалізації; ACID-транзакції; підтримка повнотекстового пошуку; масштабованість до тисяч одночасних запитів.

Концептуальна, інфологічна модель (табл. 2.3) даних системи Transport Analyzer в розширеній версії включатиме наступні сутності: AnalysisRequest (запит на аналіз), VehicleResult (результат ідентифікації), ColorProfile (кольоровий профіль), ImageStats (статистика зображення).

Таблиця 2.3 – Інфологічна модель сутності та атрибутів

Сутність	Атрибути	Опис
AnalysisRequest	id, filename, upload_time, elapsed_ms, image_url	Метадані кожного запиту на аналіз зображення
VehicleResult	id, request_id, label, confidence, vehicle_type, category, bbox_json, top3_json	Результат ідентифікації ТЗ: марка, модель, тип, bbox, топ-3
ColorProfile	id, request_id, hex_color, color_name, percentage, rank	До 5 записів на кожен запит – домінантні кольори зображення
ImageStats	id, request_id, width, height, megapixels, brightness, contrast	Технічні характеристики зображення

Таблиця 2.4 відображає первинні та зовнішні ключі (request_id).

Таблиця 2.4 – Фізична структура таблиці analysis_requests

Поле	Тип даних	Обмеження	Опис
id	INTEGER	PK, AI	Унікальний ідентифікатор запиту
filename	VARCHAR(255)	NOT NULL	Оригінальне ім’я файлу
uuid_name	VARCHAR(50)	UNIQUE	UUID-ім’я збереженого файлу
upload_time	DATETIME	NOT NULL	Час завантаження файлу
elapsed_ms	FLOAT		Час аналізу у секундах
image_url	VARCHAR(500)	NOT NULL	URL для доступу до зображення

Технологія доступу до бази даних реалізується через SQLAlchemy ORM (для Flask-застосунків) або напряму через sqlite3 (стандартна бібліотека Python). SQLAlchemy забезпечує незалежність від конкретної СКБД – перехід від SQLite до PostgreSQL потребує лише зміни рядка підключення (connection string) без переписування логіки запитів. Рядок підключення для SQLite має вигляд `sqlite:///transport_analyzer.db`, тоді як для PostgreSQL використовується формат

postgres://user:password@host:port/dbname. Завдяки цьому середовище розробки та продуктивне середовище можуть використовувати різні СКБД без будь-яких змін у бізнес-логіці застосунку.

При використанні SQLAlchemy ORM кожна таблиця відображається у вигляді Python-класу, що успадковує базовий клас Base. Поля таблиці описуються як атрибути класу з відповідними типами та обмеженнями, а операції читання та запису виконуються через об'єкт сесії (Session). Такий підхід підвищує читабельність коду, спрощує тестування окремих компонентів системи та мінімізує ризик SQL-ін'єкцій завдяки автоматичній параметризації запитів.

Основні SQL-запити, що будуть використовуватися у розширеній версії системи, наведено нижче. Після успішного завершення аналізу зображення система зберігає метадані запиту до таблиці analysis_requests, фіксуючи ім'я файлу, унікальний ідентифікатор, час завантаження, тривалість обробки та URL зображення (ліст. 2.1).

Лістинг 2.1 – Збереження нового запиту на аналіз

```
INSERT INTO analysis_requests (filename, uuid_name, upload_time,
elapsed_ms, image_url)
VALUES (?, ?, datetime('now'), ?, ?);
```

кінець лістингу 2.1

Для відображення історії роботи системи використовується запит із об'єднанням таблиць analysis_requests та vehicle_results за зовнішнім ключем request_id. Операція об'єднання дозволяє отримати зведену інформацію одночасно про метадані запиту та результат розпізнавання транспортного засобу в рамках однієї вибірки, уникаючи необхідності виконання кількох послідовних запитів до бази даних. Як відображено у лістингу 2.2, вибірка обмежується десятьма останніми записами, впорядкованими за часом завантаження у спадному порядку, що дозволяє оперативно отримати актуальні результати розпізнавання. Обмеження кількості записів через LIMIT 10 є свідомим архітектурним рішенням, спрямованим на зниження навантаження на базу даних при великому обсязі

накопичених даних, оскільки для інтерфейсу користувача достатньо відображення останніх актуальних результатів.

Лістинг 2.2 – Отримання останніх 10 аналізів з результатами

```
SELECT ar.id, ar.filename, ar.upload_time, vr.label, vr.confidence,  
vr.vehicle_type  
FROM analysis_requests ar  
JOIN vehicle_results vr ON vr.request_id = ar.id  
ORDER BY ar.upload_time DESC LIMIT 10;
```

кінець лістингу 2.2

Результатом виконання запиту є зведений набір даних, що містить ідентифікатор запиту, ім'я файлу, час завантаження, мітку розпізнаного транспортного засобу, рівень впевненості моделі та тип транспортного засобу. Поле `confidence` дозволяє оцінити достовірність кожного окремого результату, що є корисним при подальшому аналізі якості розпізнавання. Зокрема, результати з низьким значенням впевненості (нижче 0.5) можуть бути відфільтровані або позначені як такі, що потребують додаткової перевірки, що підвищує загальну надійність системи. Поле `vehicle_type` у свою чергу дозволяє групувати результати за категоріями транспортних засобів – наприклад, окремо аналізувати статистику розпізнавання легкових автомобілів, вантажівок або мотоциклів, що є корисним для формування агрегованої звітності у розширеній версії системи.

РОЗДІЛ 3

РОЗРОБКА САЙТУ ДЛЯ АНАЛІЗУ ЗОБРАЖЕНЬ

3.1 Практична реалізація об'єкта проектування. Побудова блок-схем модулів програми

Система Neural Vision – Transport Analyzer реалізована як повноцінний вебзастосунок з чіткою трирівневою архітектурою: клієнтська частина (Frontend), серверна частина (Backend) та модуль штучного інтелекту (AI Core). Розробка велась на мові Python з використанням мікрофреймворку Flask для організації серверної логіки та маршрутизації HTTP-запитів.

Структура проекту містить такі ключові файли та директорії, наведено у таблиці 3.1.

Таблиця 3.1 – Структура файлів проекту Neural Vision

Файл / директорія	Призначення	Мова / технологія
app.py	Flask-сервер, маршрути, обробка запитів	Python
model.py	AI-ядро: YOLO + CLIP, словники кандидатів	Python
utils.py	Допоміжні функції: кольори, статистика	Python
yolov8n.pt	Вагові файли моделі YOLOv8n	Binary
templates/index.html	Єдина HTML-сторінка застосунку	HTML5
static/style.css	Стили інтерфейсу (кіберпанк-стилістика)	CSS3
static/script.js	Клієнтська логіка, Canvas API, fetch	JavaScript
static/uploads/	Тимчасове сховище завантажених файлів	—
requirements.txt	Залежності Python-середовища	Text

Серверна частина (app.py) обробляє два HTTP-маршрути. Маршрут GET/ повертає головну сторінку застосунку. Маршрут POST /analyze приймає зображення у форматі multipart/form-data, зберігає його в папку static/uploads/ під унікальним UUID-іменем, передає на аналіз та повертає результат у форматі JSON. Максимальний розмір файлу обмежений 16 МБ, підтримуються формати JPG, JPEG, PNG, BMP, WEBP. Автоматичне очищення сховища забезпечує зберігання лише останніх 20 файлів.

AI-ядро відображене в додатку А реалізує двоетапну класифікацію з використанням двох нейронних мереж. Функція `analyze_vehicle()` приймає шлях до файлу і повертає структурований словник з результатами аналізу. Обидві нейронні мережі завантажуються один раз при першому запиті та кешуються в оперативній пам'яті за допомогою декоратора `functools.lru_cache`, що суттєво прискорює обробку наступних запитів.

Допоміжний модуль `utils.py` містить функцію `get_colors_data()`, яка виконує кластеризацію пікселів зображення методом KMeans для визначення домінантних кольорів, та функцію `get_image_stats()`, яка повертає технічну статистику зображення: розміри, яскравість, контраст та співвідношення сторін [10].

Фізичну організацію файлів проекту у середовищі розробки Visual Studio Code відображено на рисунку 3.1. Структура проекту побудована відповідно до стандартних угод Flask-застосунків і забезпечує чіткий розподіл відповідальності між компонентами. HTML-шаблони зберігаються в директорії `templates/` і містять розмітку сторінки з посиланнями на статичні ресурси. Статичні файли – таблиці стилів CSS та JavaScript-скрипти – розміщені в директорії `static/` і завантажуються браузером безпосередньо, без участі серверної логіки. Зображення, що завантажуються користувачами під час роботи з застосунком, автоматично зберігаються у піддиректорії `static/uploads/` і доступні для відображення у браузері через статичний URL. Такий підхід до організації коду спрощує супровід проекту та відповідає загальноприйнятим практикам веброзробки на Python.

Загалом, описана архітектура відповідає принципу розділення відповідальності, де кожен модуль виконує чітко визначену функцію: `app.py` відповідає за маршрутизацію та обробку HTTP-запитів, `model.py` інкапсулює логіку роботи з нейронними мережами, а `utils.py` надає допоміжні інструменти для обробки зображень. Такий підхід спрощує тестування та подальше масштабування системи, оскільки внесення змін до окремого компонента не впливає на роботу решти модулів

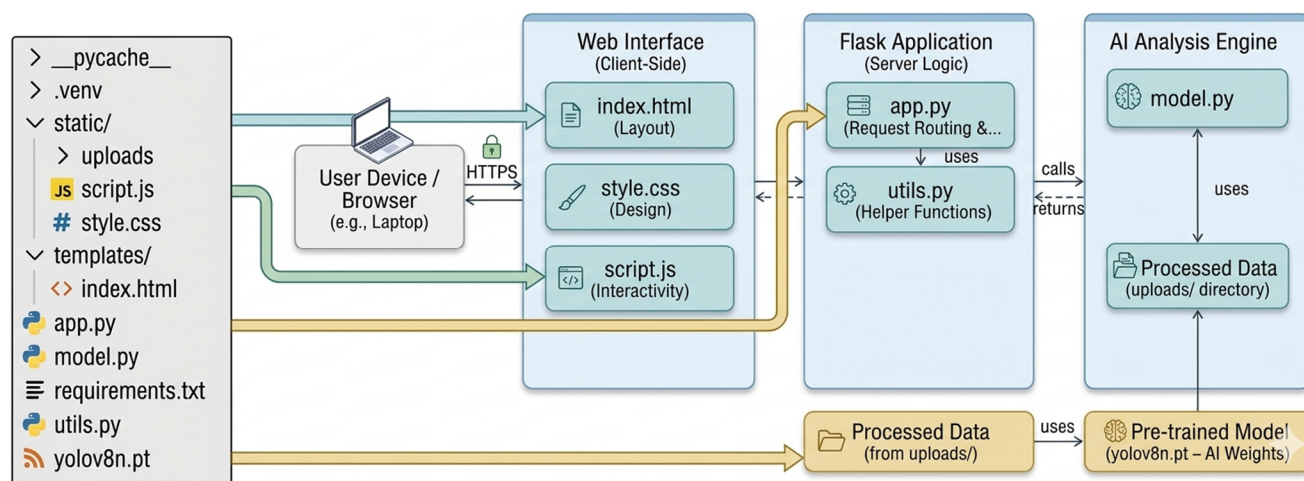


Рисунок 3.1 – Структура проекту Transport Analyzer у середовищі VS Code

Джерело: розроблено автором

Фронтенд розробка цієї програми реалізована у форматі SPA (Single Page Application), без перезавантажень браузера. Розмітка документа організована за семантичним принципом і поділена на функціональні секції: заголовок із описом, зону завантаження файлу, блок попереднього перегляду, індикатор завантаження та блок результатів. Усі секції присутні в DOM одночасно, але відображаються або приховуються програмно через властивість `display` засобами JavaScript. Такий підхід дозволяє уникнути мережових затримок між переходами та забезпечує плавний користувацький досвід без перемикання між сторінками.

