НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри комп'ютерних наук**
_____ Ігор БОЛБОТ	**_____ Белла ГОЛУБ**
(підпис)	(підпис)
«___» _____ 2026 р.	«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Програмне забезпечення системи ідентифікації об'єктів в полі зору відеокамери»

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент _____ **Ганна ВАЙГАНГ**
(підпис)

Керівник бакалаврської кваліфікаційної роботи
к.т.н., доцент _____ **Ганна ВАЙГАНГ**
(підпис)

Виконав _____ **Ілля ХИЖНЯК**
(підпис)

КИЇВ – 2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук
к.т.н., доцент _____ Белла ГОЛУБ
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

Хижняку Іллі Володимировичу

Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи
«Програмне забезпечення системи ідентифікації об'єктів в полі зору відеокамери»
затверджена наказом від «19» грудня 2025 р. № 3204 «С»

Термін подання завершеної роботи на кафедру «_22__» травня 2026 р.

Вихідні дані до роботи: Опис програмного забезпечення; десктоп-застосунок для виявлення та ідентифікації об'єктів у відеопотоці з локальним збереженням результатів.

Перелік питань що розглядаються: системний аналіз предметної області інформаційної системи; проєктування інформаційного та програмного забезпечення; розробка інформаційного та програмного забезпечення; рекомендації щодо впровадження та експлуатації системи

Перелік графічних документів (за необхідності).

Дата видачі завдання «_22_» ___грудня_____ 2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ **Ганна ВАЙГАНГ**
(підпис)

Завдання взяв до виконання _____ **Ілля ХИЖНЯК**
(підпис)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AI – Artificial Intelligence (штучний інтелект)

API – Application Programming Interface (інтерфейс програмування застосунків)

BGR – Blue–Green–Red (колірна модель OpenCV)

bbox – bounding box (обмежувальна рамка об’єкта)

CNN – Convolutional Neural Network (згорткова нейронна мережа)

COCO – Common Objects in Context (датасет для навчання моделей виявлення об’єктів)

CPU – Central Processing Unit (центральний процесор)

CUDA – Compute Unified Device Architecture (платформа GPU-обчислень Nvidia)

DB – Database (база даних)

FPS – Frames Per Second (кадрів за секунду)

GPU – Graphics Processing Unit (графічний процесор)

GUI – Graphical User Interface (графічний інтерфейс користувача)

IoU – Intersection over Union (метрика оцінювання перетину bounding box)

JPG/JPEG – Joint Photographic Experts Group (формат стиснення зображень)

JSON – JavaScript Object Notation (формат обміну даними)

mAP – Mean Average Precision (метрика оцінювання точності детектування)

ML – Machine Learning (машинне навчання)

MPS – Metal Performance Shaders (API апаратного прискорення Apple Silicon)

NMS – Non-Maximum Suppression (алгоритм усунення дубльованих обмежувальних рамок)

RGB – Red–Green–Blue (колірна модель зображень)

RDBMS – Relational Database Management System (реляційна система управління базами даних)

SQL – Structured Query Language (мова структурованих запитів)

SQLite – Self-Contained Lightweight Database Engine (вбудована реляційна система керування базами даних)

SVM – Support Vector Machine (метод опорних векторів)

UML – Unified Modeling Language (уніфікована мова моделювання)

YOLO – You Only Look Once (сімейство моделей виявлення об’єктів у реальному часі)

ВСТУП

У сучасному інформаційному суспільстві системи комп’ютерного зору займають важливе місце у широкому спектрі прикладних задач, пов’язаних з автоматизованим аналізом візуальної інформації [1]. Вони активно використовуються у системах безпеки та відеоспостереження, транспортних системах, промисловій автоматизації, робототехніці, медичних технологіях та інших галузях, де необхідне оперативне виявлення й ідентифікація об’єктів у реальному часі [5–7]. Розвиток алгоритмів глибинного навчання та сучасних моделей комп’ютерного зору значно підвищив ефективність автоматичної обробки відеоданих, що зробило системи аналізу відеопотоку одними з найбільш перспективних напрямів сучасної інженерії програмного забезпечення [13].

Одним із найбільш поширених підходів до реалізації задач виявлення об’єктів є використання нейромережевих моделей сімейства YOLO (You Only Look Once), які забезпечують високу швидкість та точність аналізу зображень і відеопотоку [2]. Сучасна версія YOLOv8 дозволяє здійснювати виявлення та класифікацію об’єктів у режимі реального часу із використанням оптимізованих алгоритмів глибинного навчання [3]. Використання бібліотек OpenCV та PyTorch забезпечує можливість ефективної реалізації алгоритмів комп’ютерного зору та нейронних мереж у програмних системах різного призначення [11, 17, 18].

Незважаючи на значний розвиток сучасних технологій комп’ютерного зору, більшість комерційних програмних рішень для автоматичного виявлення об’єктів мають ряд обмежень. Значна частина таких систем потребує використання хмарних сервісів, характеризується закритістю програмного коду, високими вимогами до апаратного забезпечення або пов’язана з необхідністю придбання ліцензійного програмного забезпечення [5–7]. Це обумовлює потребу у створенні доступних, автономних та гнучких програмних рішень, здатних функціонувати на сучасному споживчому обладнанні без використання зовнішніх серверних платформ.

Актуальність теми бакалаврської кваліфікаційної роботи полягає у необхідності розробки програмного забезпечення системи ідентифікації об’єктів у полі зору відеокамери, яке забезпечує локальну обробку відеопотоку, автоматичне виявлення та класифікацію об’єктів у режимі реального часу, а також збереження результатів роботи у локальній базі даних.

Об’єктом дослідження є процес автоматичного виявлення та класифікації об’єктів у цифровому відеопотоці з використанням методів глибинного навчання.

Предметом дослідження є методи та засоби розробки програмного забезпечення системи ідентифікації об’єктів у відеопотоці на основі нейронної мережі YOLOv8 із локальним збереженням результатів.

Метою бакалаврської кваліфікаційної роботи є проєктування та реалізація програмного забезпечення системи ідентифікації об’єктів у полі зору відеокамери, яке забезпечує обробку відеоданих у реальному часі на платформі macOS Apple Silicon, надає графічний інтерфейс користувача та здійснює структуроване збереження результатів у локальній базі даних.

Для досягнення поставленої мети необхідно виконати такі завдання:

- провести аналіз предметної області, сучасних методів виявлення об’єктів у відеопотоці та існуючих програмних рішень;
- сформулювати функціональні та нефункціональні вимоги до розроблюваного програмного забезпечення;
- виконати проєктування системи із побудовою UML-моделей та логічної моделі даних;
- реалізувати модуль виявлення та класифікації об’єктів на базі моделі YOLOv8;
- розробити підсистему збереження даних на основі SQLite та графічний інтерфейс користувача засобами tkinter;

- провести тестування програмного забезпечення, проаналізувати результати роботи системи та сформулювати рекомендації щодо її впровадження і подальшого розвитку.

У роботі застосовано методи системного аналізу, UML-моделювання, об’єктно-орієнтованого проєктування, компонентного моделювання, цифрової обробки зображень та тестування програмного забезпечення [1, 4, 11]. Для програмної реалізації використано мову програмування Python [12], бібліотеки OpenCV [17], PyTorch [18], Pillow [19], а також засоби роботи з базами даних SQLite [10, 15].

Практична значущість отриманих результатів полягає у створенні програмного забезпечення, яке може бути використане у системах відеоспостереження, автоматизованого контролю та моніторингу для виявлення та ідентифікації об’єктів у режимі реального часу. Розроблена система забезпечує автономну локальну обробку відеоданих, збереження результатів аналізу та можливість подальшого розширення функціональних можливостей програмного забезпечення.

Структура бакалаврської кваліфікаційної роботи складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків. У першому розділі розглянуто предметну область, визначено актуальність теми, проведено огляд існуючих рішень та сформульовано постановку завдання. Другий розділ присвячено проєктуванню програмної системи, побудові логічної моделі даних, UML-діаграм та архітектури системи. У третьому розділі наведено опис реалізації програмних модулів системи виявлення та ідентифікації об’єктів. Четвертий розділ містить результати тестування, рекомендації щодо впровадження та експлуатації програмного забезпечення, а також вимоги до програмного й апаратного забезпечення системи.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ІНФОРМАЦІЙНОЇ СИСТЕМИ

1.1 Опис предметної області

Комп'ютерний зір є галуззю штучного інтелекту, що займається розробкою методів і систем, які надають машинам здатність отримувати, аналізувати та інтерпретувати інформацію з цифрових зображень і відео. Однією з найважливіших прикладних задач у цій галузі є виявлення та ідентифікація об'єктів у відеопотоці - процес автоматичного розпізнавання та класифікації об'єктів реального світу на послідовних кадрах відеозапису.

Системи виявлення об'єктів у відеопотоці набули надзвичайно широкого застосування в різноманітних сферах: безпека та контроль доступу, інтелектуальне відеоспостереження, промислова автоматизація, транспортна аналітика, наукові дослідження та побутові застосунки. Ключовою перевагою таких систем є здатність автоматично виконувати аналіз, що традиційно потребував постійної присутності людини-оператора, - із вищою стабільністю та значно меншими витратами.

Сучасні підходи до виявлення об'єктів поділяються на три основні категорії. Класичні алгоритми комп'ютерного зору, зокрема метод віднімання фону, оператори Canny та Sobel для виявлення контурів, алгоритми SIFT/SURF/ORB для виявлення ключових точок, є обчислювально ефективними, проте характеризуються низькою точністю в умовах змінного освітлення, складного фону та варіацій форми об'єктів. Методи традиційного машинного навчання - SVM, k-NN, Random Forest - потребують ручного виокремлення ознак і демонструють обмежену точність при широкому розмаїтті об'єктів. Методи глибинного навчання на основі згорткових нейронних мереж (CNN), зокрема архітектури YOLO, Faster R-CNN і SSD, забезпечують найвищу точність і швидкість обробки навіть у складних умовах зйомки, що робить їх стандартом де-факто для задач виявлення об'єктів у реальному часі [1].

Окремої уваги заслуговує сімейство моделей YOLO (You Only Look Once), яке вперше запропонувало підхід до виявлення об'єктів за один прохід нейронної мережі крізь зображення. На відміну від двоетапних методів, таких як Faster R-CNN, де спочатку генеруються регіони-кандидати, а потім класифікуються, YOLO поєднує обидва процеси в одній мережі, що забезпечує значно вищу швидкість обробки без суттєвої втрати точності [2]. Актуальна версія YOLOv8 від компанії Ultralytics демонструє показники mAP@0.5 близько 58% на датасеті COCO при обробці 30–120 кадрів на секунду залежно від апаратного забезпечення та розміру моделі [3].

З практичної точки зору, предметна область даної роботи охоплює розробку десктоп-застосунку, що виконує такі функції: отримання відеопотоку з вбудованої камери пристрою; обробку кадрів у реальному часі засобами нейронної мережі; виявлення та класифікацію об'єктів; відображення результатів у графічному інтерфейсі; збереження анотованих кадрів і метаданих у локальну базу даних. Цільовою апаратною платформою виступає MacBook Pro 2023 на базі чіпа Apple M2, а ключовою вимогою - повністю автономна робота без підключення до мережі Інтернет.

Важливим аспектом предметної області є проблема уникнення дублікатів при збереженні результатів. У реальних сценаріях використання один і той самий об'єкт може перебувати у полі зору камери протягом тривалого часу. Для запобігання надмірному накопиченню записів у базі даних необхідно реалізувати алгоритм відстеження об'єктів між кадрами, що дозволить зберігати запис лише при першій появі об'єкта або після закінчення встановленого тайм-ауту його відсутності.

1.2 Аналіз вимог до програмної системи

На основі проведеного аналізу предметної області, сучасних методів комп’ютерного зору та існуючих програмних рішень було сформовано вимоги до програмного забезпечення системи ідентифікації об’єктів у полі зору

відеокамери. Визначення вимог є одним із ключових етапів проєктування програмної системи, оскільки саме вони формують основу для подальшої розробки архітектури, структури даних, програмних модулів та механізмів взаємодії компонентів системи.