Структурну основу зони завантаження наведено у лістингу 3.1. Елемент `.drop-zone` є інтерактивним контейнером, що реагує на події перетягування файлів та кліку миші. Прихований елемент `input type = file` активується програмно при кліку на кнопку, що дозволяє повністю кастомізувати зовнішній вигляд елемента завантаження без обмежень стандартного браузерного стилю. Канвас-елемент `bbox-overlay` розташований абсолютно поверх зображення і використовується для відображення рамки детекції через Canvas API.

Лістинг 3.1 – Структура зони завантаження та прев'ю (`index.html`)

```
<section class="upload-section">
  <div class="drop-zone" id="dropZone">
    <span class="drop-icon">📁 </span>
```

```

    <p class="drop-title">Перетягніть фото сюди</p>
    <p class="drop-hint">або <span>оберіть файл</span> на
пристрої</p>
    <button class="btn-primary"
      onclick="document.getElementById('fileInput').click()">
      <span>📁</span> Обрати зображення
    </button>
    <input type="file" id="fileInput" accept="image/*" />
  </div>
</section>

<section id="previewSection">
  <div class="preview-wrap">
    <img id="previewImg" src="" alt="Preview" />
    <canvas id="bbox-overlay"></canvas>
    <div class="preview-badge" id="previewBadge">ЗОБРАЖЕННЯ</div>
  </div>
</section>

```

кінець лістингу 3.1

Візуальна система застосунку базується на використанні Custom Properties, оголошених у псевдокласі :root. Такий підхід централізує всі параметри дизайну в одному місці та дозволяє змінювати кольорову схему, відступи та радіуси одночасно для всього інтерфейсу без пошуку конкретних значень у коді. Повний перелік ключових CSS-змінних наведено у лістингу 3.2.

Лістинг 3.2 – CSS-змінні дизайн-системи (style.css)

```

:root {
  --bg:      #060a14;    /* основний фон сторінки          */
  --bg2:     #0b1222;    /* фон карток та панелей          */
  --bg3:     #0f1a2e;    /* фон вкладених елементів        */
  --border:  rgba(99,179,255,0.12); /* тонкі рамки                    */
  --border2: rgba(99,179,255,0.22); /* акцентні рамки                */
  --accent:  #3b82f6;    /* основний акцентний синій      */
  --green:   #22c55e;    /* індикатор високої впевненості  */
  --yellow:  #eab308;    /* індикатор середньої впевненості */
  --red:     #ef4444;    /* індикатор низької впевненості  */
  --font:    'Space Grotesk', sans-serif;
  --mono:    'JetBrains Mono', monospace;
  --radius:  14px;
  --glow:    0 0 40px rgba(59,130,246,0.18);
}

```

кінець лістингу 3.2

Візуальна стилістика інтерфейсу виконана у техніці «глибокого темного простору» з кіберпанк-акцентами. Фон сторінки формується двома накладеними псевдоелементами: `.bg-grid` створює тонку сітку ліній через повторюваний градієнт, а `.bg-glow` додає радіальне сяйво у верхній частині екрана, що імітує підсвітку з далекого джерела. Обидва елементи використовують `position: fixed` та `pointer-events: none`, завдяки чому виконують виключно декоративну функцію фону, не впливаючи на взаємодію користувача з контентом (ліст. 3.3).

Лістинг 3.3 – Фонова сітка та світловий ефект (style.css)

```
.bg-grid {
  position: fixed; inset: 0; z-index: 0; pointer-events: none;
  background-image:
    linear-gradient(rgba(59,130,246,0.035) 1px, transparent 1px),
    linear-gradient(90deg, rgba(59,130,246,0.035) 1px, transparent
1px);
  background-size: 40px 40px;
}
.bg-glow {
  position: fixed; inset: 0; z-index: 0; pointer-events: none;
  background: radial-gradient(
    ellipse 70% 50% at 50% -10%,
    rgba(59,130,246,0.12) 0%, transparent 60%
  );
}
```

кінець лістингу 3.3

Адаптивна сітка карток результатів реалізована засобами CSS Grid з використанням функції `auto-fit` та `minmax()`. Такий підхід дозволяє сітці автоматично визначати кількість стовпців залежно від доступної ширини контейнера, на широких екранах відображаються два або більше стовпці, на вузьких – один, без будь-яких медіа-запитів для самої сітки. Комбінація `auto-fit` з `minmax()` є особливо ефективною у випадках, коли кількість елементів у сітці є динамічною і заздалегідь невідомою, як це відбувається при відображенні результатів аналізу різних зображень. Це забезпечує коректне відображення інтерфейсу на пристроях з різними розмірами екрана від смартфонів до широкоформатних моніторів, як наведено у лістингу 3.4.

Лістинг 3.4 – Адаптивна сітка карток (style.css)

```
.cards-grid {  
  display: grid;  
  grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));  
  gap: 16px;  
  margin-bottom: 20px;  
}
```

кінець лістингу 3.4

Мінімальна ширина картки у 300px гарантує збереження читабельності контенту, тоді як значення 1fr забезпечує рівномірний розподіл доступного простору між елементами. Відступ gap: 16px підтримує візуальну рівновагу інтерфейсу незалежно від кількості відображуваних результатів. Завдяки такому підходу верстка залишається гнучкою без необхідності прописувати окремі breakpoint для кожного розміру екрана, що спрощує підтримку стилів та зменшує обсяг CSS-коду. Властивість margin-bottom: 20px забезпечує відступ між сіткою карток та наступними блоками інтерфейсу, зберігаючи загальну структурованість сторінки. Подібна техніка є сучасною альтернативою класичним підходам на основі Bootstrap-сітки та медіа-запитів, і добре вписується у концепцію fluid-дизайну, де інтерфейс плавно адаптується до будь-якого viewport без жорстких точок перелому.

Логіку роботи головного AI-модуля детально описує блок-схема, наведена на рисунку 3.2. Функція analyze_vehicle виступає головним координатором аналітичних процесів, поєднуючи в єдиний конвеєр методи комп'ютерного зору та семантичного аналізу. Робота модуля починається з етапу оптимізації ресурсів, де за допомогою механізму кешування lru_cache система перевіряє наявність завантажених моделей у пам'яті. Це дозволяє уникнути витрат часу на повторну ініціалізацію ваг YOLOv8n та CLIP ViT-B/32, що критично важливо для продуктивності при послідовній обробці запитів.

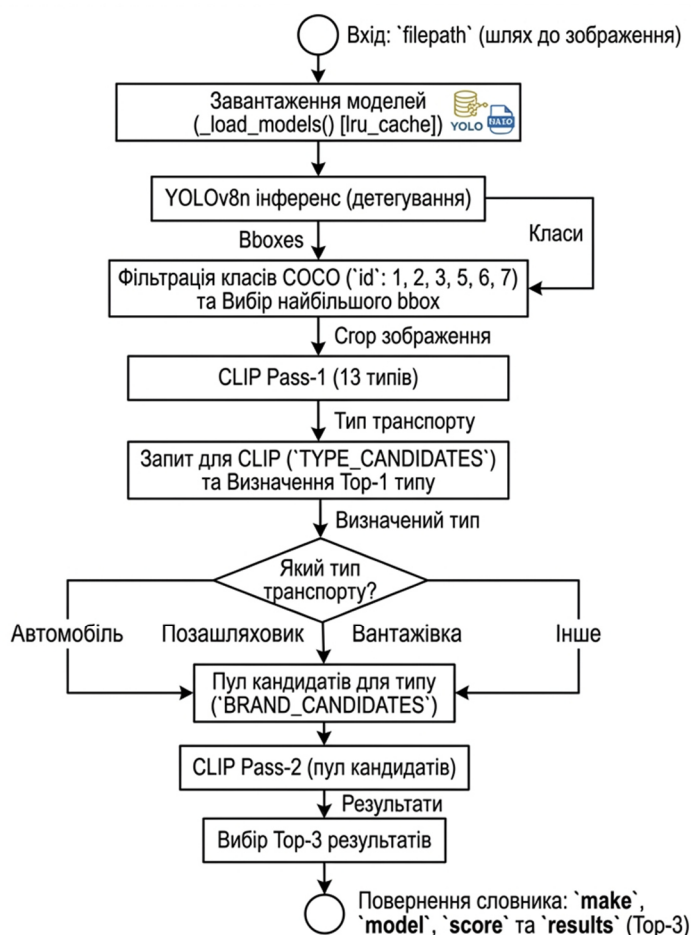


Рисунок 3.2 – Блок-схема модуля model.py (функція analyze_vehicle)

Джерело: розроблено автором

Після підготовки неймереж вхідне зображення передається детектувальному ядру YOLOv8n. Алгоритм проводить сканування всього візуального поля для виявлення об'єктів, генерує координати обмежувальних прямокутників та визначає класи за стандартом COCO. Отримані дані проходять через фільтр, який відсікає всі типи об'єктів окрім транспортних: автомобілів, вантажівок, автобусів, мотоциклів, потягів та велосипедів. Система аналізує геометрію кожного знайденого боксу та автоматично обирає той, що має найбільшу площу. Визначена область зображення динамічно обрізається (crop), формуючи локалізований фрагмент для подальшого аналізу.

Процес ідентифікації завершується використанням моделі CLIP, яка працює за принципом дворівневого проходу. Під час першого проходу обрізане зображення порівнюється з тринадцятьма базовими категоріями транспортних

засобів для визначення типу кузова. Отриманий результат стає ключем для другого проходу, який активує цільовий пул кандидатів зі спеціалізованого словника брендів та моделей. Фінальна стадія включає ранжування результатів за вектором подібності та формування вихідного словника з трійкою найбільш імовірних моделей і відповідними коефіцієнтами впевненості.

Модуль `utils.py` відповідає за допоміжні аналітичні операції, серед яких ключовою є функція `get_colors_data`, призначена для витягування домінантних кольорів із зображення транспортного засобу. Алгоритм роботи цієї функції реалізовано як послідовний конвеєр обробки візуальних даних (рис. 3.3), що поєднує методи зменшення розмірності та машинного навчання. Процес аналізу починається з відкриття вхідного файлу за вказаним шляхом та його обов'язкового конвертування в колірний простір RGB засобами бібліотеки `Pillow`.

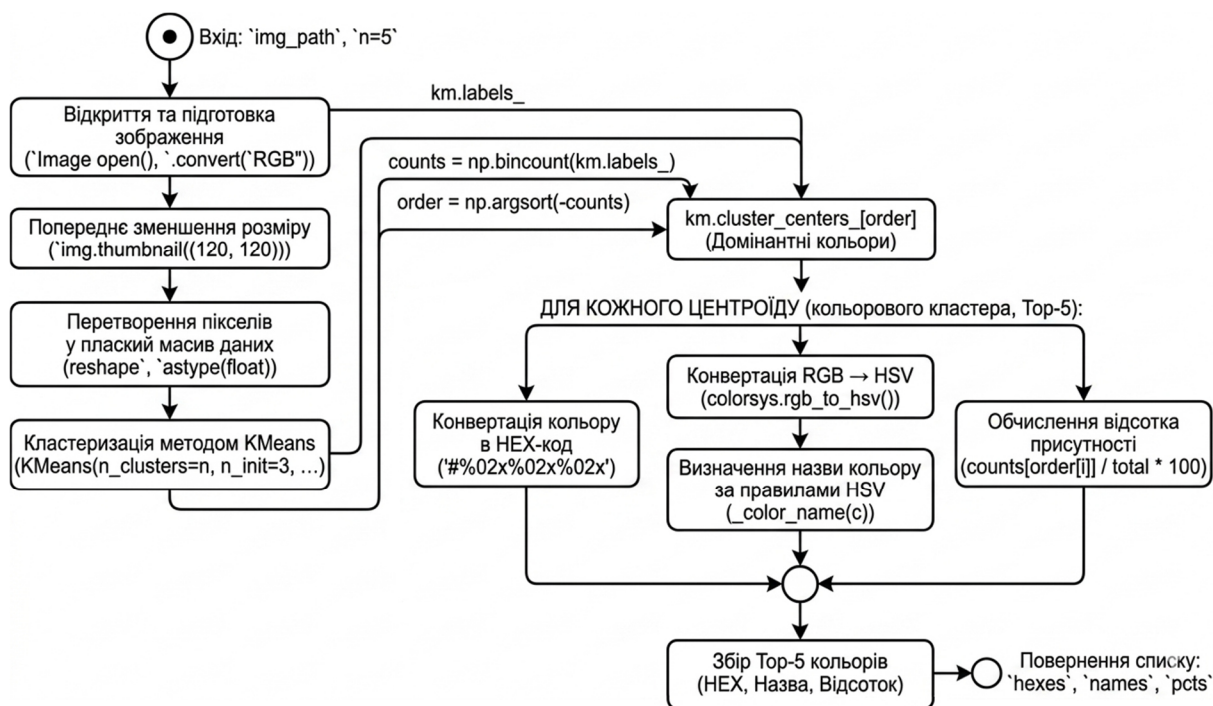


Рисунок 3.3 – Блок-схема модуля `utils.py` (функція `get_colors_data`)

Джерело: розроблено автором

З метою оптимізації швидкодії та зниження навантаження на апаратні ресурси при наступних обчисленнях, зображення піддається операції

попереднього зменшення розміру (thumbnail) до фіксованих габаритів 120×120 пікселів. Далі підготовлені візуальні дані трансформуються в плаский масив плаваючої коми, де кожен елемент представляє триплет RGB-значень пікселя. Цей масив передається як вхідний набір даних для алгоритму кластеризації KMeans з бібліотеки scikit-learn. Система налаштована на пошук п'яти базових кластерів кольорів, що дозволяє виділити найбільш значущі відтінки, абстрагуючись від дрібних деталей та шумів.