Розроблювана система орієнтована на автоматизовану обробку відеопотоку у режимі реального часу із використанням алгоритмів глибинного навчання та моделі YOLOv8. Основними задачами програмного забезпечення є захоплення відеоданих з камери, виявлення об’єктів на окремих кадрах, визначення їх класу, відображення результатів аналізу та збереження інформації про виконані ідентифікації у локальній базі даних. Важливим аспектом функціонування системи є забезпечення автономної роботи без необхідності використання зовнішніх серверних платформ або хмарних сервісів.

У процесі формування вимог було враховано особливості цільового програмного середовища, специфіку роботи із відеопотоком, необхідність забезпечення швидкодії та стабільності функціонування системи, а також вимоги до зручності взаємодії користувача з програмним забезпеченням.

Додатково під час формування вимог було враховано необхідність забезпечення масштабованості та можливості подальшого розширення функціональності системи. Архітектура програмного забезпечення повинна підтримувати інтеграцію нових моделей комп’ютерного зору, розширення переліку підтримуваних джерел відеопотоку, удосконалення механізмів аналізу даних та підключення додаткових модулів обробки інформації без суттєвого перепроєктування основних компонентів системи. Такий підхід дозволяє підвищити адаптивність програмного забезпечення до змін вимог користувачів і забезпечує перспективу подальшого розвитку системи відповідно до сучасних тенденцій у сфері комп’ютерного зору та інтелектуального аналізу відеоданих.

Функціональні вимоги визначають перелік основних можливостей системи та описують поведінку програмного забезпечення під час виконання задач виявлення й ідентифікації об’єктів. До функціональних вимог належать

засоби роботи з відеопотоком, механізми аналізу кадрів, збереження результатів роботи та взаємодія користувача із системою.

Функціональні вимоги до системи наведено в табл. 1.1.

Таблиця 1.1

Функціональні вимоги до системи

№	Вимога	Опис вимоги
1	Захоплення відеопотоку	Система повинна отримувати відеопотік з камери пристрою та відображати його у вікні застосунку.
2	Виявлення об'єктів у реальному часі	Система повинна виявляти об'єкти на кожному кадрі відеопотоку з використанням моделі YOLOv8.
3	Класифікація об'єктів	Система повинна визначати клас кожного виявленого об'єкта з набору 80 класів датасету COCO.
4	Фільтрація за порогом впевненості	Система повинна відкидати виявлення з рівнем впевненості моделі нижче встановленого порогу (CONFIDENCE_THRESHOLD = 0.8).
5	Запобігання дублікатам записів	Система не повинна створювати повторні записи для того самого об'єкта, що вже перебуває у полі зору та зареєстрований.
6	Збереження результатів	При першому виявленні нового об'єкта система повинна зберігати анотований кадр у форматі JPG.
7	Запис до бази даних	Метадані кожної ідентифікації (клас, мітка часу, шлях до файлу) повинні зберігатися у базі даних SQLite.
8	Перегляд результатів	Користувач повинен мати змогу переглядати збережені записи у табличному вигляді з можливістю пошуку та сортування.
9	Перегляд збереженого зображення	Подвійний клік на запис у таблиці повинен відкривати відповідне збережене зображення.
10	Відображення FPS	Система повинна відображати поточну швидкість обробки відеопотоку (кадрів на секунду) на відео-фреймі.
11	Вибір камери	Система повинна автоматично виявляти доступні камери та надавати можливість обрати потрібну перед запуском.

Окрім функціональних можливостей, важливе значення мають нефункціональні вимоги, які визначають характеристики якості програмного забезпечення, зокрема продуктивність, надійність, портативність та зручність використання. Дотримання нефункціональних вимог забезпечує стабільну

роботу системи, ефективне використання ресурсів та можливість практичного застосування програмного забезпечення на цільовій платформі.

Нефункціональні вимоги до системи наведено в табл. 1.2.

Таблиця 1.2

Нефункціональні вимоги до системи

№	Вимога	Опис вимоги
1	Продуктивність	Система повинна забезпечувати обробку відеопотоку зі швидкістю не менше 10 кадрів на секунду на цільовій платформі (MacBook Pro M2) без GPU-прискорення.
2	Зручність використання	Графічний інтерфейс повинен бути інтуїтивним, мінімалістичним і не перевантаженим зайвими елементами.
3	Надійність	Система повинна коректно обробляти помилки підключення камери, відсутність кадру та збої бібліотек без аварійного завершення.
4	Портативність	Система повинна запускатися на macOS Apple Silicon без необхідності мережевого підключення після першого встановлення.
5	Цілісність даних	База даних повинна зберігати узгодженість записів, не допускаючи появи записів без відповідного файлу зображення.
6	Простота розгортання	Система повинна встановлюватися через стандартний менеджер пакетів pip і запускатися єдиним файлом main.py.

Сформований перелік функціональних та нефункціональних вимог дозволяє визначити основні характеристики програмної системи, встановити межі її функціонування та забезпечити основу для подальшого проєктування архітектури, структури даних і програмних модулів. Визначені вимоги також дозволяють оцінити ефективність реалізації системи під час подальшого тестування та впровадження програмного забезпечення.

1.3 Моделювання предметної області

Для формального опису структури та поведінки програмної системи використовується UML-підхід (Unified Modeling Language), який є одним із найбільш поширених засобів моделювання в об’єктно-орієнтованому

проєктуванні програмного забезпечення [4]. Використання UML дозволяє виконати формалізацію вимог до системи, визначити взаємодію між її компонентами, описати поведінку користувачів та сформувати основу для подальшого проєктування архітектури програмного забезпечення. Застосування UML-моделей є важливим етапом розробки складних програмних систем, оскільки забезпечує структуроване представлення функціональних можливостей, логіки роботи та взаємозв’язків між окремими модулями системи.

Моделювання предметної області системи виявлення та ідентифікації об’єктів у відеопотоці здійснюється з урахуванням особливостей функціонування програмного забезпечення, орієнтованого на обробку відеоданих у режимі реального часу. Основними компонентами системи є модуль захоплення відеопотоку, модуль аналізу кадрів, модуль виявлення та класифікації об’єктів на базі нейронної мережі YOLOv8, підсистема збереження результатів та графічний інтерфейс користувача [2, 3]. Взаємодія зазначених компонентів забезпечує автоматизований цикл обробки відеоданих, починаючи від отримання кадру з відеокамери та завершуючи збереженням результатів ідентифікації у локальній базі даних.

Особливу увагу під час моделювання предметної області приділено організації інформаційних потоків між окремими компонентами системи. У процесі роботи програмне забезпечення здійснює безперервний цикл обробки відеоданих, який включає отримання кадрів із камери, попередню обробку зображення, виконання аналізу нейронною мережею, формування результатів класифікації та збереження інформації про виявлені об’єкти. Такий підхід дозволяє забезпечити узгоджену взаємодію між модулями системи, оптимізувати процес обробки відеопотоку та підвищити ефективність роботи програмного забезпечення в режимі реального часу [2, 11, 17].

Під час моделювання предметної області було визначено основних акторів системи та сформовано перелік прецедентів, що описують взаємодію користувача із програмним забезпеченням. Центральним актором системи є користувач, який здійснює керування процесом виявлення об’єктів, взаємодіє з

графічним інтерфейсом та переглядає результати роботи системи. Додатковим зовнішнім актором виступає джерело відеопотоку, яке забезпечує надходження відеоданих до програмної системи незалежно від дій користувача.

Основні прецеденти системи наведено в табл. 1.3.

Таблиця 1.3

Основні прецеденти системи

№	Прецедент	Опис
1	Запуск виявлення	Користувач обирає камеру зі списку та натискає кнопку «Старт». Система відкриває камеру та запускає цикл обробки кадрів.
2	Зупинка виявлення	Користувач натискає кнопку «Стоп». Система завершує цикл обробки та звільняє ресурси камери.
3	Перегляд відеопотоку	Система відображає відеопотік у реальному часі з накладеними обмежувальними рамками та підписами класів виявлених об'єктів.
4	Виявлення та ідентифікація об'єкта	Модуль комп'ютерного зору обробляє кожен кадр і визначає наявність, клас і положення об'єктів.
5	Збереження результату	При виявленні нового об'єкта система автоматично зберігає анотований кадр і запис до бази даних. Є розширенням прецеденту «Виявлення та ідентифікація об'єкта».
6	Відкриття бази даних	Користувач натискає кнопку «База даних». Система відкриває окреме вікно з таблицею збережених ідентифікацій.
7	Пошук та фільтрація записів	Користувач вводить текст у поле пошуку. Система фільтрує записи у таблиці в реальному часі.
8	Перегляд збереженого зображення	Користувач робить подвійний клік на запис у таблиці. Система відкриває відповідне анотоване зображення.

На рис. 1.1 наведено діаграму прецедентів системи виявлення та ідентифікації об’єктів, яка відображає основні сценарії взаємодії користувача та зовнішніх компонентів із програмною системою. Діаграма демонструє функціональні можливості застосунку, взаємозв’язки між окремими прецедентами та послідовність виконання основних операцій, пов’язаних із запуском обробки відеопотоку, виявленням об’єктів, збереженням результатів і переглядом інформації у базі даних.

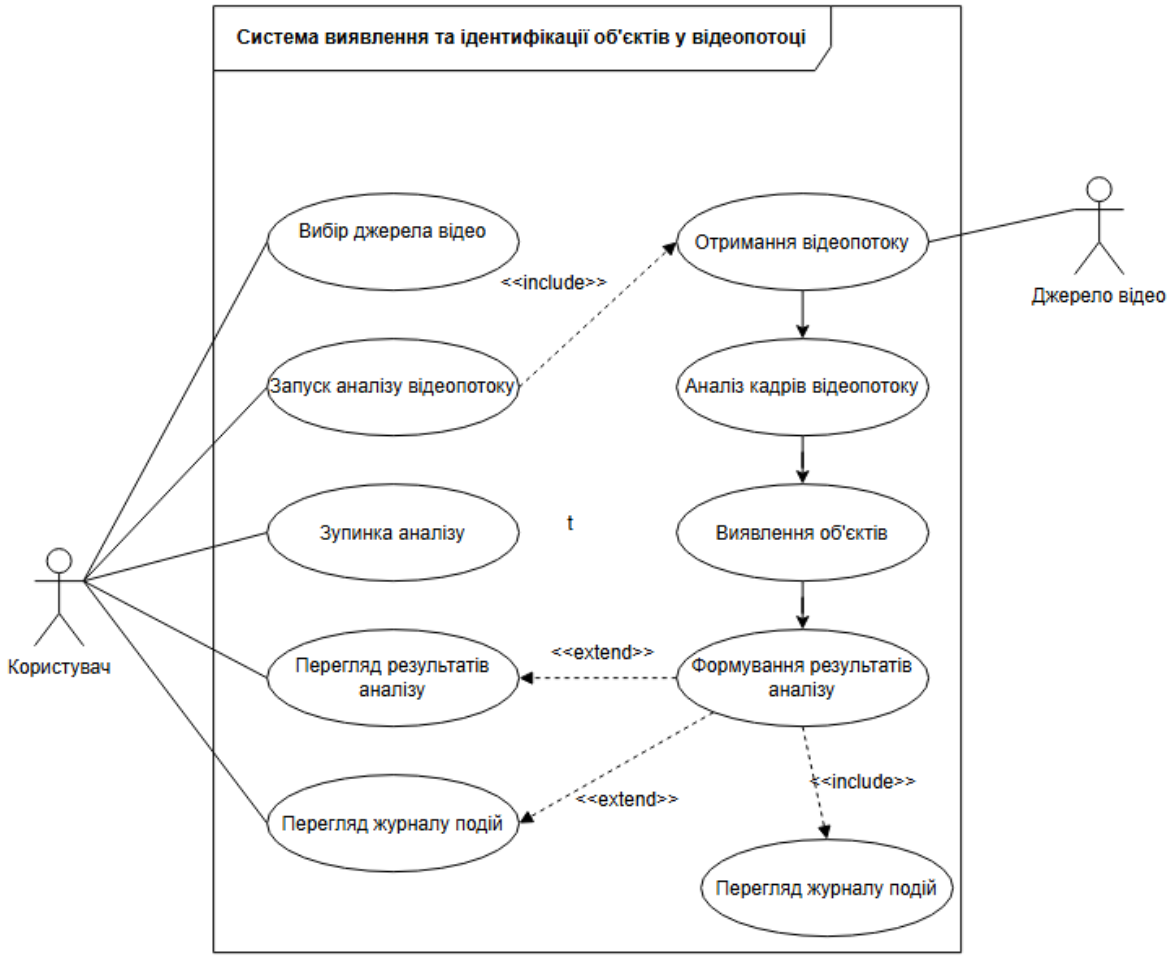

Рис. 1.1 Діаграма прецедентів системи виявлення та ідентифікації об’єктів

Діаграма прецедентів відображає основні сценарії взаємодії користувача із програмною системою та демонструє взаємозв’язки між окремими функціональними можливостями застосунку. Більшість прецедентів ініціюється безпосередньо користувачем через графічний інтерфейс системи. Водночас процес виявлення та ідентифікації об’єктів виконується автоматично після запуску обробки відеопотоку та функціонує без постійного втручання користувача.