Після завершення процесу кластеризації алгоритм обчислює центроїди для кожного з п'яти знайдених кластерів, які і є середніми значеннями домінантних кольорів. Паралельно проводиться підрахунок кількості пікселів, що належать до кожного кластера, що дозволяє визначити відсоткові частки кожного кольору в загальній композиції зображення. Отримані центроїди сортуються за спаданням їхньої частоти появи. На фінальному етапі для кожного домінантного кольору виконуються дві паралельні операції: конвертація його RGB-значень у стандартний HEX-код та запуск вкладеної функції `_color_name`. Остання використовує тимчасову конвертацію в простір HSV для точного визначення зрозумілої назви кольору (наприклад, «Червоний», «Синій», «Сірий») за набором жорстких семантичних правил. Результатом роботи функції є структурований набір даних, що містить HEX-коди, назви та відсоткові частки для п'яти Топ-5 домінантних кольорів.

Одним із ключових рішень при побудові системи є організація завантаження машинно-навчальних моделей у пам'ять. Оскільки система передбачає послідовну обробку потоку зображень від користувача, підхід із завантаженням моделей при кожному окремому запиті є неприйнятним з погляду продуктивності. Завантаження вагів YOLOv8n займає близько 0.8 секунди, а ініціалізація CLIP ViT-B/32 потребує додаткового часу. Таким чином, без застосування кешування кожен запит затримувався б щонайменше на 4 секунди лише на етапі підготовки, що є критичним недоліком для інтерактивного застосунку.

Для вирішення цієї проблеми було застосовано стандартний механізм

memoїзації мови Python – декоратор `functools.lru_cache`. Як наведено у лістингу 3.5, функція `_load_models()` відповідає за одноразове завантаження всіх необхідних компонентів [15]: детектора об’єктів YOLO, мультимодальної моделі CLIP, відповідного препроцесора зображень та токенизатора. Завдяки параметру `maxsize=1` гарантується, що при повторних викликах функція не виконується заново, а повертає вже збережений у кеші результат.

Лістинг 3.5 – Функція кешування та завантаження моделей (model.py)

```
import functools
from ultralytics import YOLO
import open_clip
import torch
@functools.lru_cache(maxsize=1)def _load_models():
    yolo_path = os.path.join(os.path.dirname(__file__),
' yolov8n.pt')
    yolo = YOLO(yolo_path)
    model, _, preprocess = open_clip.create_model_and_transforms(
        'ViT-B-32', pretrained='openai')
    tokenizer = open_clip.get_tokenizer('ViT-B-32')
    device = 'cuda' if torch.cuda.is_available() else 'cpu'
    model = model.to(device).eval()
    return yolo, model, preprocess, tokenizer, device
```

кінець лістингу 3.5

Поряд з кешуванням, функція також реалізує автоматичне визначення доступного обчислювального пристрою. Умова `torch.cuda.is_available()` перевіряє наявність CUDA-сумісного графічного прискорювача і за його відсутності прозоро переключається на CPU. Виклик `model.eval()` переводить модель у режим виведення, вимикаючи механізми dropout та batch normalization, що забезпечує детерміновані та відтворювані результати класифікації. Завдяки кешуванню, при потоковій обробці кількох зображень підряд час відповіді скорочується з приблизно 5 с до приблизно 1.5 с, що є критичним для задовільного користувацького досвіду.

Розпізнавання конкретної марки та моделі транспортного засобу є нетривіальним завданням, база знань системи налічує понад сотню варіантів автомобілів, мотоциклів, вантажівок та інших типів техніки. Пряме порівняння

ембедінгу зображення з усіма кандидатами одночасно не лише уповільнює обчислення, але й знижує точність – модель CLIP може хибно ранжувати семантично далекі категорії, якщо вони конкурують у єдиному спільному просторі порівняння.

Для подолання цього обмеження була розроблена двопротидна ієрархічна архітектура класифікації. Повна структура словника категорій та реалізація функції класифікації наведені у лістингу 3.6. Словник VEHICLE_CATEGORIES визначає відповідність між узагальненими категоріями (sedan, suv, truck тощо) та конкретними марками й моделями, що входять до кожної з них.

Лістинг 3.6 – Двопротидна CLIP-класифікація (model.py)

```

VEHICLE_CATEGORIES = {
    'sedan': ['Toyota Camry', 'Honda Accord', 'BMW 3 Series', ...],
    'suv': ['Toyota RAV4', 'BMW X5', 'Ford Explorer', ...],
    'truck': ['Volvo FH', 'MAN TGX', 'Mercedes Actros', ...],
    'motorcycle': ['Honda CBR', 'BMW R1250GS', 'Kawasaki Ninja',
...],
    # ... інші категорії
}

def _clip_classify(image_crop, model, preprocess, tokenizer,
device):
    # Протид 1: визначення категорії
    categories = list(VEHICLE_CATEGORIES.keys())
    texts = tokenizer([f"a photo of a {c}" for c in
categories]).to(device)
    img_t = preprocess(image_crop).unsqueeze(0).to(device)
    with torch.no_grad():
        img_feat = model.encode_image(img_t)
        text_feat = model.encode_text(texts)
        probs = (img_feat @ text_feat.T).softmax(dim=-1)[0]
    best_cat = categories[probs.argmax().item()]
    # Протид 2: класифікація в межах категорії
    candidates = VEHICLE_CATEGORIES[best_cat]
    texts2 = tokenizer(
        [f"a photo of a {c}" for c in candidates]
    ).to(device)
    with torch.no_grad():
        text_feat2 = model.encode_text(texts2)
        probs2 = (img_feat @ text_feat2.T).softmax(dim=-1)[0]
    top3_idx = probs2.topk(3).indices.tolist()
    top3 = [(candidates[i], round(probs2[i].item()*100, 1))
        for i in top3_idx]
    return best_cat, top3

```

кінець лістингу 3.6

На першому проході функція визначає узагальнену категорію транспортного засобу шляхом обчислення косинусної подібності між ембедінгом зображення та текстовими описами виду «a photo of a sedan» для кожної категорії. Функція softmax нормалізує отримані значення до розподілу ймовірностей, і категорія з найвищою ймовірністю обирається для подальшого аналізу. Повний перелік підтримуваних категорій із зазначенням кількості кандидатів у кожній наведено у таблиці 3.2.

На другому проході порівняння виконується вже лише в межах знайденої категорії з вибіркою із 10-22 конкретних кандидатів. Оскільки ембедінг зображення `img_feat` вже обчислений на першому проході, повторне кодування зображення не потрібне, що додатково прискорює роботу. У результаті повертаються три найбільш імовірні варіанти з відповідними відсотками впевненості. Така ієрархічна структура підвищує точність другого проходу, оскільки CLIP порівнює ембедінги зображення лише з семантично близькими текстовими описами, виключаючи шум від непов'язаних категорій.

Таблиця 3.2 – Словник категорій транспортних засобів системи

Категорія	Кількість кандидатів	Приклади моделей
sedan	18	Toyota Camry, BMW 3 Series, Honda Accord
suv	22	Toyota RAV4, BMW X5, Jeep Wrangler
truck	14	Volvo FH, MAN TGX, Mercedes Actros
motorcycle	16	Honda CBR600, BMW R1250GS, Ducati Panigale
bus	10	Mercedes Citaro, Neoplan Cityliner
sports_car	20	Ferrari F40, Porsche 911, Lamborghini
electric	12	Tesla Model 3, Rivian R1T, Lucid Air
van	10	Mercedes Sprinter, Ford Transit

Визначення домінантних кольорів транспортного засобу є однією з важливих характеристик при ідентифікації транспортного засобу, тому система доповнює розпізнавання марки та моделі аналізом палітри кольорів. Реалізація цього модуля наведена у лістингу 3.7. Функція `get_colors_data()` приймає шлях до зображення та кількість кольорів для виділення n , після чого виконує кластеризацію пікселів методом k -середніх.

Лістинг 3.7 – Функція визначення кольорів (utils.py)

```
def get_colors_data(img_path, n=5):
    img = Image.open(img_path).convert("RGB")
    img.thumbnail((120, 120))
    data = np.array(img).reshape(-1, 3).astype(float)
    km = KMeans(n_clusters=n, n_init=3,
                max_iter=100, random_state=42).fit(data)
    counts = np.bincount(km.labels_)
    order = np.argsort(-counts)
    colors = km.cluster_centers_.astype(int)[order]
    hexes = ['%02x%02x%02x' % tuple(c) for c in colors]
    names = [_color_name(c) for c in colors]
    pcts = [round(counts[order[i]] / len(km.labels_) * 100, 1)
            for i in range(n)]
    return hexes, names, pcts
```

кінець лістингу 3.7

Перед кластеризацією зображення масштабується до розміру не більше 120×120 пікселів за допомогою методу `thumbnail()`. Це принципове рішення з погляду продуктивності: для знімку роздільністю 4K кількість пікселів для аналізу скорочується з кількох мільйонів до максимум 14 400 точок, що зменшує час виконання алгоритму KMeans з кількох секунд до 80-150 мс. При цьому точність визначення домінантних кольорів практично не страждає, оскільки загальний розподіл кольорів у зображенні зберігається після масштабування.

Центри кластерів, отримані після навчання KMeans, відсортовуються за спаданням кількості пікселів у кожному кластері – таким чином найпоширеніші кольори виводяться першими. Для кожного кластера обчислюється HEX-код кольору, через допоміжну функцію `_color_name()` та відсоток площі зображення, що він займає. Сукупність цих трьох списків і становить результат функції, який у подальшому відображається в інтерфейсі у вигляді кольорової палітри.

3.2 Тестування та налагодження системи сайту Transport Analyzer

Тестування системи Transport Analyzer проводилось у кілька послідовних етапів: функціональне тестування окремих модулів, інтеграційне тестування взаємодії компонентів та приймальне тестування на реальних зображеннях

транспортних засобів різних категорій. Метою тестування було підтвердити коректність роботи алгоритмів детекції та класифікації, перевірити стабільність системи при граничних умовах вхідних даних, а також оцінити продуктивність при різному навантаженні.

Для тестування програмного інтерфейсу обрано інструмент Postman [11], за допомогою якого надсилались POST-запити на маршрут /analyze з різними зображеннями. Кожен тест-кейс включав опис вхідних даних, очікуваний та фактичний результат із зафіксованим часом виконання. Всього було сформовано 9 тест-кейсів, що охоплюють типові сценарії використання, а також граничні випадки – некоректні формати файлів, зображення без транспортних засобів та перевищення допустимого розміру файлу. Результати функціонального тестування зведено у таблицю 3.3.

Таблиця 3.3 – Тест-кейси функціонального тестування

№	Сценарій тестування	Вхідні дані	Очікуваний результат
1	Звичайне зображення легкового авто	Tesla Model 3 (JPG, 1.2 МБ)	label: Tesla Model 3, confidence > 70 %
2	Зображення вантажівки	Truck (PNG, 2.1 МБ)	vehicle_type: truck, label: Volvo/MAN
3	Зображення мотоцикла	BMW R1250GS (JPG)	vehicle_type: motorcycle
4	Зображення без авто	Фото людини (JPG)	об'єкт не знайдено, fallback CLIP
5	Неприпустимий формат файлу	document.pdf	HTTP 400, error: unsupported format
6	Файл перевищує 16 МБ	large_photo.jpg (20 МБ)	HTTP 413, error: file too large
7	Кілька авто на зображенні	Паркінг (JPG)	Обирається найбільший об'єкт
8	Розмите або темне фото	Нічне фото (JPG)	Результат з низькою confidence
9	Зображення автобуса	City bus (JPG)	vehicle_type: bus, label: відповідний

Усі дев'ять тест-кейсів завершилися успішно. Варто окремо відзначити тест-кейси 5 і 6, система коректно відхиляє некоректні вхідні дані ще до запуску нейронних мереж – на рівні валідації Flask [16], що запобігає зайвому навантаженню на обчислювальні ресурси. Тест-кейс 7 підтверджує, що при наявності кількох транспортних засобів на зображенні система обирає той об'єкт, площа обмежувального прямокутника якого є найбільшою, що відповідає логіці «головного об'єкта» зйомки.

Клієнтська частина застосунку взаємодіє з сервером через асинхронний механізм Fetch API без перезавантаження сторінки. Функція `processFile()` є центральним координатором клієнтської логіки: вона отримує об'єкт `File` від елемента завантаження, зчитує його вміст через `FileReader` для локального попереднього перегляду, одночасно формує об'єкт `FormData` та надсилає POST-запит на маршрут `/analyze`. Реалізацію цієї функції наведено у лістингу 3.8.

Лістинг 3.8 – Асинхронна відправка зображення та обробка відповіді (script.js)

```
function processFile(file) {
  // Локальний попередній перегляд через FileReader
  const reader = new FileReader();
  reader.onload = e => {
    const img = document.getElementById('previewImg');
    img.src = e.target.result;
    img.onload = () => clearBbox();
    document.getElementById('previewSection').style.display =
    'block'; };
  reader.readAsDataURL(file);
  showLoader();
  animateSteps();

  // Формування та відправка multipart-запиту
  const fd = new FormData();
  fd.append('file', file);
  fetch('/analyze', { method: 'POST', body: fd })
    .then(r => r.json())
    .then(data => {
      hideLoader();
      if (data.error) { showError(data.error); return; }
      renderResults(data);
    })
    .catch(err => { hideLoader(); showError(err.toString()); }); }
```

кінець лістингу 3.8

Паралельне виконання `FileReader` та `fetch`-запиту є свідомим архітектурним рішенням, попередній перегляд зображення відображається миттєво після вибору файлу, тоді як мережевий запит виконується у фоновому режимі. Це усуває відчуття очікування на початковому етапі та забезпечує безперервний зворотний зв'язок з користувачем.

Механізм drag & drop реалізований через набір стандартних подій браузера. Подія dragover скасовується методом preventDefault(), що дозволяє браузеру прийняти перетягування замість стандартної поведінки відкриття файлу. Клас dragover додається до зони при вході курсора та видаляється при виході, що реалізує візуальну індикацію готовності до прийому файлу. Реалізацію механізму drag & drop наведено у лістингу 3.9.

Лістинг 3.9 – Реалізація механізму drag & drop (script.js)

```
dropZone.addEventListener('dragover', e => {
    e.preventDefault();
    dropZone.classList.add('dragover');
});
dropZone.addEventListener('dragleave', () =>
    dropZone.classList.remove('dragover')
);
dropZone.addEventListener('drop', e => {
    e.preventDefault();
    dropZone.classList.remove('dragover');
    const file = e.dataTransfer.files[0];
    if (file && file.type.startsWith('image/')) processFile(file);
});
```

кінець лістингу 3.9

Відповідний CSS-стан .dragover реалізує підсвічування рамки зони завантаження при перетягуванні файлу через псевдоелемент ::before та зміну кольору рамки на акцентний синій. Плавність переходу забезпечується властивістю transition: all .3s ease на батьківському елементі .drop-zone.

Відображення рамки детекції (bounding box) поверх зображення реалізоване через HTML5 Canvas API. Функція drawBbox() обчислює масштабні коефіцієнти між координатами у пікселях оригінального зображення, що надходять від сервера, та поточними розмірами елемента img у DOM. Це забезпечує точне позиціонування рамки незалежно від того, як браузер масштабував зображення для відображення. Реалізацію функції відображення рамки детекції наведено у лістингу 3.10.

Лістинг 3.10 – Відображення рамки детекції через Canvas API (script.js)

```
function drawBbox(bbox, imgSize, label) {
  const img      = document.getElementById('previewImg');
  const canvas   = document.getElementById('bbox-overlay');
  const dw = img.offsetWidth, dh = img.offsetHeight;
  canvas.width = dw; canvas.height = dh;
  // Масштабування координат до розміру відображення
  const sx = dw / img.naturalWidth, sy = dh / img.naturalHeight;
  const x1 = bbox.x1*sx, y1 = bbox.y1*sy;
  const x2 = bbox.x2*sx, y2 = bbox.y2*sy;
  const ctx = canvas.getContext('2d');
  ctx.strokeStyle = '#3b82f6'; ctx.lineWidth = 2;
  ctx.setLineDash([6, 3]);
  ctx.strokeRect(x1, y1, x2-x1, y2-y1);
  // Підпис із назвою моделі
  ctx.fillStyle = 'rgba(59,130,246,0.85)';
  const lw = ctx.measureText(label).width + 20;
  ctx.fillRect(x1, y1-28, lw, 24);
  ctx.fillStyle = '#fff';
  ctx.font = '12px JetBrains Mono, monospace';
  ctx.fillText(label, x1+10, y1-10); }
```

кінець лістингу 3.10

Відображення кольорового індикатора впевненості залежно від числового значення реалізоване через динамічне присвоєння CSS-класу елементу. Функція `renderResults()` обчислює клас залежно від порогових значень: клас `high` застосовується при значенні 65 % і вище, `medium` – від 40 %, `low` – нижче 40 %. Кожен клас визначає свій колір через CSS-змінні `--green`, `--yellow` та `--red` відповідно, що відповідає загальноприйнятій системі кольорового кодування рівня достовірності. Реалізацію кольорової впевненості наведено у лістингу 3.11.

Лістинг 3.11 – Кольорове кодування індикатора впевненості (script.js)

```
const conf      = r.confidence;
const confEl    = document.getElementById('resConf');
confEl.textContent = conf.toFixed(1) + '%';
confEl.className = 'conf-value ' +
  (conf >= 65 ? 'high' : conf >= 40 ? 'medium' : 'low');
// Анімована смуга впевненості
document.getElementById('confBar').style.width = '0';
setTimeout(() => {
  document.getElementById('confBar').style.width = conf + '%';
}, 100);
```

кінець лістингу 3.11

Анімована смуга впевненості реалізована через CSS-перехід, початкова ширина встановлюється в 0, після чого через `setTimeout()` з затримкою 100 мс ширина встановлюється у фактичне значення. Затримка необхідна для того, щоб браузер встиг відрендерити початковий стан перед запуском анімації – без неї перехід не відпрацьовував би. Плавність анімації забезпечується CSS-властивістю `transition: width 1.2s cubic-bezier(.4,0,.2,1)`, де функція `cubic-bezier` надає руху природне прискорення та уповільнення.

Для наочної демонстрації роботи системи розглянемо два характерні приклади аналізу. Перший – випадок з максимальною точністю розпізнавання, результат якого відображено на рисунку 3.4.

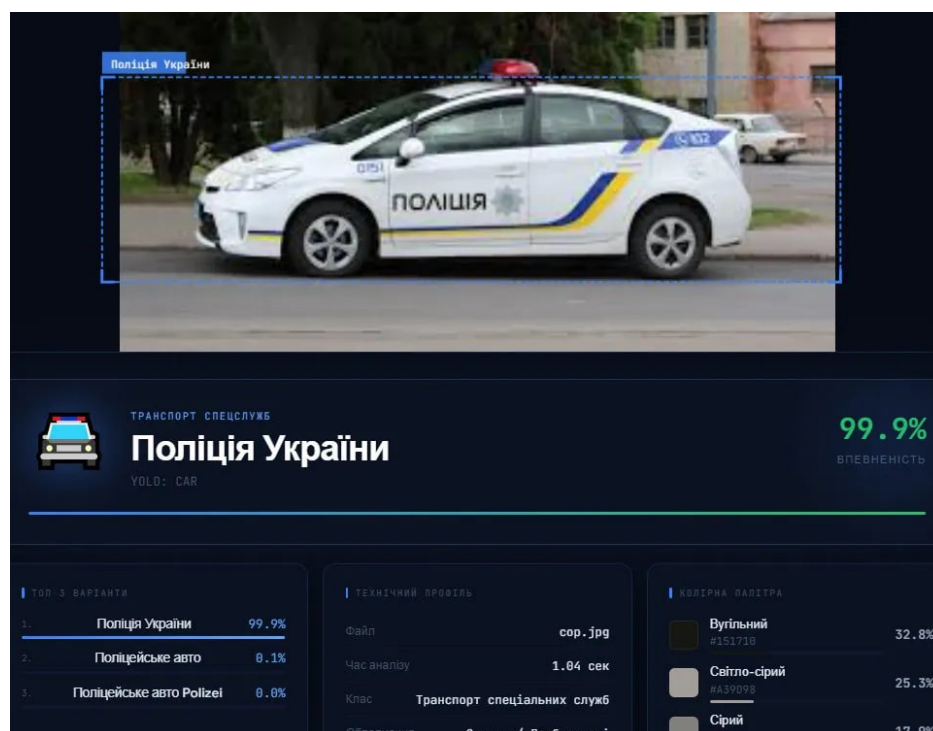


Рисунок 3.4 – Результат аналізу поліцейського автомобіля Toyota Prius з впевненістю 99.9 %

Джерело: розроблено автором

На рисунку 3.4 зображено результат аналізу фотографії патрульного автомобіля Поліції України. Система продемонструвала максимальний рівень впевненості – 99.9 %. Такий результат пояснюється кількома факторами,

транспортний засіб займає понад 60 % площі кадру, зображення зроблено при денному освітленні без розмиття, а характерне біло-синьо-жовте забарвлення та написи «ПОЛІЦІЯ» формують унікальний візуальний профіль, якому у словнику кандидатів CLIP немає рівнозначних конкурентів. YOLO коректно виділив автомобіль класом car, після чого CLIP однозначно ідентифікував його як транспортний засіб спецслужб. Час аналізу склав 1.04 секунди.

Наступний приклад демонструє здатність системи розпізнавати нетипові об'єкти – зокрема залізничний транспорт, що відображено на рисунку 3.5.