Особливістю розробленої моделі є використання механізму розширення між прецедентами «Виявлення та ідентифікація об’єктів» та «Збереження результатів». Збереження інформації виконується лише у випадку першого виявлення нового об’єкта у полі зору камери, що дозволяє уникнути дублювання записів та надмірного накопичення графічних файлів у системі. Такий підхід забезпечує підвищення ефективності роботи програмного забезпечення,

оптимізацію використання дискового простору та збереження цілісності інформації у базі даних.

У процесі моделювання предметної області також було враховано необхідність забезпечення масштабованості та модульності архітектури програмного забезпечення. Використання UML-моделювання дозволяє формалізувати структуру системи та створити основу для подальшого проєктування класів, пакетів, компонентів і механізмів взаємодії між окремими програмними модулями [4]. Отримані результати моделювання використовуються на наступних етапах проєктування інформаційного забезпечення та програмної реалізації системи.

1.4 Огляд інформаційних джерел та існуючих рішень

Одним із важливих етапів розробки програмного забезпечення є аналіз сучасних інформаційних джерел та існуючих програмних рішень, що використовуються для виявлення та ідентифікації об’єктів у відеопотоці. Проведення такого аналізу дозволяє визначити актуальні підходи до реалізації систем комп’ютерного зору, оцінити переваги та недоліки наявних рішень, а також сформувати обґрунтовані вимоги до проєктованої системи [1, 4].

Сучасні системи аналізу відеоданих базуються на використанні алгоритмів цифрової обробки зображень, методів машинного навчання та моделей глибинних нейронних мереж [13]. Одним із найбільш поширених підходів до реалізації задач виявлення об’єктів є використання алгоритмів сімейства YOLO, які забезпечують високу швидкість обробки відеопотоку та достатній рівень точності ідентифікації об’єктів у режимі реального часу [2]. Зокрема, модель YOLOv8 є однією з сучасних реалізацій алгоритму, оптимізованою для задач комп’ютерного зору та інтеграції у прикладні програмні системи [3].

Для формування обґрунтованих проєктних рішень проведено аналіз найбільш поширених програмних систем виявлення та ідентифікації об’єктів у

відеопотоці, які використовуються у сфері відеоспостереження, аналітики та автоматизованого моніторингу.

Одним із відомих рішень є система Rekor Scout (раніше OpenALPR), яка призначена для автоматичного розпізнавання номерних знаків транспортних засобів у режимі реального часу [5]. Система використовує алгоритми комп'ютерного зору та машинного навчання для аналізу відеопотоку з камер спостереження. Програмне забезпечення дозволяє визначати номерний знак, марку, модель та колір транспортного засобу, а також зберігати результати аналізу разом із часовими й просторовими мітками. Додатковою перевагою системи є наявність відкритого API для інтеграції із зовнішніми інформаційними системами та підтримка локального й хмарного розгортання. Основним недоліком Rekor Scout є вузька спеціалізація, орієнтована виключно на транспортні засоби, що обмежує можливість використання системи для задач загального виявлення об'єктів.

Іншим поширеним рішенням є Milestone XProtect – професійна платформа управління відеоспостереженням (VMS), призначена для централізованої роботи з великою кількістю відеокамер [6]. Система підтримує інтеграцію з аналітичними модулями сторонніх виробників, що дозволяє реалізовувати функції розпізнавання облич, відстеження об'єктів, виявлення руху та аналізу поведінки. Перевагами платформи є масштабованість, підтримка корпоративної інфраструктури та широкі можливості налаштування. Водночас використання Milestone XProtect пов'язане з високою вартістю ліцензій, складністю налаштування та необхідністю використання потужної серверної інфраструктури, що ускладнює її застосування на звичайному користувацькому обладнанні.

Серед хмарних рішень значного поширення набула платформа Amazon Rekognition від Amazon Web Services [7]. Дана система використовує алгоритми глибинного навчання для автоматичного аналізу зображень та відео, забезпечуючи виявлення об'єктів, розпізнавання облич, аналіз сцен та текстової інформації. Платформа інтегрується з іншими сервісами AWS, зокрема Amazon

S3, Lambda та DynamoDB, що дозволяє реалізовувати масштабовані хмарні рішення. Основними недоліками Amazon Rekognition є залежність від мережевого підключення, необхідність використання зовнішньої хмарної інфраструктури, тарифікація за кількість API-запитів та потенційні ризики, пов’язані із конфіденційністю даних.

Порівняльний аналіз розглянутих систем та проєктованого програмного забезпечення наведено в табл. 1.4.

Таблиця 1.4

Порівняльний аналіз систем виявлення об'єктів

Характеристика	Rekor Scout	Milestone XProtect	Amazon Rekognition	Проєктована система
Тип розгортання	Локальне / хмарне	Локальне (сервер)	Хмарне (AWS)	Локальне (десктоп)
Ліцензія	Комерційна	Комерційна	Комерційна (pay-per-use)	Відкрите ПЗ
Потребує Інтернет	Частково	Ні	Так	Ні
Класи об'єктів	Транспорт	Обличчя, рух	80+ класів	80 класів COCO
Складність встановлення	Середня	Висока	Низька (API)	Низька (pip)
Зберігання результатів	Хмара / локально	Сервер	AWS S3	Локальна SQLite
Вартість впровадження	Висока	Висока	Середня	Мінімальна

На основі проведеного аналізу можна зробити висновок, що існуючі програмні рішення не забезпечують повного поєднання таких характеристик, як автономність роботи, відкритість програмного забезпечення, підтримка широкого набору класів об’єктів, мінімальні вимоги до інфраструктури та простота розгортання на споживчому обладнанні. Більшість професійних платформ орієнтовані на корпоративне використання та потребують значних фінансових витрат або використання хмарної інфраструктури.

Отримані результати аналізу підтверджують доцільність розробки власного програмного забезпечення системи ідентифікації об’єктів у полі зору

відеокамери, яке забезпечуватиме локальну автономну роботу, використання сучасних алгоритмів комп’ютерного зору на базі YOLOv8, підтримку широкого набору класів об’єктів та можливість ефективного функціонування на сучасному користувацькому обладнанні.

1.5 Постановка завдання

На основі проведеного аналізу предметної області, дослідження сучасних методів комп’ютерного зору, аналізу функціональних та нефункціональних вимог, а також огляду існуючих програмних рішень було сформульовано постановку завдання бакалаврської кваліфікаційної роботи. Основною метою розробки є створення програмного забезпечення системи ідентифікації об’єктів у полі зору відеокамери, здатного забезпечувати автоматизовану обробку відеопотоку у режимі реального часу із використанням сучасних алгоритмів глибинного навчання [2, 3].

Проєктована система повинна забезпечувати отримання відеоданих із вбудованої або зовнішньої камери пристрою, виконувати аналіз кожного кадру відеопотоку, здійснювати виявлення та класифікацію об’єктів, а також зберігати результати роботи у локальній базі даних. Важливою особливістю системи є її автономність, що передбачає можливість функціонування без використання зовнішніх хмарних сервісів та постійного мережевого підключення. Такий підхід дозволяє забезпечити підвищення швидкодії, конфіденційності даних та незалежності роботи програмного забезпечення від сторонньої інфраструктури.

У межах бакалаврської кваліфікаційної роботи необхідно розробити програмне забезпечення системи виявлення та ідентифікації об’єктів у відеопотоці, яке реалізує:

- автоматичне захоплення та обробку відеопотоку у режимі реального часу;
- виявлення, класифікацію та відстеження об’єктів із використанням нейронної мережі YOLOv8 [3];
- фільтрацію результатів виявлення та запобігання дублюванню записів;

- збереження анотованих зображень і метаданих у локальній базі даних SQLite [10, 15];
- графічний інтерфейс користувача для перегляду результатів роботи системи;
- автономну роботу та спрощене встановлення програмного забезпечення засобами стандартного Python-середовища.

Під час розробки системи необхідно забезпечити високу швидкість обробки відеопотоку, стабільність роботи програмного забезпечення та коректну взаємодію між окремими програмними модулями. Особливу увагу необхідно приділити ефективності роботи алгоритмів комп'ютерного зору, оптимізації процесу обробки кадрів та зменшенню навантаження на апаратні ресурси системи.

Цільовою платформою програмного забезпечення є операційна система macOS версії 13.0 і вище на базі процесорів Apple Silicon. Реалізація програмної системи виконується мовою програмування Python [12] із використанням бібліотек OpenCV [17], ultralytics YOLOv8 [3], PyTorch [18], tkinter [16] та SQLite [10]. Використання зазначених технологій дозволяє забезпечити ефективну реалізацію алгоритмів комп'ютерного зору, створення графічного інтерфейсу користувача та організацію локального збереження результатів роботи системи.

Результатом виконання бакалаврської кваліфікаційної роботи має стати програмне забезпечення системи ідентифікації об'єктів у полі зору відеокамери, яке забезпечуватиме автоматичне виявлення та класифікацію об'єктів у відеопотоці, збереження результатів аналізу та можливість подальшого використання системи у задачах моніторингу, відеоспостереження та автоматизованого контролю.

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Проєктування інформаційного забезпечення програмної системи є одним із ключових етапів розробки застосунку, оскільки саме структура даних визначає особливості їх зберігання, обробки та подальшого використання в програмних модулях системи. Для формалізації структури інформаційної бази було використано підхід ER-моделювання (Entity–Relationship), який дозволяє представити основні сутності предметної області, їх атрибути та взаємозв’язки між ними [8].

Інформаційна система орієнтована на збереження результатів автоматичного виявлення об’єктів на зображеннях. У процесі функціонування система накопичує два основних типи даних: структуровані метадані про результати ідентифікації та графічні файли із зафіксованими результатами детекції. Метадані зберігаються у реляційній базі даних SQLite, тоді як анотовані зображення розміщуються у файловій системі застосунку.

Логічна модель системи містить одну основну сутність - таблицю detections. Вона фіксує кожен випадок виявлення нового об’єкта і складається з чотирьох полів: унікального числового ідентифікатора запису, назви класу об’єкта, часової мітки виявлення та шляху до збереженого зображення.

Пов’язане сховище зображень є другим рівнем зберігання: директорія detections/ містить JPG-файли, кожен з яких відповідає рівно одному запису в таблиці. Зв’язок між ними підтримується через поле image_path - відношення один до одного між записом і файлом.

ER-діаграма логічної моделі даних системи наведена на рис. 2.1.

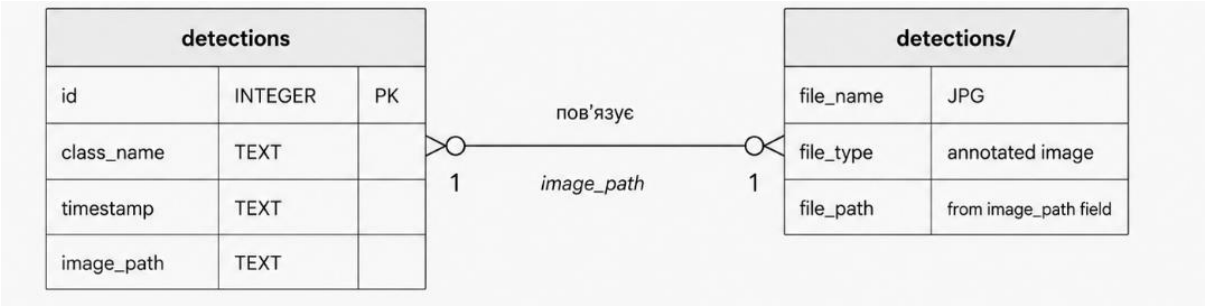

Рис. 2.1. ER-діаграма логічної моделі даних системи

Мінімалістична схема з однієї таблиці та файлової системи обрана навмисно: відсутність зовнішніх ключів і зв'язаних таблиць забезпечує максимальну портативність. Вся база даних є єдиним файлом detections.db, що дозволяє переносити або резервувати її одним копіюванням.

Запропонована структура інформаційної бази відповідає базовим принципам реляційного проєктування. У таблиці відсутні надлишкові залежності між атрибутами, а всі поля містять атомарні значення, що дозволяє вважати структуру такою, що відповідає вимогам третьої нормальної форми. Використання компактної схеми даних забезпечує простоту реалізації, підвищує портативність програмного забезпечення та зменшує складність супроводу системи.

Додатковою перевагою обраного підходу є використання SQLite у вигляді єдиного файлу detections.db, що спрощує процес розгортання застосунку, резервного копіювання інформації та перенесення системи між різними програмно-апаратними платформами.

2.2 Діаграма класів та кооперації

Для опису статичної структури програмного забезпечення системи виявлення та ідентифікації об'єктів у відеопотоці було розроблено UML-діаграму класів. Вона дозволяє відобразити основні програмні сутності системи, їхні атрибути, методи та зв'язки між ними. Така діаграма є важливою частиною

проєктування, оскільки показує, які об’єкти беруть участь у роботі системи та як вони взаємодіють між собою на рівні програмної логіки.

У межах системи було виділено кілька основних класів: Користувач, ДжерелоВідео, ПотікВідео, АналізаторКадрів, ВиявленийОб’єкт, РезультатАналізу та ЖурналПодій. Кожен із цих класів відповідає за окрему частину функціональності системи та має власний набір властивостей і операцій.

Клас Користувач описує особу, яка взаємодіє із системою через графічний інтерфейс. Він містить такі атрибути, як ідентифікатор користувача, ім’я користувача, роль та електронна пошта. До основних методів цього класу належать початок аналізу, перегляд результатів, перегляд журналу подій та керування налаштуваннями. Цей клас показує, що користувач є ініціатором основних дій у системі.

Клас ДжерелоВідео відповідає за представлення джерела, з якого система отримує відеодані. Таким джерелом може бути камера комп’ютера, USB-камера, відеофайл або мережевий потік. Клас містить інформацію про ідентифікатор джерела, тип, місце знаходження, роздільну здатність, частоту кадрів і статус підключення. Основними методами є підключення, відключення та отримання кадру. Через цей клас система встановлює зв’язок із джерелом відеопотоку.

Клас ПотікВідео описує активний відеопотік, який формується на основі вибраного джерела відео. Він містить ідентифікатор потоку, ідентифікатор джерела, статус потоку та час його початку. Основними методами є запуск потоку, зупинка потоку та отримання кадру. Саме цей клас забезпечує безперервну передачу кадрів для подальшої обробки.

Клас АналізаторКадрів реалізує логіку аналізу відеокадрів. Він містить ідентифікатор аналізатора, назву моделі, поріг впевненості та поточний статус аналізатора. Основним методом цього класу є проаналізувати кадр, результатом якого є інформація про виявлені об’єкти. У межах даної системи цей клас пов’язаний із використанням моделі комп’ютерного зору, яка виконує пошук об’єктів у кадрі.

Клас ВиявленийОб’єкт описує окремий об’єкт, знайдений у відеопотоці. До його атрибутів належать ідентифікатор об’єкта, тип об’єкта, межова рамка, рівень впевненості, час першого виявлення та час останнього виявлення. Метод отримання інформації дозволяє сформувати опис знайденого об’єкта для подальшого відображення, збереження або включення до результату аналізу.

Клас РезультатАналізу використовується для збереження підсумкової інформації про виконаний аналіз. Він містить ідентифікатор результату, ідентифікатор потоку, часову мітку та кількість виявлених об’єктів. До його методів належать отримання списку об’єктів і збереження результату. Цей клас поєднує інформацію про відеопотік та знайдені об’єкти, формуючи цілісний результат роботи системи.

Клас ЖурналПодій відповідає за фіксацію службових повідомлень і подій, які виникають під час роботи системи. Він містить ідентифікатор запису, часову мітку, тип події, опис події та ідентифікатор користувача. Метод додавання події дозволяє записувати інформацію про запуск аналізу, успішне виявлення об’єктів, помилки підключення до джерела відео або інші важливі дії системи.

Зв’язки між класами показують логіку роботи програмної системи. Користувач обирає джерело відео, після чого на його основі формується відеопотік. Потік відео передає кадри до аналізатора, який виконує їх обробку та виявляє об’єкти. Виявлені об’єкти входять до складу результату аналізу, а результат може бути збережений і відображений користувачу. Окремо ведеться журнал подій, у якому фіксуються важливі етапи роботи системи та можливі помилки.

Для формалізації структури програмної системи та опису взаємодії між її основними компонентами було побудовано діаграму класів і кооперації. Дана UML-модель відображає ключові класи системи, їх атрибути, методи та зв’язки між об’єктами, що забезпечують реалізацію функціональних можливостей програмного застосунку. Крім того, діаграма дозволяє визначити логіку обміну даними між окремими модулями та сформувати цілісне уявлення про

архітектуру програмного забезпечення. Загальну структуру взаємодії класів і кооперацій наведено на рис. 2.2.

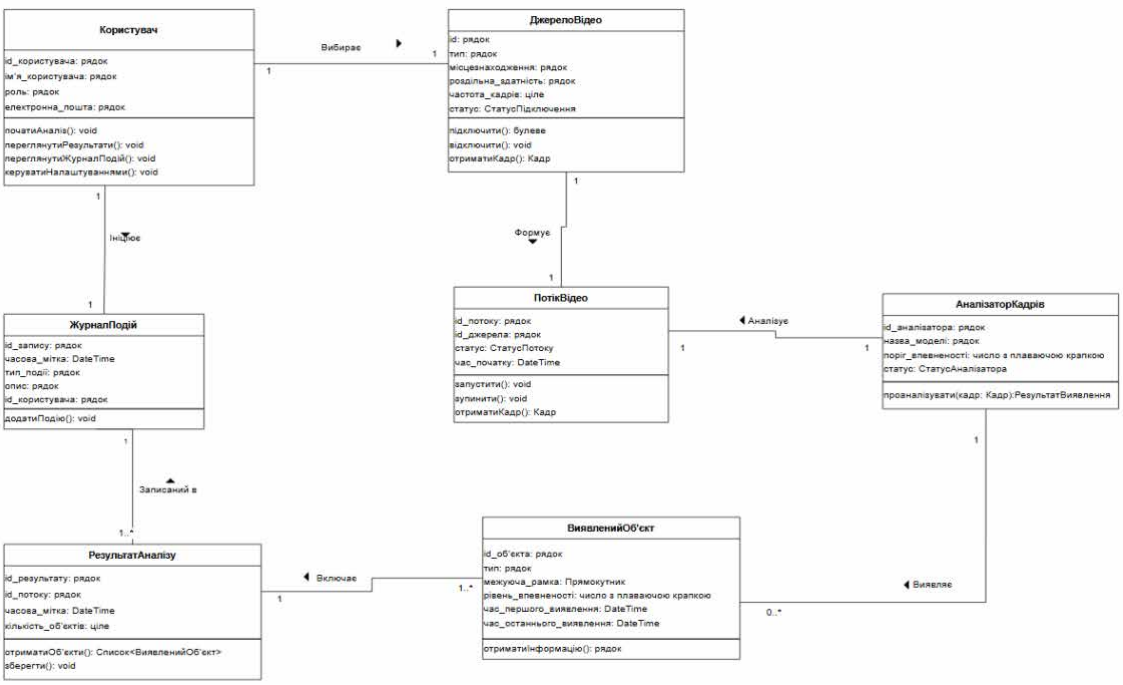

Рис. 2.2. Діаграма класів програмного забезпечення системи

Діаграма класів демонструє основну об’єктну структуру системи. Вона показує, які класи необхідні для реалізації функціональності програмного забезпечення, які дані вони зберігають, які операції виконують та яким чином пов’язані між собою.

Для більш детального опису взаємодії об’єктів системи було розроблено діаграми кооперації. На відміну від діаграми класів, яка показує статичну структуру системи, діаграми кооперації відображають, як саме об’єкти взаємодіють між собою під час виконання конкретних сценаріїв. Вони дозволяють краще зрозуміти послідовність обміну повідомленнями між класами та роль кожного об’єкта у виконанні окремої функції.

У межах системи було виділено три основні кооперації: отримання відеоданих, аналіз відеопотоку, а також формування результатів і журналу. Кожна з цих кооперацій описує окремий етап роботи системи та демонструє взаємодію між класами, які беруть участь у його виконанні.

Перша кооперація описує процес отримання відеоданих. У цьому сценарії основними учасниками є Користувач, ДжерелоВідео та ПотікВідео. Спочатку користувач обирає джерело відеоданих у системі. Після цього система звертається до класу ДжерелоВідео, який перевіряє доступність вибраного джерела та виконує підключення. Якщо підключення успішне, на основі джерела формується об’єкт ПотікВідео, який відповідає за подальше отримання кадрів.

Ця кооперація показує початковий етап роботи системи, коли користувач ініціює процес аналізу, а система готує відеопотік для подальшої обробки. Вона також демонструє залежність потоку відео від джерела: без доступного джерела неможливо сформувати потік і передати кадри до аналізатора (рис. 2.3).