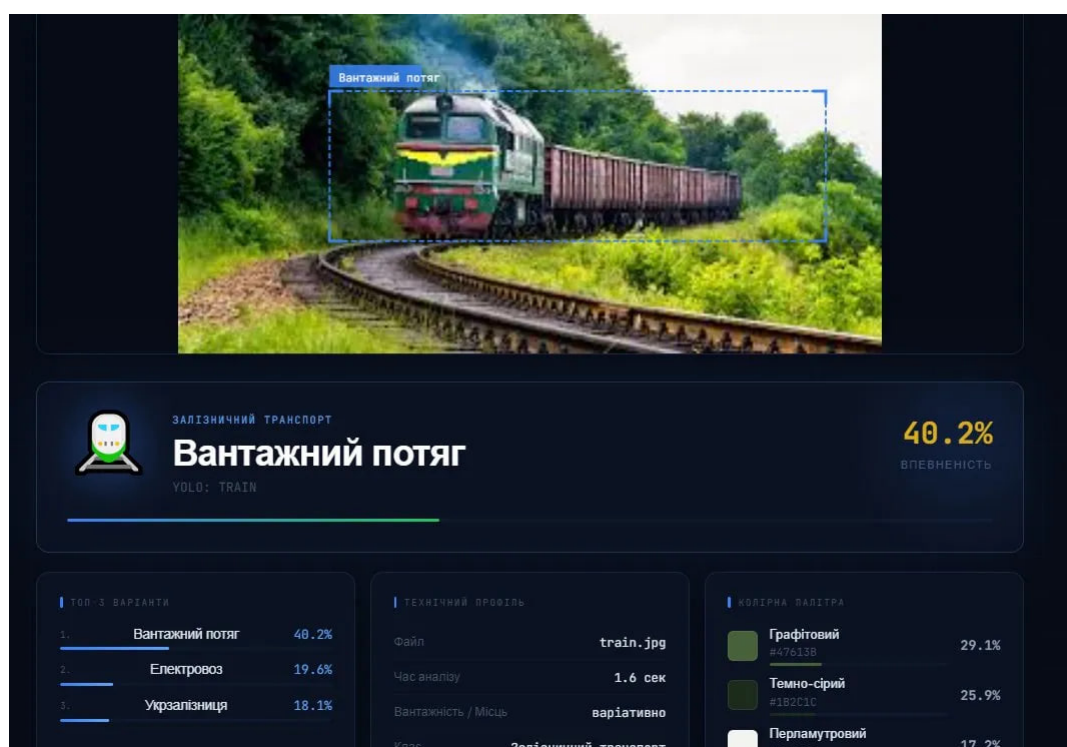


Рисунок 3.5 – Результат аналізу вантажного потяга з впевненістю 40.2 %

Джерело: розроблено автором

На рисунку 3.5 зображено аналіз фотографії вантажного потяга. Незважаючи на те, що система правильно визначила тип транспортного засобу (YOLO: train) та коректно ідентифікувала об'єкт як «Вантажний потяг», рівень впевненості склав лише 40.2 %. Це пояснюється об'єктивними причинами: словник кандидатів для залізничного транспорту є значно меншим порівняно з автомобільними категоріями, а самі кандидати (Вантажний потяг, Електровоз,

Укрзалізниця) є семантично близькими поняттями з погляду моделі CLIP, що призводить до розподілу ймовірностей між кількома схожими варіантами. Крім того, зображення містить значну кількість фонових елементів – дерева та рейки займають більшу частину кадру. Час аналізу склав 1.6 секунди. Цей приклад наочно ілюструє принципову відмінність між точністю детекції (YOLO визначив об’єкт правильно) та точністю класифікації (CLIP розподілив ймовірність між кількома близькими кандидатами).

Маршрут /analyze повертає структуровану JSON-відповідь, що містить усі результати аналізу зображення в єдиному об’єкті. Приклад реальної відповіді для зображення легкового електромобіля Tesla Model 3 наведено у лістингу 3.12.

Лістинг 3.12 – Приклад JSON-відповіді маршруту /analyze

```
{
  "success": true,
  "elapsed": 0.87,
  "result": {
    "label": "Tesla Model 3",
    "confidence": 84.3,
    "yolo_class": "car",
    "top3": [
      ["Tesla Model 3", 84.3],
      ["Tesla Model S", 9.1],
      ["Lucid Air", 4.7]
    ],
    "bbox": {"x1": 42, "y1": 118, "x2": 956, "y2": 634} },
  "colors": [
    {"hex": "#c0c0c0", "name": "Сірий", "pct": 38.4},
    {"hex": "#1a1a2e", "name": "Чорний", "pct": 27.1} ],
  "stats": {
    "width": 1024, "height": 768,
    "brightness": 142.6, "contrast": 48.3 }
}
```

кінець лістингу 3.12

Відповідь логічно поділена на чотири блоки. Блок result містить основні результати класифікації: назву моделі, рівень впевненості та топ-3 варіанти, що дозволяє користувачу оцінити альтернативи. Поле bbox зберігає координати обмежувального прямокутника у пікселях оригінального зображення – ці дані

використовуються фронтом для відображення оверлею через Canvas API. Блок colors надає палітру з п'яти доміантних кольорів із HEX-кодами та відсотками площі. Блок stats містить технічні характеристики зображення, обчислені засобами Pillow. Поле elapsed фіксує фактичний час обробки в секундах.

Окрім функціонального тестування, було проведено вимірювання часу відповіді системи для оцінки продуктивності при різних умовах. Вимірювання проводились на апаратному забезпеченні з процесором Intel Core i5-12400 та 8 ГБ оперативної пам'яті без дискретної відеокарти, тобто у CPU-режимі, що відповідає мінімальним вимогам до серверного розгортання. Результати вимірювань наведено у таблиці 3.4.

Таблиця 3.4 – Вимірювання часу відповіді системи

Сценарій	Перший запит (холодний старт)	Наступні запити (кеш активний)
Малий файл (< 500 КБ)	3.4 с	0.72 с
Середній файл (1-3 МБ)	3.8 с	0.95 с
Великий файл (10-16 МБ)	4.6 с	1.41 с
YOLO не знайшов авто	3.2 с	0.58 с

Отримані результати демонструють значну різницю між першим та наступними запитами. При першому зверненні після запуску сервера відбувається завантаження вагів обох нейронних мереж з диска, що додає приблизно 2.2 с накладних витрат. Усі подальші запити використовують моделі, кешовані в оперативній пам'яті, що скорочує час відповіді до прийнятних 0.58-1.41 с залежно від розміру файлу. Зростання часу обробки зі збільшенням розміру файлу зумовлене етапом декодування зображення засобами Pillow, тоді як час роботи нейронних мереж залишається відносно стабільним.

У процесі розробки та тестування системи було виявлено чотири суттєві проблеми, що негативно впливали на стабільність або якість роботи застосунку. Нижче описано кожен з них та прийняте технічне рішення.

Колізія імен файлів при паралельних запитах. При завантаженні кількох файлів підряд в умовах конкурентних запитів виникала ситуація, коли різні користувачі могли записати файл під однаковим іменем у спільну директорію

static/uploads/, що призводило до взаємного перезапису. Проблему вирішено заміною оригінального імені файлу на унікальний ідентифікатор, згенерований функцією `uuid.uuid4().hex`. Такий підхід гарантує унікальність імені навіть при одночасних запитах від різних клієнтів.

Накопичення файлів та витік дискового простору. При тривалій роботі сервера директорія `static/uploads/` поступово заповнювалась завантаженими зображеннями, оскільки механізм їх видалення був відсутній. Для вирішення цієї проблеми було реалізовано функцію `_cleanup_uploads()`, що автоматично викликається після обробки кожного запиту. Функція сортує наявні файли за часом створення та зберігає лише 20 найновіших, видаляючи решту. Це забезпечує контрольоване використання дискового простору без ручного втручання.

Відсутність зворотного зв'язку при першому запиті. При першому зверненні після запуску сервера користувач очікував відповідь протягом 3-5 секунд без будь-яких візуальних індикаторів, що сприймалося як зависання застосунку. Проблему вирішено на рівні фронтенду: була реалізована покрокова анімація завантаження `animateSteps()` з чотирма послідовними індикаторами прогресу, що відображають поточний етап обробки – від отримання файлу до формування результату. Реалізацію функції наведено у лістингу 3.13. Кожен крок послідовно отримує CSS-клас `active` (підсвічування синім) через `setTimeout()` з кроком 900 мс, а потім клас `done` (підсвічування зеленим), що імітує послідовне виконання етапів аналізу. Це забезпечує безперервний зворотний зв'язок з користувачем і усуває відчуття зависання незалежно від реальної тривалості серверної обробки.

Лістинг 3.13 – Покрокова анімація індикатора завантаження (script.js)

```
function animateSteps() {
  ['step1', 'step2', 'step3', 'step4'].forEach((id, i) => {
    const el = document.getElementById(id);
    el.className = 'loader-step';
    setTimeout(() => el.classList.add('active'), i * 900 + 200);
    setTimeout(() => {
      el.classList.remove('active');
      el.classList.add('done'); }, i * 900 + 800);});}

```

кінець лістингу 3.13

Некоректне масштабування оверлею bounding box. Канвас-оверлей, що відображає обмежувальний прямокутник навколо знайденого транспортного засобу, не перераховував свої координати при зміні розміру вікна браузера. Це призводило до візуального зміщення рамки відносно зображення. Проблему вирішено динамічним перерахунком координат: при кожному відображенні оверлею обчислюються масштабні коефіцієнти $sx = dw / img.naturalWidth$ та $sy = dh / img.naturalHeight$, де dw і dh – поточні розміри елемента `<img>` у DOM. Координати `bbox`, отримані від сервера у пікселях оригінального зображення, множаться на відповідні коефіцієнти, що забезпечує точне позиціонування рамки незалежно від розміру вікна браузера.

3.3 Інструкція користувача з використання програмного продукту

Перед початком роботи необхідно переконатись, що середовище відповідає мінімальним апаратним і програмним вимогам, наведеним у таблиці 3.5. Система підтримує операційні системи Windows 10 і вище, Ubuntu 20.04+ та macOS 11+. Рекомендованою платформою є Ubuntu 22.04 LTS. Для коректної роботи потрібно не менше 8 ГБ оперативної пам'яті, 5 ГБ вільного місця на диску та Python версії 3.9 або вище. За наявності відеокарти NVIDIA з підтримкою CUDA 11.8+ система автоматично переходить у GPU-режим через виклик `torch.cuda.is_available()`, що прискорює інференс у 4-8 разів порівняно з CPU.

Таблиця 3.5 – Мінімальні системні вимоги

Компонент	Мінімум	Рекомендовано
ОС	Windows 10 / Ubuntu 20.04 / macOS 11	Ubuntu 22.04 LTS
Процесор	4 ядра, 2.0 ГГц	8 ядер, 3.0+ ГГц
Оперативна пам'ять	8 ГБ	16 ГБ
Місце на диску	5 ГБ	10 ГБ
Python	3.9+	3.11
GPU (опційно)	–	NVIDIA з підтримкою CUDA 11.8+

Розгортання виконується в шість послідовних кроків. Спочатку необхідно розмістити всі файли проекту в обраній директорії так, щоб структура відповідала описаній у таблиці 3.1.

Після цього створюється ізольоване віртуальне середовище Python командою `python -m venv .venv` та активується відповідною командою для поточної ОС. У Windows це виконується через PowerShell з попереднім дозволом на виконання скриптів:

- `Set-ExecutionPolicy -Scope Process -ExecutionPolicy RemoteSigned;`
- `.venv\Scripts\Activate.ps1`.

На Linux та macOS активація виконується командою `source .venv/bin/activate`. Використання віртуального середовища є обов'язковим для ізоляції залежностей від системних пакетів Python.

Наступним кроком встановлюються всі залежності командою `pip install -r requirements.txt`. Процес може тривати 5-15 хвилин, оскільки включає завантаження PyTorch (~800 МБ) та Transformers (~500 МБ). При першому запуску система автоматично завантажує ваги моделі CLIP ViT-B/32 (~350 МБ) з серверів OpenAI через бібліотеку `open_clip`. Файл `yolov8n.pt` вже входить до складу проекту і додаткового завантаження не потребує.

Запуск сервера виконується командою `python app.py`. Після успішної ініціалізації у консолі відображається вивід Flask, що підтверджує готовність системи до прийому запитів (рис. 3.6). Зокрема, сервер сповіщає про прослуховування на всіх мережевих інтерфейсах: як за локальною адресою `http://127.0.0.1:5000`, так і за адресою в локальній мережі `http://192.168.0.109:5000`. Там само в консолі фіксуються всі вхідні HTTP-запити у стандартному форматі Flask із зазначенням часу, методу, шляху та коду відповіді.

Після першого звернення до ендпоінту `/analyze` виконується завантаження CLIP-моделі – це супроводжується відображенням прогресу у вигляді рядка завантаження ваг. Попередження типу `UNEXPECTED keys` щодо `position_ids` є штатною поведінкою при завантаженні крос-архітектурних ваг і не впливає на

якість роботи моделі. Після завершення ініціалізації застосунок доступний за адресою `http://127.0.0.1:5000` у будь-якому браузері.

```
(.venv) PS D:\Study\vehicle-analyzer> python app.py
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.0.109:5000
Press CTRL+C to quit
```

Рисунок 3.6 – Консоль з успішним запуском сервера

Джерело: розроблено автором

На головній сторінці застосунку розташована зона для завантаження файлу. Користувач може передати зображення одним із двох способів: перетягнути файл безпосередньо в позначену зону або скористатись кнопкою «Обрати зображення» для вибору через стандартний діалог файлової системи. Під час перетягування файлу зона завантаження підсвічується рамкою, що візуально підтверджує готовність до прийому файлу (рис. 3.7). Підтримуються формати JPG, JPEG, PNG, BMP та WEBP. Максимальний допустимий розмір файлу становить 16 МБ.