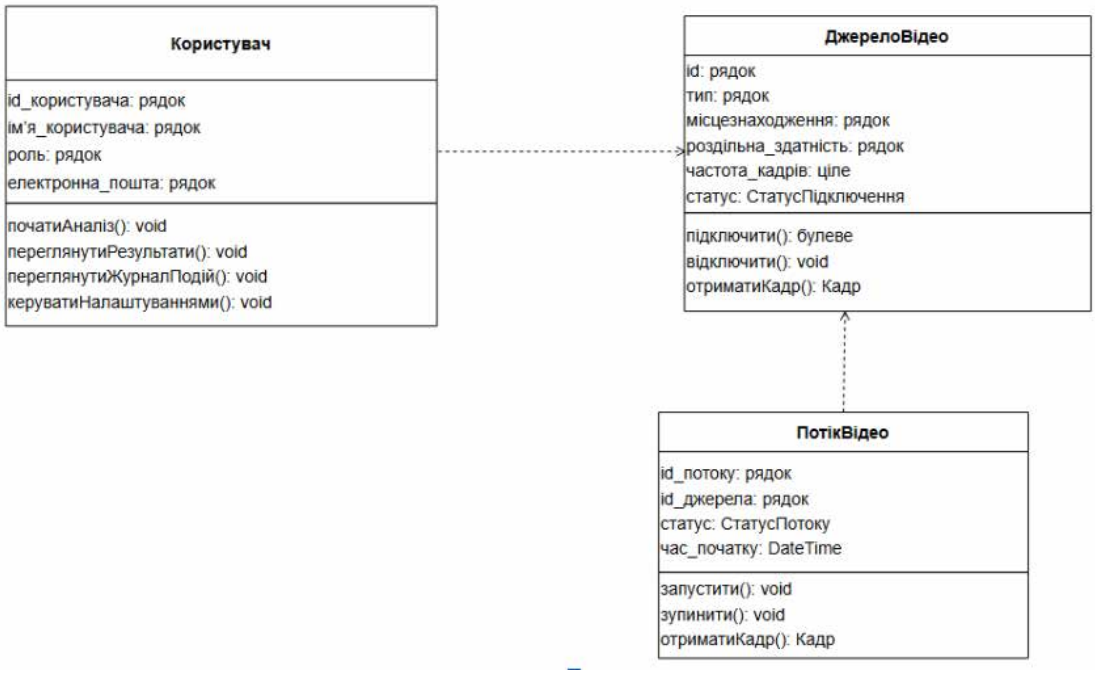

Рис. 2.3. Діаграма кооперації “Отримання відеоданих”

Друга кооперація описує процес аналізу відеопотоку. У цьому сценарії беруть участь ПотікВідео, АналізаторКадрів та ВиявленийОб’єкт. ПотікВідео отримує кадр із активного джерела та передає його до аналізатора кадрів. Аналізатор виконує обробку кадру, застосовує модель виявлення об’єктів і визначає, чи присутні на зображенні потрібні об’єкти.

Якщо об’єкт виявлено, система створює відповідний об’єкт класу ВиявленийОб’єкт. У ньому зберігається тип знайденого об’єкта, координати

межової рамки, рівень впевненості моделі, час першого виявлення та час останнього оновлення. Після цього інформація про знайдений об’єкт може бути передана до результату аналізу або відображена в інтерфейсі користувача.

Ця кооперація демонструє центральний робочий цикл системи: отримання кадру, його передача на аналіз, виконання розпізнавання та формування даних про знайдений об’єкт. Саме цей процес реалізує головну функцію програмного забезпечення - виявлення та ідентифікацію об’єктів у відеопотоці (рис. 2.4).

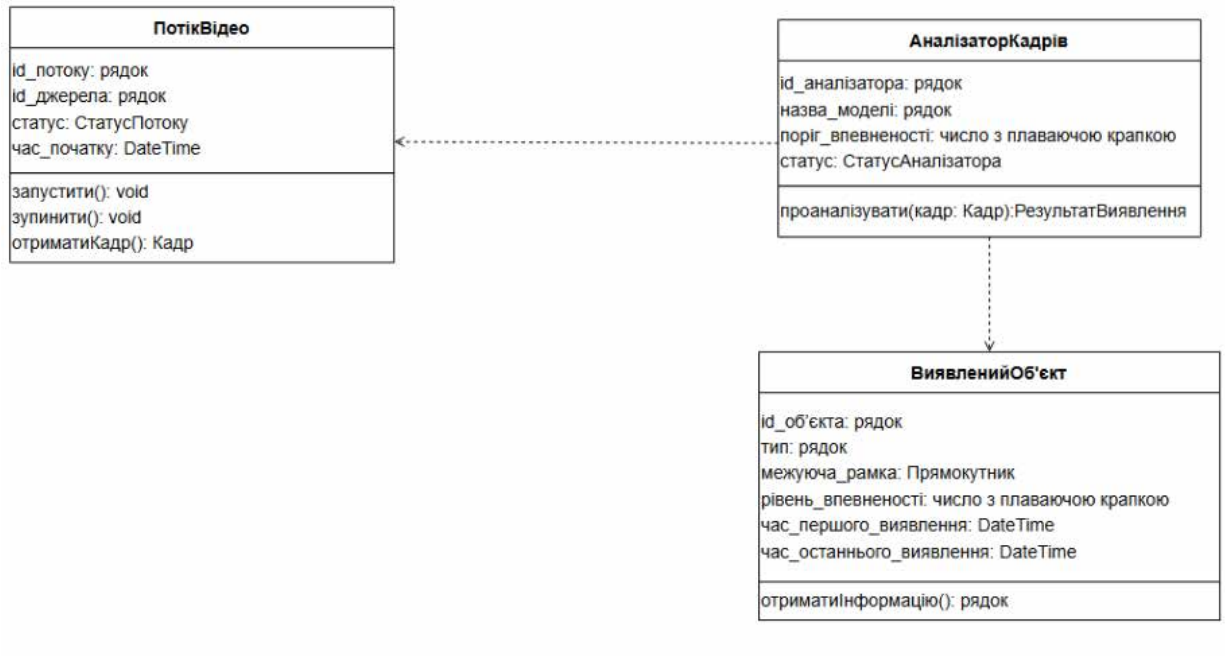

Рис. 2.4. Діаграма кооперації “Аналіз відеопотоку”

Третя кооперація описує процес формування результатів і журналу. У цьому сценарії беруть участь Користувач, РезультатАналізу, ВиявленийОб’єкт та ЖурналПодій. Після аналізу відеопотоку система формує результат, у якому зберігається інформація про потік, час аналізу та кількість виявлених об’єктів. До результату додаються об’єкти, сформовані під час аналізу кадрів.

Після створення результату система може зберегти його для подальшого перегляду. Одночасно важливі дії та стани системи записуються до журналу подій. У журналі можуть фіксуватися запуск аналізу, виявлення нового об’єкта, завершення обробки, помилка підключення до джерела відео або помилка збереження результатів.

Користувач має можливість переглядати як результати аналізу, так і журнал подій. Це дозволяє контролювати роботу системи, перевіряти історію виявлень і аналізувати можливі помилки. Таким чином, дана кооперація показує перехід від технічної обробки кадрів до представлення результатів користувачу та збереження службової інформації про роботу системи (рис. 2.5).

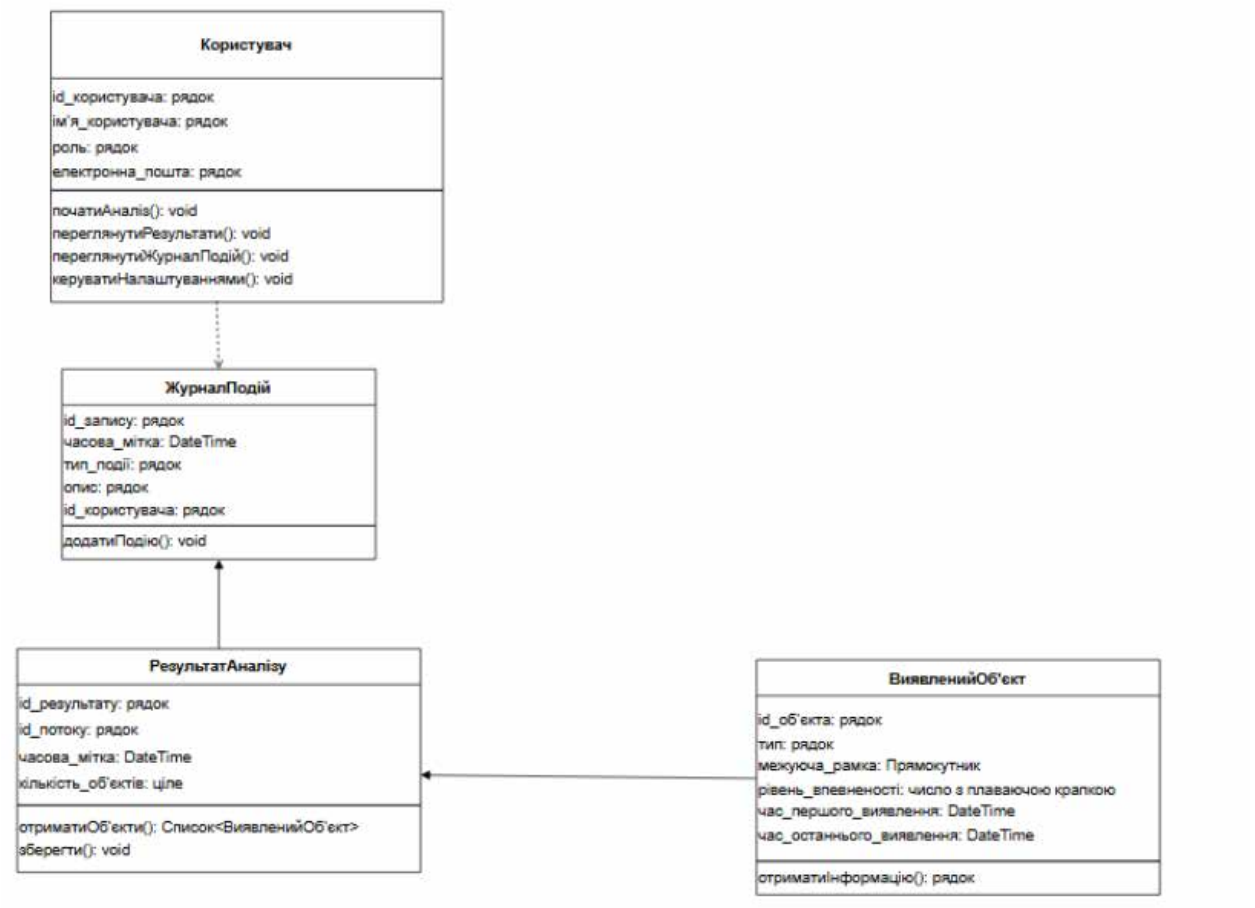

Рис. 2.5. Діаграма кооперації “Формування результатів і журналу”

Загалом діаграми кооперації доповнюють діаграму класів, оскільки показують не лише наявність класів і зв’язків між ними, а й динаміку їхньої взаємодії. Якщо діаграма класів відповідає на питання, з яких об’єктів складається система, то діаграми кооперації пояснюють, як ці об’єкти спільно виконують конкретні функції.

Таким чином, діаграма класів і діаграми кооперації разом дають повне уявлення про об’єктну модель системи. Діаграма класів описує основні сутності програмного забезпечення, їхні атрибути, методи та зв’язки, а діаграми

кооперації деталізують сценарії взаємодії між цими сутностями під час отримання відеоданих, аналізу відеопотоку, формування результатів і ведення журналу подій.

2.3 Діаграма пакетів

Для відображення логічної структури програмного забезпечення системи виявлення та ідентифікації об’єктів у відеопотоці було розроблено UML-діаграму пакетів. Вона дозволяє показати, з яких основних програмних модулів складається система, які частини відповідають за окремі функції та які зовнішні бібліотеки використовуються під час роботи застосунку.

Основним контейнером системи є пакет Diplom Application, який об’єднує всі внутрішні модулі програмного забезпечення. Центральним елементом цього пакета є файл main.py, що виконує роль точки входу до застосунку. Через нього відбувається запуск програми, ініціалізація користувацького інтерфейсу, підключення необхідних модулів та передача керування основним функціональним частинам системи.

Структура застосунку поділена на кілька основних пакетів: gui**,** detection**,** persistence**,** camera**,** console mode та shared**.** Такий поділ дозволяє розмежувати відповідальність між окремими частинами програми та зробити архітектуру більш зрозумілою для подальшої підтримки й розвитку.

Пакет gui відповідає за графічний інтерфейс користувача. У ньому розміщуються елементи, які забезпечують взаємодію користувача із системою: головне вікно застосунку, стилі, віджети та засоби перегляду даних із бази. Саме через цей пакет користувач обирає джерело відеопотоку, запускає або зупиняє аналіз, переглядає відео з виділеними об’єктами та відкриває результати виявлень.

Пакет detection містить основну логіку комп’ютерного зору. До нього належать модулі, що відповідають за використання моделі YOLOv8, відстеження об’єктів та анотацію кадрів. Цей пакет виконує ключову функцію системи -

аналіз відеокадрів, виявлення об’єктів, визначення їх координат і підготовку результатів для подальшого відображення або збереження.

Пакет persistence відповідає за роботу зі збереженням даних. У його складі передбачені модулі init_db та save_detection. Модуль init_db використовується для створення або ініціалізації бази даних, а save_detection - для збереження інформації про виявлені об’єкти. У базі даних фіксуються назва класу об’єкта, час виявлення та шлях до збереженого анотованого зображення.

Пакет camera забезпечує взаємодію з джерелами відеоданих. Він відповідає за пошук доступних камер, підключення до вибраного джерела та отримання кадрів для подальшої обробки. Завдяки цьому система може працювати з камерою комп’ютера, USB-камерою або іншими підтримуваними джерелами відеопотоку.

Пакет console mode може використовуватися для запуску окремих функцій системи без графічного інтерфейсу. Це корисно під час тестування, налагодження або перевірки роботи моделі виявлення. Пакет shared містить спільні службові елементи, зокрема константи та допоміжні функції, які використовуються різними модулями програми.

Окремо на діаграмі пакетів показано зовнішні бібліотеки та ресурси, з якими взаємодіє система. До них належать Ultralytics YOLO, OpenCV, SQLite, Pillow та Tkinter. Бібліотека Ultralytics YOLO використовується для безпосереднього виявлення об’єктів, OpenCV - для роботи з відеопотоком і кадрами, SQLite - для локального збереження метаданих, Pillow - для обробки зображень, а Tkinter - для створення графічного інтерфейсу.

Також система взаємодіє з апаратними та файловими ресурсами: камерою, локальною базою даних detections.db і директорією detections/images, у якій зберігаються анотовані зображення.

Для представлення архітектурної організації програмного забезпечення та логічного групування взаємопов’язаних компонентів було розроблено діаграму пакетів. Дана UML-діаграма дозволяє відобразити структуру системи у вигляді окремих пакетів, що містять класи, модулі та підсистеми, а також демонструє

взаємозв’язки між ними. Використання пакетної структури сприяє підвищенню модульності, спрощує супровід програмного забезпечення та забезпечує більш ефективну організацію процесу розробки. Загальну структуру пакетів програмної системи наведено на рис. 2.6.

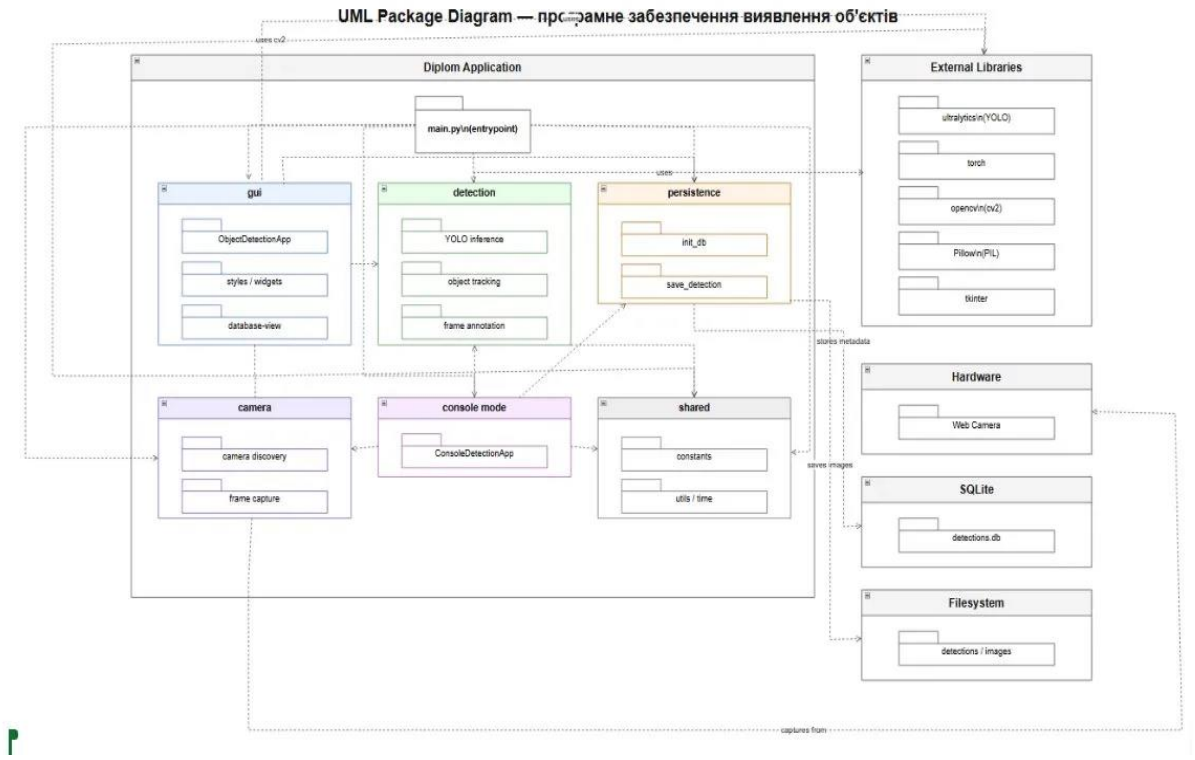

Рис. 2.6. Діаграма пакетів

Як показано на рис. 2.6, програмна система складається з кількох функціонально відокремлених пакетів, кожен з яких відповідає за реалізацію окремої підсистеми або групи функцій. Такий підхід забезпечує чіткий розподіл відповідальності між модулями, спрощує інтеграцію компонентів та підвищує масштабованість програмного продукту. Крім того, використання пакетної структури дозволяє зменшити залежність між окремими елементами системи та забезпечує зручність подальшого супроводу і модернізації програмного забезпечення.

Таким чином, діаграма пакетів демонструє логічну організацію програмного коду, основні модулі системи та залежності між внутрішніми й зовнішніми компонентами.

2.4 Діаграма компонентів

Для опису функціональної архітектури системи було розроблено UML-діаграму компонентів. На відміну від діаграми пакетів, яка показує логічну організацію коду, діаграма компонентів відображає взаємодію основних програмних блоків під час виконання системи. Вона дозволяє зрозуміти, як окремі частини програмного забезпечення обмінюються даними та які функції виконують у загальному процесі роботи.

Архітектуру системи можна умовно поділити на три основні підсистеми: Frontend, Backend та Data & Storage. Такий поділ є зручним, оскільки він відокремлює користувацький інтерфейс від логіки обробки відеопотоку та механізмів збереження результатів.

Підсистема Frontend відповідає за взаємодію користувача із програмним забезпеченням. До неї належать компоненти головного вікна, панелі вибору джерела відео, вікна перегляду відеопотоку, вікна результатів аналізу та вікна журналу подій. Через ці компоненти користувач керує роботою системи: обирає джерело відеоданих, запускає або зупиняє аналіз, переглядає результати виявлення об’єктів та отримує повідомлення про стан роботи програми.

Підсистема Backend реалізує основну логіку функціонування системи. Вона містить компоненти, які відповідають за керування відеопотоком, отримання кадрів, аналіз зображень, роботу з моделлю YOLOv8, обробку результатів та передачу даних до інших частин системи. Саме ця підсистема виконує основний процес: отримує кадр із джерела відео, передає його до модуля аналізу, запускає модель виявлення об’єктів і формує результат.

Компонент керування відеопотоком забезпечує підключення до вибраного джерела відеоданих і контроль процесу отримання кадрів. Компонент отримання кадрів передає окремі зображення до модуля аналізу. Після цього кадр надходить до компонента моделі YOLOv8, який виконує розпізнавання об’єктів. Результатом роботи цього компонента є інформація про знайдені об’єкти, їх класи, координати обмежувальних рамок і рівень упевненості моделі.

Після виявлення об’єктів результати передаються до компонента обробки результатів. Він формує структуровані дані, які можуть бути використані для відображення в інтерфейсі, збереження в базі даних або запису до журналу подій. Якщо система повинна зафіксувати результат, відповідні дані передаються до підсистеми зберігання.

Підсистема Data & Storage відповідає за збереження результатів роботи системи. Вона включає локальну базу даних SQLite, файлове сховище збережених кадрів, журнал подій та репозиторій результатів аналізу. У базі даних зберігаються структуровані метадані про виявлення, зокрема назва класу об’єкта, час виявлення та шлях до збереженого зображення. Файлова система використовується для збереження анотованих кадрів, на яких відображено знайдені об’єкти.

Журнал подій використовується для фіксації важливих дій системи. До нього можуть записуватися повідомлення про запуск аналізу, успішне виявлення об’єкта, помилки підключення до камери, проблеми з обробкою кадру або помилки збереження результатів. Це дає змогу контролювати роботу системи та спрощує пошук причин можливих збоїв.

Користувач взаємодіє із системою через інтерфейсні компоненти. Після вибору джерела відео та запуску аналізу відповідний запит передається до backend-компонентів. Вони виконують обробку відеопотоку, передають кадри до моделі YOLOv8 і отримують результати розпізнавання. Після цього результати повертаються до frontend-частини для візуального відображення у вигляді відеопотоку з рамками навколо об’єктів, списку результатів або повідомлень у журналі.

Діаграма компонентів демонструє, що система побудована за принципом розділення відповідальності. Інтерфейсна частина відповідає за керування та відображення даних, backend - за обробку відеопотоку та виконання виявлення об’єктів, а підсистема зберігання - за фіксацію результатів і подій. Така архітектура робить систему більш зрозумілою, зручною для тестування та придатною до подальшого розширення.

Для деталізації фізичної структури програмного забезпечення та відображення взаємодії між окремими програмними модулями було побудовано діаграму компонентів системи. Дана UML-діаграма демонструє основні компоненти програмного застосунку, їх функціональне призначення, а також інтерфейси та залежності між ними. Використання діаграми компонентів дозволяє сформувати цілісне уявлення про архітектуру програмної системи, визначити способи інтеграції модулів та забезпечити ефективну організацію процесу розробки і супроводу програмного забезпечення. Структуру компонентів системи наведено на рис. 2.7.

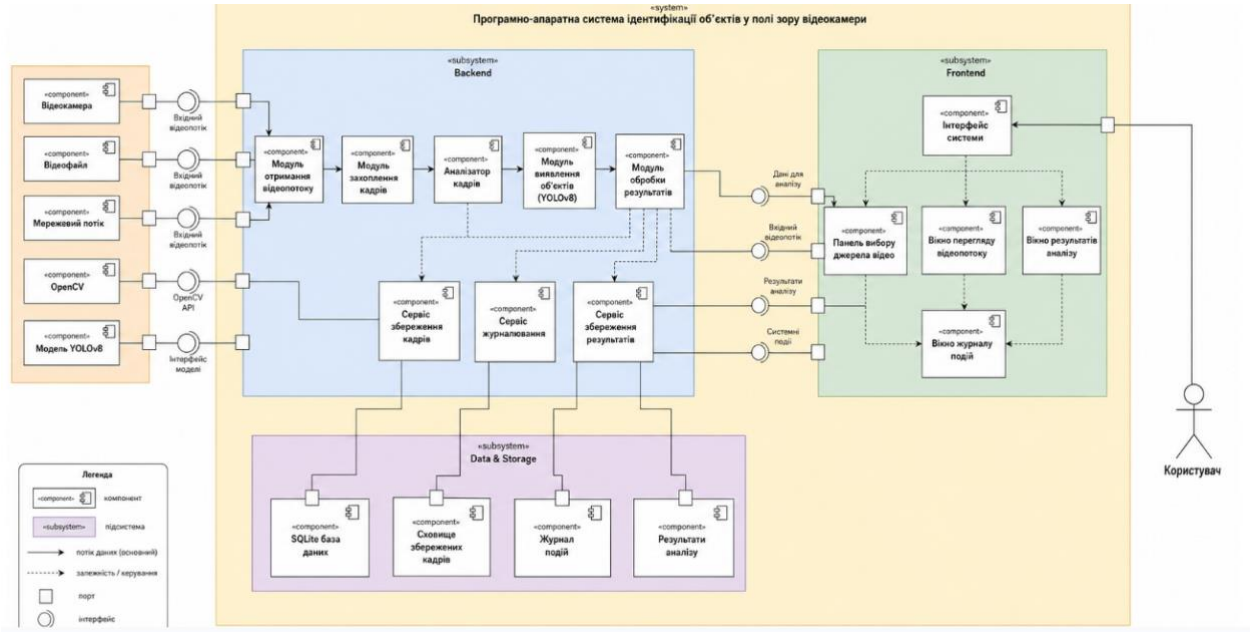

Рис. 2.7. Діаграма компонентів системи

Як показано на рис. 2.7, програмна система складається з набору взаємопов'язаних компонентів, кожен з яких реалізує окрему функціональну частину застосунку. Взаємодія між компонентами здійснюється через визначені інтерфейси, що забезпечує модульність, гнучкість та можливість незалежного оновлення окремих елементів системи. Така архітектурна організація сприяє підвищенню надійності програмного забезпечення, спрощує тестування та забезпечує можливість подальшого масштабування й модернізації програмного продукту.