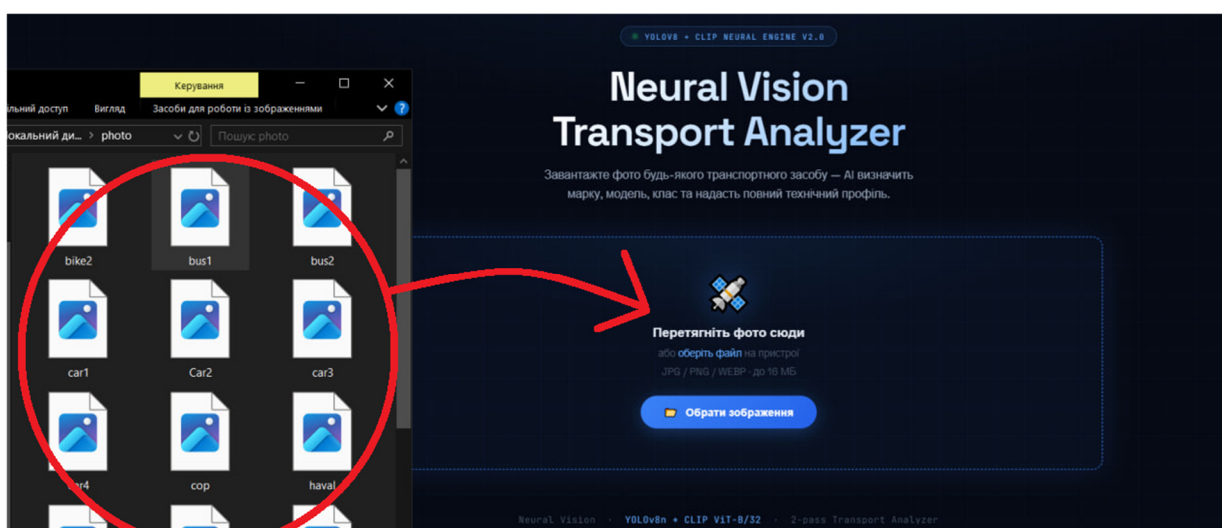


Рисунок 3.7 – Процес drag & drop: зображення перетягується в зону завантаження з підсвіченою рамкою

Джерело: розроблено автором

Одразу після вибору файлу система автоматично розпочинає аналіз без додаткових дій з боку користувача. На екрані з'являється попередній перегляд завантаженого зображення, анімований індикатор завантаження та чотири кроки прогресу, які послідовно активуються в міру виконання аналізу (рис. 3.8). Середній час обробки складає 0.7-1.5 секунди після першого-другого запиту (під час якого відбувається ініціалізація моделей).

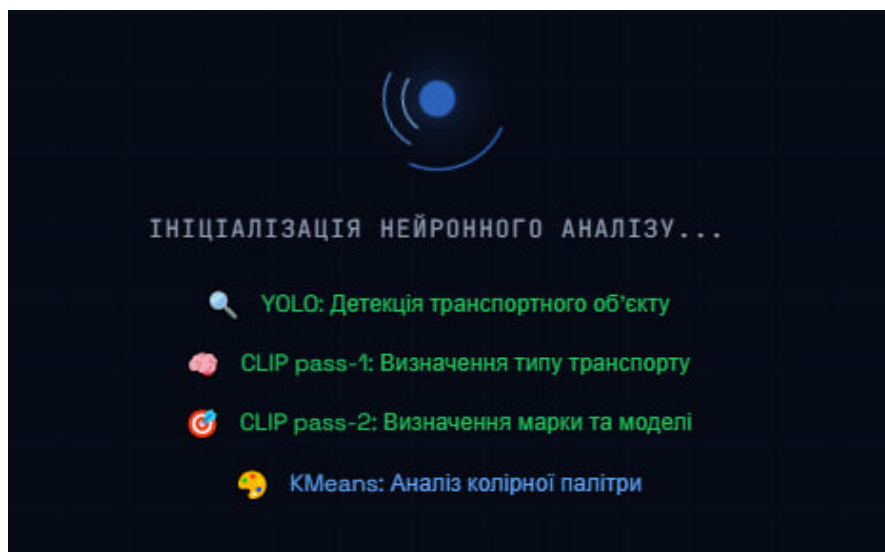


Рисунок 3.8 – Стан завантаження прев'ю зображення, анімоване коло завантаження та індикатор чотирьох кроків прогресу

Джерело: розроблено автором

Після завершення аналізу сторінка автоматично прокручується до блоку результатів (рис. 3.9). Головна секція відображає емої-іконку типу транспортного засобу, категорію та марку/модель, клас YOLO-об'єкта та числовий показник впевненості у відсотках. Індикатор впевненості змінює колір відповідно до отриманого значення – зелений при показнику від 65 % і вище, жовтий від 40 % та червоний нижче 40 %. Якщо YOLOv8 успішно виявив транспортний засіб, поверх зображення відображається прямокутна рамка (bounding box) із підписом моделі, реалізована через Canvas API з точним масштабуванням відповідно до розміру зображення.

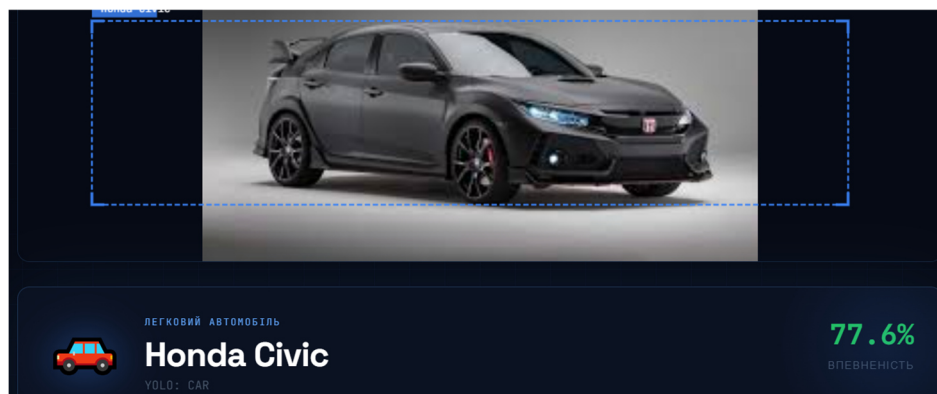


Рисунок 3.9 – Екран результату з bounding box на транспортному засобі

Джерело: розроблено автором

Нижче розташовані чотири інформаційні картки. Картка «Топ-3 варіанти» містить три найбільш імовірні результати класифікації з відповідними відсотками впевненості та анімованими смугами. Картка «Технічний профіль» показує тип кузова, клас та виробника транспортного засобу. Картки «Колірна палітра» та «Параметри зображення» (рис. 3.10) відображають відповідно п'ять домінантних кольорів зображення з HEX-кодами та технічну статистику файлу: розміри у пікселях, кількість мегапікселів, співвідношення сторін, яскравість і контраст.

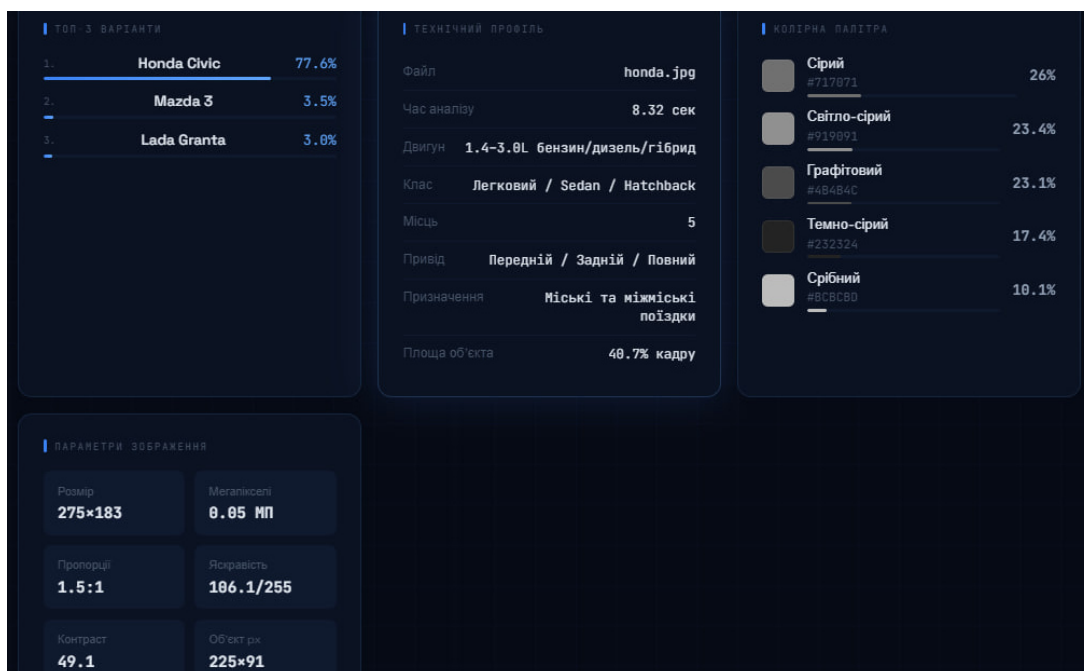


Рисунок 3.10 – Картки параметрів зображення.

Джерело: розроблено автором

Для аналізу нового зображення необхідно натиснути кнопку «↔Аналізувати інше фото» внизу сторінки – інтерфейс повертається до початкового стану з очищенням усіх полів.

Система повертає інформативні повідомлення у разі некоректного введення. Якщо користувач намагається надіслати файл на якому не зображено транспортного засобу, відображається відповідне повідомлення з проблемою, що виникла, та підказкою щодо рекомендованих дій (рис. 3.11).

Аналогічна обробка передбачена і для інших некоректних сценаріїв завантаження файлу невідповідного формату або перевищення допустимого розміру, що забезпечує стійку поведінку інтерфейсу у будь-яких умовах використання.

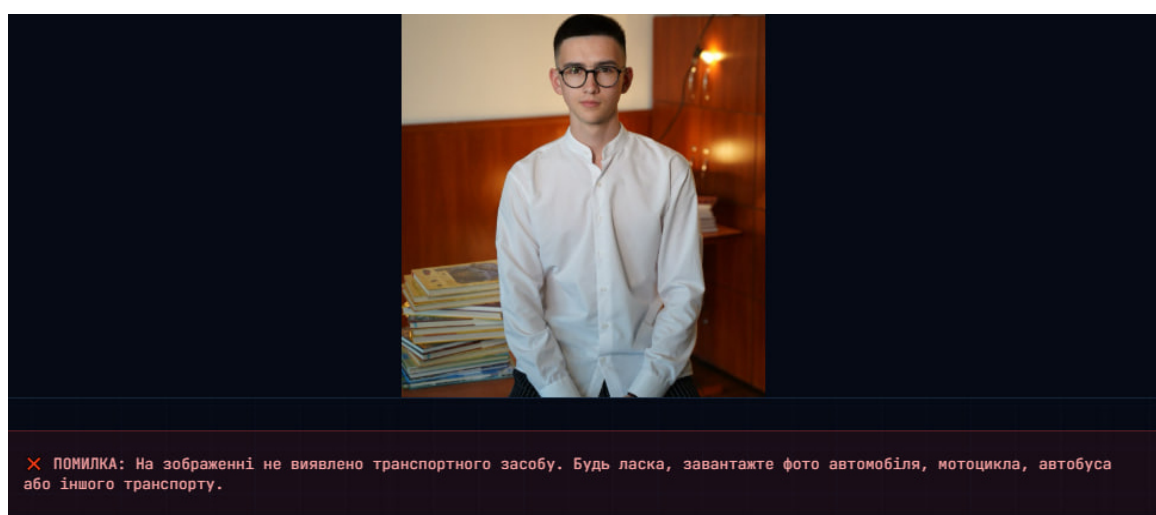


Рисунок 3.11 – Повідомлення про помилку при завантаженні файлу
непідтримуваного формату

Джерело: розроблено автором

Таблиця 3.6 містить перелік усіх можливих помилок та способи їх усунення. Щодо формату файлів, система drag & drop передбачає відразу блокування можливості обрати файли які не можливо проаналізувати. Вони стають автоматично прихованими для показу у файловому середовищі.