2.5 Діаграма розгортання

Діаграма розгортання відображає фізичну структуру виконання системи та розподіл компонентів між вузлами [4].

Система розгорнута на єдиному вузлі - MacBook Pro 2023 з чіпом Apple M2. Апаратна складова включає 8-ядерний процесор (4 продуктивних і 4 ефективних ядра), 10-ядерний GPU, 16-ядерний Neural Engine та 16 ГБ оперативної пам’яті. Вбудована камера FaceTime HD (1080p, 30 кадрів/с) є єдиним зовнішнім джерелом відеопотоку. Програмну складову формують: середовище виконання macOS 13.0+, ізольоване Python-середовище (venv) з усіма залежностями, файл застосунку main.py, нейромережева модель yolov8n.pt, база даних detections.db та директорія detections/ для анотованих зображень.

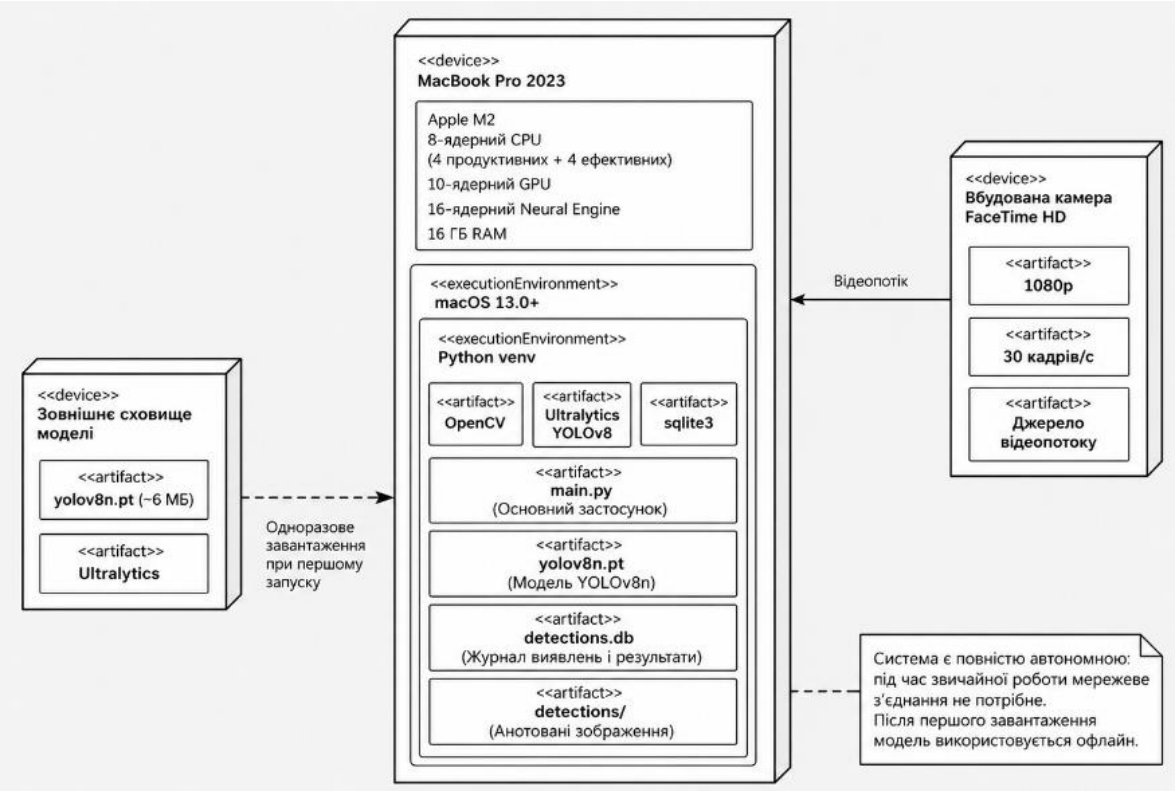

Рис. 2.8. Діаграма розгортання системи

Система є повністю автономною: жодного мережевого з’єднання під час роботи не потрібно. Єдиний зовнішній запит - автоматичне завантаження yolov8n.pt (~6 МБ) при першому запуску, після чого застосунок функціонує офлайн.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Реалізація системи управління даними

Система управління даними є важливою складовою програмного застосунку та забезпечує збереження, обробку й доступ до інформації про виявлені об’єкти протягом усього циклу роботи системи. Основними вимогами до підсистеми управління даними є забезпечення цілісності інформації, швидкого доступу до записів, можливості пошуку та сортування даних, а також надійного зберігання графічних матеріалів, отриманих у процесі роботи застосунку.

Для реалізації системи зберігання інформації було використано комбінований підхід, який передбачає використання реляційної бази даних для збереження структурованих метаданих та файлової системи для збереження зображень. Такий підхід дозволяє оптимізувати використання ресурсів системи, спростити організацію доступу до даних та забезпечити ефективну роботу із файлами великого обсягу.

Як реляційну СУБД обрано SQLite - вбудовану безсерверну базу даних, що зберігає всі дані в єдиному файлі. Вибір SQLite зумовлений кількома чинниками. По-перше, це стандартний модуль Python (sqlite3), що не потребує встановлення додаткових залежностей. По-друге, SQLite є повноцінною RDBMS із підтримкою SQL-запитів, транзакцій та первинних ключів, що забезпечує необхідний рівень структурованості для задач пошуку та сортування. По-третє, файловий характер бази даних робить її портативною - для резервного копіювання достатньо скопіювати один файл [10].

Ініціалізацію бази даних виконує функція init_db(), яка викликається при кожному запуску застосунку. Функція підключається до файлу detections.db (або створює його, якщо він відсутній), перевіряє наявність таблиці detections і, за потреби, створює її. Використання конструкції CREATE TABLE IF NOT EXISTS

гарантує ідемпотентність операції - повторний запуск не призводить до помилки і не впливає на існуючі дані.

Код функції init_db() наведено на рис. 3.1.

```
# -------------------- Робота з БД --------------------
def init_db():
    conn = sqlite3.connect(DB_NAME)
    c = conn.cursor()
    c.execute('''
        CREATE TABLE IF NOT EXISTS detections (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            class_name TEXT,
            timestamp TEXT,
            image_path TEXT
        )
    ''')
    conn.commit()
    conn.close()
```

Рис. 3.1. Код функції ініціалізації бази даних init_db()

Після виклику init_db() функція commit() фіксує транзакцію, а close() закриває з'єднання. Такий підхід «відкрити - виконати - закрити» застосовується в усіх функціях роботи з БД у системі, що унеможливлює витік з'єднань і забезпечує потокобезпечність операцій.

Паралельно з базою даних функціонує файлова система для зберігання зображень. Директорія detections/ створюється автоматично засобами модуля os при першому збереженні зображення. Кожен файл отримує унікальну назву за шаблоном {class_name}_{timestamp}.jpg, де timestamp формується функцією datetime.now().strftime('%Y-%m-%d_%H-%M-%S'). Такий формат дозволяє одночасно забезпечити унікальність назв і зберегти їх хронологічне впорядкування у файловому менеджері.

Запропонована структура системи управління даними забезпечує ефективне поєднання структурованого зберігання інформації та роботи з графічними файлами, що підвищує надійність, масштабованість і зручність подальшого супроводу програмного забезпечення.

3.2 Розробка інформаційної бази

Розробка інформаційної бази є одним із ключових етапів створення програмної системи, оскільки саме вона забезпечує організацію структури зберігання, обробки та доступу до даних. На даному етапі було сформовано структуру таблиць бази даних, визначено основні поля, типи даних, а також зв’язки між елементами інформаційної системи. Проєктування інформаційної бази виконувалося з урахуванням вимог до швидкодії, цілісності та масштабованості системи, що дозволяє забезпечити ефективну роботу застосунку при збереженні та обробці інформації про результати детектування об’єктів.

Інформаційна база системи реалізована у вигляді єдиної таблиці detections у файлі SQLite detections.db і тісно пов'язаної з нею файлової структури директорії detections/. Основна таблиця бази даних призначена для збереження метаданих про виявлені об’єкти, включаючи їх назву, час фіксації, координати або додаткові характеристики, а також шлях до відповідного графічного файлу. Структуру інформаційної бази та характеристику її основних полів наведено у табл. 3.1.

Таблиця 3.1

Структура таблиці detections

Поле	Тип даних	Обмеження	Опис
id	INTEGER	PRIMARY KEY AUTOINCREMENT	Унікальний ідентифікатор запису. Автоматично збільшується при додаванні нового рядка.
class_name	TEXT	NOT NULL (неявно)	Назва класу виявленого об'єкта відповідно до датасету COCO (наприклад: «person», «car», «laptop», «cup»).
timestamp	TEXT	NOT NULL (неявно)	Дата і час ідентифікації у форматі «РРРР-ММ-ДД_ГГ-ХХ-СС». Використовується для сортування у вікні перегляду БД.
image_path	TEXT	NOT NULL (неявно)	Шлях до анотованого JPG-файлу у директорії detections/. Використовується для відкриття зображення при подвійному кліку у таблиці.

Додавання нового запису до інформаційної бази здійснюється функцією save_detection(), яка реалізує операцію збереження даних за допомогою параметризованого SQL-запиту INSERT. Використання параметрів у запитах замість прямої конкатенації рядків є важливою практикою безпечної роботи з базами даних, оскільки дозволяє уникнути SQL-ін’єкцій та забезпечує коректну обробку текстових і службових символів у вхідних даних [10]. Крім підвищення рівня безпеки, параметризовані запити спрощують підтримку програмного коду та забезпечують кращу сумісність із механізмами роботи бібліотеки sqlite3.

Для отримання та відображення інформації у вікні перегляду результатів використовується SQL-запит SELECT, який виконує вибірку записів із таблиці бази даних. Сортування результатів здійснюється за полем timestamp у спадному порядку із застосуванням конструкції ORDER BY timestamp DESC, що забезпечує відображення найновіших результатів детектування на початку списку. Такий підхід підвищує зручність роботи користувача із системою, оскільки дозволяє оперативно переглядати останні ідентифіковані об’єкти без необхідності додаткового пошуку.

Механізм фільтрації даних реалізовано на рівні програмної логіки Python шляхом аналізу відповідності вмісту рядків введеному користувачем пошуковому запиту. Даний підхід забезпечує достатню гнучкість обробки даних та дозволяє швидко виконувати локальний пошук без перевантаження системи складними SQL-конструкціями. У подальшому така архітектура може бути розширена підтримкою багатокритеріального пошуку та фільтрації за окремими атрибутами об’єктів.