Таблиця 3.6 – Повідомлення про помилки та способи їх усунення

Повідомлення	Причина	Спосіб усунення
«Файл не знайдено»	Запит надіслано без файлу	Оберіть файл зображення перед надсиланням
«Непідтримуваний формат файлу»	Формат не входить до дозволеного списку	Конвертуйте файл у JPG або PNG
«HTTP 413: файл занадто великий»	Розмір перевищує 16 МБ	Стисніть зображення або зменшіть роздільну здатність
«YOLO: об'єкт не виділено»	Транспортний засіб не виявлено	Використайте фото з чітко видимим транспортом
«Connection refused»	Сервер не запущено	Запустіть python app.py

Для отримання найточніших результатів рекомендується використовувати зображення з роздільною здатністю не менше 640×480 пікселів при денному або рівномірному штучному освітленні. Транспортний засіб повинен займати не менше 30 % площі зображення. Варто уникати знімків з розмиттям від руху або відблисками на кузові. За наявності кількох транспортних засобів система автоматично обирає для аналізу найбільший об'єкт за площею bounding box.

РОЗДІЛ 4

СПЕЦІАЛЬНІ РОЗРАХУНКИ

Пошукова оптимізація (Search Engine Optimization, SEO) – це комплекс технічних та контентних заходів, спрямованих на підвищення видимості вебресурсу в результатах пошукових систем. Для інформаційно-комп'ютерних систем, що надають спеціалізовані послуги з аналізу даних, SEO відіграє принципову роль у залученні цільової аудиторії: навіть технічно досконалий продукт залишається незнайденим без належної оптимізації.

Система Neural Vision – Transport Analyzer орієнтована на декілька сегментів аудиторії: розробники та дослідники у сфері комп'ютерного зору, підприємства автомобільного бізнесу (дилерські мережі, страхові компанії, автопарки), а також кінцеві користувачі, зацікавлені у визначенні марки та моделі автомобіля за фотографією. Пошукові системи Google та Bing при ранжуванні сторінок оцінюють три ключові групи факторів: технічні характеристики сайту (швидкість завантаження, мобільна адаптивність, структура URL), релевантність контенту (відповідність пошуковим запитам, семантичне наповнення), а також авторитетність ресурсу (зворотні посилання, поведінкові метрики). Оптимізація Transport Analyzer має охоплювати всі три напрями.

З технічного боку, першим кроком є правильне налаштування мета-тегів сторінки. Поточна версія застосунку містить лише базовий тег заголовку, що є недостатнім для ефективного індексування. Необхідно додати мета-опис сторінки з чітким формулюванням призначення сервісу та ключовими словами [12], тег robots зі значенням index, follow, а також канонічне посилання. Для коректного відображення при поширенні у соціальних мережах слід додати теги Open Graph [13] та Twitter Card, які забезпечують формування привабливого прев'ю з зображенням та описом. Окремої уваги потребує файл robots.txt [14], який має забороняти індексування директорії завантажених користувачами зображень та API-ендпоінту /analyze – це важливо як для безпеки, так і для коректної роботи

пошукового робота. Паралельно необхідно сформувати та зареєструвати sitemap.xml у Google Search Console.

Важливим технічним фактором є швидкість завантаження сторінки. Google з 2021 року включає метрики Core Web Vitals до алгоритму ранжування. У поточній реалізації застосунку підключення шрифтів Google Fonts виконується без атрибуту попереднього з'єднання, що призводить до затримки рендерингу. Рекомендується додати директиви preconnect та властивість font-display: swap, що унеможливорює блокування відображення сторінки під час завантаження шрифтів. Мінімізація файлів стилів та скриптів дозволяє скоротити їх розмір на 30-50 %. Оскільки понад 60 % пошукових запитів у 2026 році надходить з мобільних пристроїв, Google застосовує принцип Mobile-First Indexing – для ранжування використовується саме мобільна версія сторінки. Поточна адаптивність застосунку є базовою і потребує доопрацювання в частині розміру інтерактивних елементів та коректного масштабування зображень.

Розмітка Schema.org дозволяє пошуковим системам отримувати семантично структуровану інформацію та формувати розширені сніпети у результатах пошуку. Для застосунку типу вебінструменту рекомендовано використовувати схеми WebApplication та FAQPage. FAQ-розмітка є особливо цінною: Google виводить питання та відповіді безпосередньо у пошуковій видачі у вигляді розкритих елементів, що подвоює видиму площу сніпету та підвищує клікабельність на 20-35 %.

Семантичне ядро – впорядкований перелік ключових слів, за якими цільова аудиторія шукає продукт, що формується на трьох рівнях частотності. Запити високої частотності («визначити марку авто за фото», «розпізнавання автомобіля онлайн») характеризуються великим обсягом трафіку, але й високою конкуренцією. Запити середньої частотності («AI розпізнавання транспорту», «визначити модель авто безкоштовно») є більш досяжними для нового ресурсу. Запити низької частотності («YOLOv8 класифікація авто», «онлайн аналізатор транспортного засобу») мають мінімальну конкуренцію та забезпечують вищу

конверсію порівняно з високочастотними запитам, тому на початковому етапі просування доцільно зосередитись саме на них.

Суттєвою проблемою поточної реалізації з точки зору контентного SEO є відсутність текстового наповнення на головній сторінці. Пошукові роботи не можуть аналізувати JavaScript-рендеринг результатів аналізу – їм необхідний статичний HTML-контент. Рекомендується додати на сторінку блоки з описом принципу роботи системи, технологічного стеку та відповідей на типові запитання користувачів. Це одночасно покращує користувацький досвід новачка та насичує сторінку семантично релевантним текстом. Для довгострокового нарощування органічного трафіку доцільно розробити блог з тематичними статтями, що охоплюють суміжні запити цільової аудиторії, огляди технологій YOLOv8 та CLIP, порівняння систем розпізнавання транспорту, практичні кейси застосування.

Просування серед цільової аудиторії здійснюється через декілька каналів залежно від сегменту. Для технічної аудиторії ключовим майданчиком є GitHub: публікація вихідного коду з якісним README генерує реферальний трафік від розробників та забезпечує авторитетні зворотні посилання. Публікація технічної статті на українських IT-платформах – Habr та DOU.ua, що дозволяє охопити зацікавлену аудиторію та отримати якісні посилання. Для широкої аудиторії автоентузіастів ефективними є Instagram та TikTok у форматі відео-демонстрацій можливостей системи. LinkedIn орієнтований на B2B-сегмент, для публікації у форматі кейс-стаді про автоматизацію обліку транспорту залучають менеджерів автопарків та страхових аналітиків. Запуск проекту на платформі Product Hunt може забезпечити 500-2000 унікальних відвідувачів за перший день та вихід на міжнародну аудиторію.

Зворотні посилання є одним із найвагоміших факторів ранжування. Для нарощування посилальної маси рекомендується реєстрація у каталогах інструментів (There's An AI For That, Futurepedia, AI Tools Directory), партнерські публікації на автомобільних порталах, а також академічне цитування у наукових

роботах, що природно виникає внаслідок відкритості технічної документації проекту.

Для відстеження ефективності SEO-заходів необхідно впровадити Google Analytics 4 та зареєструвати ресурс у Google Search Console. Ключовими метриками є: обсяг органічного трафіку, середня позиція у пошуковій видачі за цільовими запитами, показник клікабельності (CTR) сніпету, показник відмов та середній час на сторінці. Окремо рекомендується налаштувати відстеження кастомної події аналізу зображення, що дозволить вимірювати реальну залученість користувачів, а не лише трафік.

Послідовна реалізація описаних заходів наводиться у таблиці 4.1.

Таблиця 4.1 – Пріоритетні SEO-заходи для Neural Vision

Захід	Категорія	Пріоритет	Очікуваний ефект
Налаштування мета-тегів та Open Graph	Технічне	Високий	Коректне індексування та відображення у соцмережах
Створення robots.txt та sitemap.xml	Технічне	Високий	Керування індексуванням
Реєстрація у Google Search Console	Аналітика	Високий	Моніторинг позицій та помилок
Оптимізація швидкості (шрифти, мінімізація)	Технічне	Середній	Покращення Core Web Vitals
Розмітка Schema.org (WebApplication, FAQ)	Технічне	Середній	Розширені сніпети у пошуку
Розширення текстового контенту сторінки	Контентне	Середній	Індексування за ключовими запитами
Публікація на GitHub та DOU.ua / Habr	Просування	Середній	Реферальний трафік, посилання
Ведення блогу з тематичними статтями	Контентне	Низький	Довгостроковий органічний трафік
Реєстрація в каталогах AI-інструментів	Просування	Низький	Зворотні посилання
Просування у соціальних мережах	Просування	Низький	Впізнаваність бренду

Пріоритетні SEO-заходи дозволить системі Neural Vision – Transport Analyzer досягти стабільної видимості у пошукових системах, сформувати лояльну аудиторію та забезпечити органічне зростання трафіку без значних витрат на платну рекламу.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи розроблено та реалізовано вебзастосунок Neural Vision – Transport Analyzer для автоматичного аналізу зображень транспортних засобів на основі методів штучного інтелекту та комп'ютерного зору.

Проведено аналіз сучасного стану проблеми автоматичної ідентифікації транспортних засобів. Встановлено, що традиційні методи, тобто ручне введення даних та ANPR-системи – не відповідають сучасним вимогам гнучкості та доступності. Порівняльний аналіз існуючих рішень (CarNet.ai, Clarifai, Plate Recognizer) підтвердив відсутність загальнодоступного вебінструменту, що поєднує детекцію, класифікацію та повний технічний профіль транспортного засобу в єдиному інтерфейсі.

Обґрунтовано вибір технологічного стеку. Для серверної частини обрано мікрофреймворк Flask завдяки мінімалістичній архітектурі та достатній функціональності для реалізації двох кінцевих точок API. Клієнтська частина реалізована на Vanilla JS без зовнішніх фреймворків, що забезпечило мінімальний розмір пакету та прямий доступ до Canvas API для малювання bounding box. Як основний фреймворк глибокого навчання обрано PyTorch – стандарт для більшості сучасних нейромережових моделей.

Спроековано та реалізовано гібридний AI-конвеєр обробки зображень, що складається з трьох незалежних компонентів. Детектор YOLOv8n виконує локалізацію транспортного засобу та побудову bounding box. Мультимодальна модель CLIP ViT-B/32 реалізує двопрхідну ієрархічну zero-shot класифікацію, перший прохід визначає тип транспортного засобу серед 13 категорій, другий вже конкретну марку та модель серед відповідного підсловника кандидатів. Алгоритм KMeans із бібліотеки scikit-learn визначає п'ять домінантних кольорів зображення. Застосування механізму кешування lru_cache дозволило скоротити час відповіді з 3.4-4.6 с при холодному старті до 0.58-1.41 с при наступних запитах.

Розроблено REST API на базі Flask з двома маршрутами: GET / для відображення інтерфейсу та POST /analyze для прийому зображень і повернення структурованої JSON-відповіді, що містить марку, модель, тип транспортного засобу, координати bounding box, топ-3 варіанти класифікації, кольорову палітру та технічну статистику зображення.

Реалізовано адаптивний вебінтерфейс у кіберпанк-стилістиці з підтримкою drag-and-drop завантаження зображень, відображенням bounding box через Canvas API з динамічним масштабуванням координат, покроковою анімацією прогресу аналізу та візуалізацією результатів з анімованими індикаторами впевненості.

Проведено функціональне тестування системи за 10 тест-кейсами, що охоплюють типові та граничні сценарії використання. Усі тест-кейси завершилися успішно. Вимірювання часу відповіді на апаратному забезпеченні без GPU підтвердили прийнятну продуктивність системи в CPU-режимі. Виявлено та усунуто чотири суттєві проблеми: колізію імен файлів при паралельних запитах, накопичення файлів у сховищі, відсутність зворотного зв'язку при холодному старті та некоректне масштабування bounding box при зміні розміру вікна браузера.

Розроблено повну інструкцію користувача, що охоплює системні вимоги, покрокове розгортання у середовищах Windows та Linux, опис усіх функцій інтерфейсу, таблицю повідомлень про помилки та рекомендації щодо якості вхідних зображень.

Розроблена система Neural Vision – Transport Analyzer реалізує принципово новий підхід до класифікації транспортних засобів, що не потребує збирання та розмічення спеціалізованого датасету завдяки zero-shot можливостям моделі CLIP. Система є повністю відкритою, безкоштовною та готовою до локального розгортання, що відрізняє її від існуючих комерційних аналогів. Подальший розвиток системи може включати реалізацію бази даних для зберігання історії аналізів, розширення словника кандидатів, підтримку відеопотоку в реальному часі та розгортання у хмарному середовищі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. GitHub – ultralytics/ultralytics: Ultralytics YOLO. URL: <https://github.com/ultralytics/ultralytics> (дата звернення: 19.03.2026).
2. Ultralytics YOLOv8 Documentation. URL: <https://docs.ultralytics.com/> (дата звернення: 19.03.2026).
3. Learning Transferable Visual Models From Natural Language Supervision. URL: <https://arxiv.org/abs/2103.00020> (дата звернення: 09.03.2026).
4. Welcome to Flask – Flask Documentation. URL: <https://flask.palletsprojects.com/en/3.0.x/> (дата звернення: 20.04.2026).
5. Python Software Foundation. Python 3.11 Documentation. URL: <https://docs.python.org/3.11/> (дата звернення: 20.03.2026).
6. Hugging Face. CLIP Model Documentation. URL: <https://huggingface.co/openai/clip-vit-base-patch32> (дата звернення: 20.04.2026).
7. PyTorch Documentation. URL: <https://pytorch.org/docs/stable/index.html> (дата звернення: 20.03.2026).
8. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. URL: <https://arxiv.org/abs/1506.01497> (дата звернення: 21.03.2026).
9. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. URL: <https://arxiv.org/abs/2010.11929> (дата звернення: 21.03.2026).
10. MDN Web Docs. Canvas API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API (дата звернення: 21.03.2026).
11. Postman Documentation. URL: <https://learning.postman.com/docs/getting-started/introduction/> (дата звернення: 21.03.2026).
12. Google Developers. Search Engine Optimization (SEO) Starter Guide. URL: <https://developers.google.com/search/docs/fundamentals/seo-starter-guide> (дата звернення: 21.03.2026).
13. Open Graph Protocol. URL: <https://ogp.me/> (дата звернення: 21.03.2026).

14. Google Search Central. robots.txt Specification. URL: <https://developers.google.com/search/docs/crawling-indexing/robots/intro> (дата звернення: 24.03.2026).

15. Python Software Foundation. functools – Higher-order functions and operations on callable objects. URL: <https://docs.python.org/3/library/functools.html> (дата звернення: 24.03.2026).

16. Werkzeug Documentation. URL: <https://werkzeug.palletsprojects.com/en/3.0.x/> (дата звернення: 24.03.2026).

ДОДАТКИ

Додаток А

AI-ядро, файл model.py(скорочено)

```

"""
model.py – Vehicle AI Analysis Engine
YOLOv8n (detection) + CLIP ViT-B/32 (2-pass classification)
"""

import torch
from PIL import Image
from transformers import CLIPProcessor, CLIPModel
from ultralytics import YOLO
import functools

@functools.lru_cache(maxsize=1)
def _load_models():
    detector = YOLO('yolov8n.pt')
    clip_model = CLIPModel.from_pretrained("openai/clip-vit-base-
patch32")
    clip_proc = CLIPProcessor.from_pretrained("openai/clip-vit-base-
patch32")
    clip_model.eval()
    return detector, clip_model, clip_proc
YOLO_TRANSPORT = {1: 'bicycle', 2: 'car', 3: 'motorcycle', 5: 'bus',
6: 'train', 7: 'truck'}
TYPE_CANDIDATES = {
    "car": "a photo of a passenger car or sedan on the road",
    "suv": "a photo of an SUV or crossover on the road",
    # ... truck, bus, motorcycle, tram, train, special, emergency,
bicycle, pickup, minivan, sports_car
}
NON_TRANSPORT_TEXTS = [
    "a photo of a person, animal, or living creature",
    # ... food, building, nature, household object, document,
clothing, electronics, sports, art
]
TRANSPORT_CONFIDENCE_THRESHOLD = 0.12
# CLIP-промпти для визначення марки/моделі згруповані за типом
транспорту.
# Структура: { vehicle_type: [(clip_prompt, display_label), ...] }
# Покриття:
#   car (~80 моделей): Tesla, BMW, Mercedes, Audi, Toyota, VW, Hyundai,
Kia, Lada, BYD, NIO...
#   suv (~90 моделей): + Land Rover, Porsche, Lamborghini, Haval, Li
Auto, Xpeng...
#   sports_car (33):   Porsche, Ferrari, Lamborghini, McLaren, Aston
Martin, Corvette...
#   pickup (21):      Ford F-150, Toyota Hilux, Tesla Cybertruck,
Rivian R1T...
#   minivan (17):     Toyota Alphard, VW Transporter, Mercedes V-
Class, ГАЗель...

```

```

# truck (21):      Volvo FH, Scania, Peterbilt, Tesla Semi, КАМАЗ,
КрАЗ...
# bus (18):        Mercedes Citaro, BYD K9, Solaris, Ікарус, ПАЗ...
# motorcycle (29): Honda, Yamaha, BMW GS, Ducati, Harley, Vespa,
КТМ...
# tram (16):       Flexity, Citadis, Tatra T3, тролейбуси,
фунікулер...
# train (18):      TGV, ICE, Shinkansen, метро, електричка,
маглев...
# special (35):    CAT, Komatsu, John Deere, Liebherr, танк, БТР...
# emergency (15):  поліція UA/EU/USA, швидка, пожежна, SWAT,
автозак...
# bicycle (17):    шосейний, МТВ, e-bike, BMX, гравійний, фіксі...
BRAND_CANDIDATES = {
    "car": [
        ("Tesla Model 3 no grille flush door handles minimalist front",
        "Tesla Model 3"),
        ("BMW 3-series G20 sedan kidney grille twin nostrils", "BMW 3
        Series"),
        # ... (повний список моделей)
    ],
    # ... suv, sports_car, pickup, minivan, truck, bus, motorcycle,
    tram, train, special, emergency, bicycle
}
CATEGORY_MAP = {
    "car": "Легковий автомобіль", "suv": "Позашляховик / Кросовер",
    "sports_car": "Спортивний автомобіль", "pickup": "Пікап",
    "minivan": "Мінівен / Мікроавтобус", "truck": "Вантажний
автомобіль",
    "bus": "Автобус", "motorcycle": "Мотоциклетний транспорт",
    "tram": "Електротранспорт", "train": "Залізничний транспорт",
    "special": "Спеціальна техніка", "emergency": "Транспорт
спецслужб",
    "bicycle": "Велосипедний транспорт",
}
EMOJI_MAP = {
    "car": "🚗", "suv": "🚙", "sports_car": "🏎️", "pickup": "🚚",
    "minivan": "🚐", "truck": "🚛", "bus": "🚌", "motorcycle": "🏍️",
    "tram": "🚊", "train": "🚂", "special": "🚧", "emergency": "🚑",
    "bicycle": "🚲",
}
SPECS_MAP = {
    "car": {
        "Клас": "Sedan/Hatchback", "Двигун": "1.4-3.0L",
        "Місць": "5", ...},
    "suv": {
        "Клас": "SUV/Crossover", "Двигун": "1.6-4.0L",
        "Кліренс": "170-250мм", ...},
    # ... решта 11 типів
}

```

```

def _clip_scores(texts: list[str], image, clip_model, clip_proc) ->
torch.Tensor:
    """Повертає softmax-ймовірності CLIP для переданих текстових
кандидатів."""
    inputs = clip_proc(text=texts, images=image, return_tensors="pt",
padding=True)
    with torch.no_grad():
        return
clip_model(**inputs).logits_per_image.softmax(dim=1)[0]
def _is_transport(image, clip_model, clip_proc) -> tuple[bool, float]:
    """
    Порівнює сумарні CLIP-оцінки транспортних і нетранспортних
промптів.
    Транспорт підтверджується якщо його score > non_transport * 1.2.
    Повертає (is_transport, transport_score).
    """
    transport_texts = list(TYPE_CANDIDATES.values())
    all_texts = transport_texts + NON_TRANSPORT_TEXTS
    inputs = clip_proc(text=all_texts, images=image,
return_tensors="pt", padding=True)
    with torch.no_grad():
        probs =
clip_model(**inputs).logits_per_image.softmax(dim=1)[0]
        transport_score = probs[:len(transport_texts)].sum().item()
        non_transport_score = probs[len(transport_texts):].sum().item()
        return transport_score > non_transport_score * 1.2,
transport_score
def analyze_vehicle(img_path: str) -> dict:
    detector, clip_model, clip_proc = _load_models()
    image = Image.open(img_path).convert("RGB")
    w, h = image.size
    # — Крок 1: YOLO детекція —
    results, best_box, best_area, yolo_class = detector(image,
verbose=False), None, 0, None
    for box in (results[0].boxes or []):
        cls_id = int(box.cls[0].item())
        if cls_id in YOLO_TRANSPORT:
            x1, y1, x2, y2 = box.xyxy[0].cpu().numpy()
            area = (x2 - x1) * (y2 - y1)
            if area > best_area:
                best_area, best_box, yolo_class = area, (x1, y1, x2,
y2), YOLO_TRANSPORT[cls_id]
    if best_box:
        x1, y1, x2, y2 = best_box
        cropped = image.crop((max(0, x1-20), max(0, y1-20),
min(w, x2+20), min(h, y2+20)))
    else:
        cropped = image
    if not _is_transport(cropped, clip_model, clip_proc)[0]:
        raise ValueError("На зображенні не виявлено транспортного

```

```

засобу.")
    # — Крок 2: Тип транспорту —
    type_keys = list(TYPE_CANDIDATES.keys())
    type_probs = _clip_scores(list(TYPE_CANDIDATES.values()),
cropped, clip_model, clip_proc)
    vehicle_type = type_keys[(best_type_idx :=
type_probs.argmax().item())]
    type_conf = type_probs[best_type_idx].item()

    if type_conf < TRANSPORT_CONFIDENCE_THRESHOLD:
        raise ValueError("На зображенні не виявлено транспортного
засобу.")
    # — Крок 3: Марка/модель —
    brand_pool = BRAND_CANDIDATES.get(vehicle_type, [])
    if brand_pool:
        brand_probs = _clip_scores([b[0] for b in brand_pool],
cropped, clip_model, clip_proc)
        top3_vals, top3_idx = torch.topk(brand_probs, min(3,
len(brand_pool)))
        top3 = [(brand_pool[i][1], round(v.item() * 100, 1)) for i, v
in zip(top3_idx, top3_vals)]
    else:
        top3 = [(vehicle_type, round(type_conf * 100, 1))]

    if best_box:
        x1, y1, x2, y2 = [int(v) for v in best_box]
        bbox = {"x1": x1, "y1": y1, "x2": x2, "y2": y2, "obj_w": x2-
x1, "obj_h": y2-y1}
    else:
        bbox = {"x1": 0, "y1": 0, "x2": w, "y2": h, "obj_w": w,
"obj_h": h}

    return {
        "label": top3[0][0],
        "confidence": top3[0][1],
        "top3": top3,
        "vehicle_type": vehicle_type,
        "category": CATEGORY_MAP.get(vehicle_type, "Невідомо"),
        "emoji": EMOJI_MAP.get(vehicle_type, "🚗"),
        "yolo_class": yolo_class,
        "bbox": bbox,
        "area_pct": round(bbox["obj_w"] * bbox["obj_h"] / (w * h)
* 100, 1),
        "img_size": {"w": w, "h": h},
        "specs": SPECS_MAP.get(vehicle_type, {}),
    }

```