Для зберігання графічних результатів роботи системи використовується окрема директорія detections/, у якій кожен файл отримує унікальне ім’я відповідно до шаблону class_name_timestamp.jpg. Прикладами структури назв файлів є: person_2026-04-15_14-23-07.jpg, laptop_2026-04-15_14-23-15.jpg, cup_2026-04-15_14-24-02.jpg тощо. Такий формат іменування дозволяє не лише уникнути дублювання назв файлів, але й забезпечує їх природне хронологічне впорядкування у файловій системі.

Кожне збережене зображення являє собою повноцінний RGB-файл, сформований у процесі роботи алгоритму комп’ютерного зору. На зображенні автоматично відображається зелена обмежувальна рамка навколо виявленого об’єкта, а також текстовий підпис класу у верхній частині рамки. Візуалізація результатів детектування дозволяє користувачеві швидко оцінити коректність роботи алгоритму розпізнавання та забезпечує зручність подальшого аналізу отриманих результатів.

3.3 Алгоритм функціонування системи

Алгоритм функціонування системи виявлення та ідентифікації об’єктів у відеопотоці починається із запуску програмного забезпечення користувачем. Після відкриття системи відбувається ініціалізація основних модулів, перевірка готовності програмного середовища та підготовка інтерфейсу користувача до роботи.

На першому етапі користувач обирає джерело відеоданих, з якого буде виконуватися подальший аналіз. Система може працювати з різними типами джерел, зокрема з камерою комп’ютера, USB-камерою, відеофайлом або мережевим потоком. Після вибору джерела система перевіряє його доступність. Якщо джерело недоступне, користувачу виводиться повідомлення про помилку або пропозиція повторно вибрати інше джерело відеопотоку.

Якщо джерело доступне, система встановлює з ним з’єднання та починає отримувати кадри відеопотоку. Отримані кадри передаються до модуля аналізу, де виконується їх попередня обробка. На цьому етапі кадр приводиться до формату, придатного для подальшої обробки моделлю комп’ютерного зору.

Далі кадр передається до модуля виявлення об’єктів на основі моделі YOLOv8. Модель аналізує зображення та визначає наявність об’єктів у кадрі. Якщо об’єкти не виявлено, система продовжує отримання та аналіз наступних кадрів. Якщо об’єкти виявлено, система формує результати розпізнавання:

визначає клас об’єкта, координати обмежувальної рамки, рівень впевненості моделі та часову мітку виявлення.

Після формування результатів система відображає їх у користувацькому інтерфейсі. На відеопотоці виводяться обмежувальні рамки навколо знайдених об’єктів, а також текстова інформація про клас об’єкта та інші параметри виявлення. Це дозволяє користувачу в реальному часі спостерігати за процесом роботи системи.

Далі система перевіряє, чи потрібно зберігати результат виявлення. Якщо збереження не потрібне, обробка продовжується для наступного кадру. Якщо результат потрібно зафіксувати, система зберігає анотоване зображення у файлову систему, а структуровані метадані виявлення записує до бази даних. До таких даних належать ідентифікатор запису, назва класу об’єкта, час виявлення та шлях до збереженого зображення.

Після цього система записує подію до журналу роботи. У журналі можуть фіксуватися як успішні дії системи, так і можливі помилки: недоступність камери, помилки підключення, проблеми з обробкою кадру або збереженням результатів. Це дозволяє надалі переглядати історію роботи системи та аналізувати її стан.

Алгоритм продовжує виконуватися циклічно: система отримує новий кадр, передає його на аналіз, виконує виявлення об’єктів, формує результат, відображає його в інтерфейсі та за потреби зберігає дані. Робота системи триває доти, доки користувач не завершить аналіз або не закриє програму.

Після завершення роботи система припиняє отримання відеопотоку, звільняє ресурси камери або відеофайлу, завершує активні процеси обробки та коректно закриває програмне забезпечення. Таким чином забезпечується повний цикл функціонування системи - від вибору джерела відео до аналізу, виявлення об’єктів, збереження результатів і завершення роботи.

Для наочного представлення послідовності роботи програмного застосунку було розроблено схему алгоритму функціонування системи виявлення об’єктів. Дана схема відображає основні етапи обробки відеопотоку

або зображення, починаючи від отримання вхідних даних і завершуючи збереженням результатів детектування у базі даних та файловій системі. Алгоритм включає етапи попередньої обробки даних, запуску моделі розпізнавання, аналізу результатів детектування, візуалізації знайдених об’єктів та формування записів у системі зберігання даних. Схему алгоритму функціонування системи наведено на рис. 3.2.

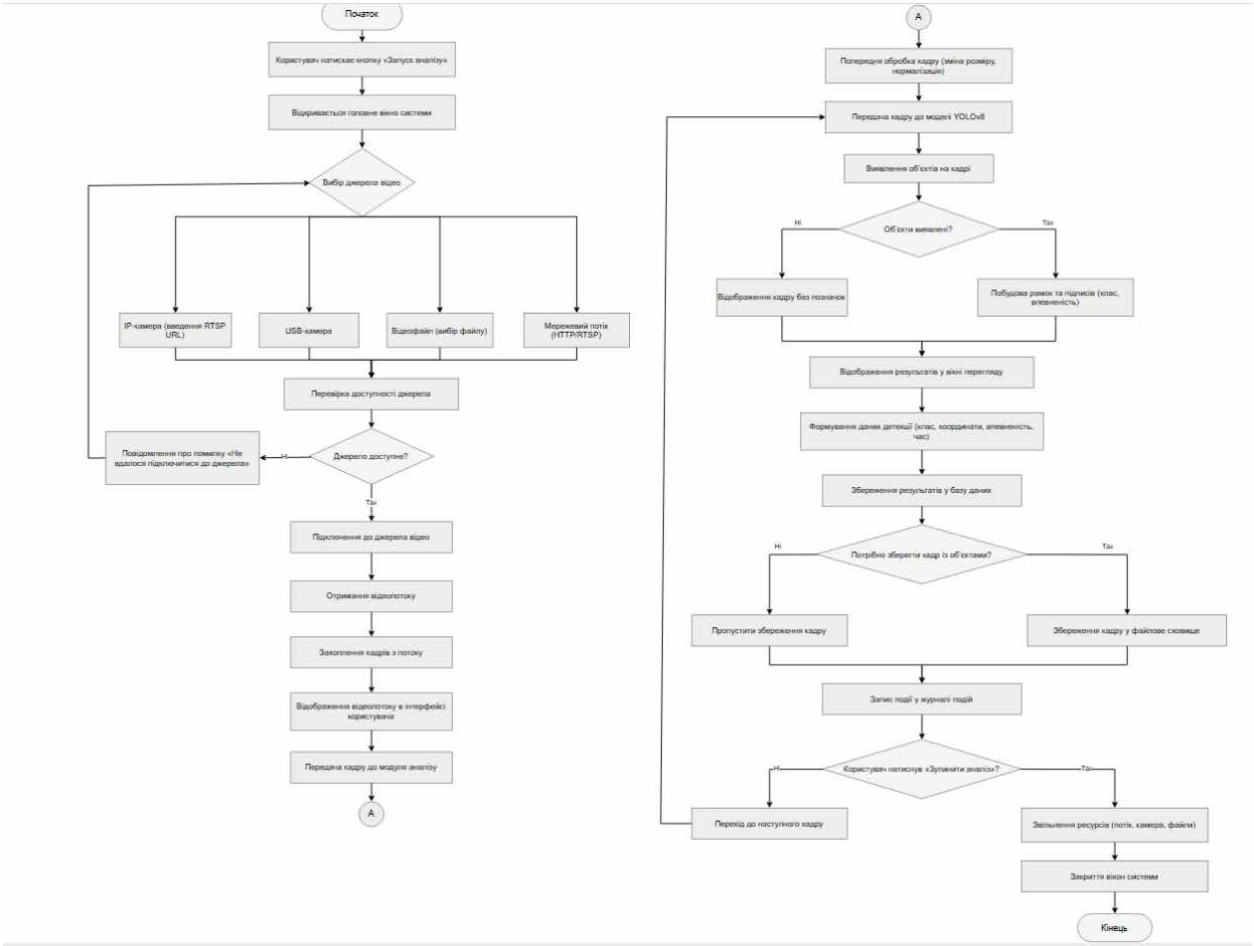

Рис. 3.2. Схема алгоритму функціонування системи виявлення об'єктів

Як показано на рис. 3.2, робота системи організована у вигляді послідовного циклу обробки даних, під час якого здійснюється отримання зображення, аналіз кадру засобами моделі комп’ютерного зору та формування результатів детектування. Після виявлення об’єктів система виконує побудову графічних позначень на зображенні, збереження інформації у базі даних та запис відповідних графічних файлів у файлову систему. Така організація алгоритму забезпечує автоматизацію процесу виявлення об’єктів, підвищує швидкодію

застосунку та дозволяє ефективно накопичувати результати роботи системи для подальшого аналізу й перегляду користувачем.

3.4 Реалізація модуля виявлення та класифікації об'єктів

Модуль виявлення та класифікації об'єктів є ключовим компонентом системи і реалізований на базі нейронної мережі YOLOv8 від компанії Ultralytics. YOLOv8 - це остання версія сімейства YOLO станом на 2023 рік, яка реалізує anchor-free підхід до виявлення об'єктів і надає збалансований набір моделей різних розмірів [3].

Вибір моделі YOLOv8n (nano) для даної системи зумовлений цільовою апаратною платформою: MacBook Pro M2 не має Nvidia GPU, а отже CUDA-прискорення недоступне. Модель nano є найлегшою у сімействі YOLOv8 і показує прийнятну продуктивність на CPU: близько 15–20 FPS при роздільній здатності 640×480 пікселів, що відповідає вимозі реального часу. Порівняння варіантів моделей наведено в табл. 3.2.

Таблиця 3.2

Порівняння варіантів моделей YOLOv8

Модель	**Параметри, млн**	**mAP@0.5**	**Швидкість (CPU)**	**Застосування**
YOLOv8n	3.2	37.3%	15–20 FPS	Десктоп, CPU-пристрої
YOLOv8s	11.2	44.9%	8–12 FPS	Збалансована точність
YOLOv8m	25.9	50.2%	4–6 FPS	Середня точність / GPU
YOLOv8l	43.7	52.9%	2–3 FPS	GPU з CUDA
YOLOv8x	68.2	53.9%	1–2 FPS	Максимальна точність / GPU

Модель YOLOv8 на стандартному датасеті COCO здатна розпізнавати 80 класів об'єктів, що поділяються на кілька категорій: люди (person); транспортні засоби (car, motorcycle, bus, bicycle, truck тощо); тварини (cat, dog, bird, horse); побутові предмети (laptop, phone, cup, bottle, chair, keyboard, mouse тощо); продукти харчування (apple, banana, pizza); спортивний інвентар та інші [3].

Центральним методом детектування є detect() класу ObjectDetectionApp. Виклик self.model(frame, verbose=False)[0] передає кадр у модель і повертає об'єкт Results. Атрибут results.boxes містить колекцію об'єктів Box, кожен з яких має: хуху - тензор із координатами кутів bounding box (x1, y1, x2, y2); conf - рівень впевненості моделі; cls - числовий ідентифікатор класу. Ім'я класу отримується через словник self.model.names[cls_id].

Алгоритм відстеження об’єктів між кадрами реалізовано за допомогою списку detected_objects, елементи якого містять назву класу об’єкта, координати його центру, час останнього виявлення та прапор збереження інформації. Під час обробки нового кадру система порівнює поточне виявлення з раніше зареєстрованими об’єктами того самого класу шляхом обчислення евклідової відстані між їх центрами. Якщо відстань не перевищує значення DISTANCE_THRESHOLD, а часовий інтервал є меншим за COORDINATE_TIMEOUT, об’єкти вважаються ідентичними. Такий підхід дозволяє уникнути багаторазової реєстрації одного й того самого об’єкта під час його безперервного перебування у полі зору камери та забезпечує стабільність роботи системи моніторингу. Принцип роботи алгоритму відстеження об’єктів між кадрами наведено на рис. 3.3.

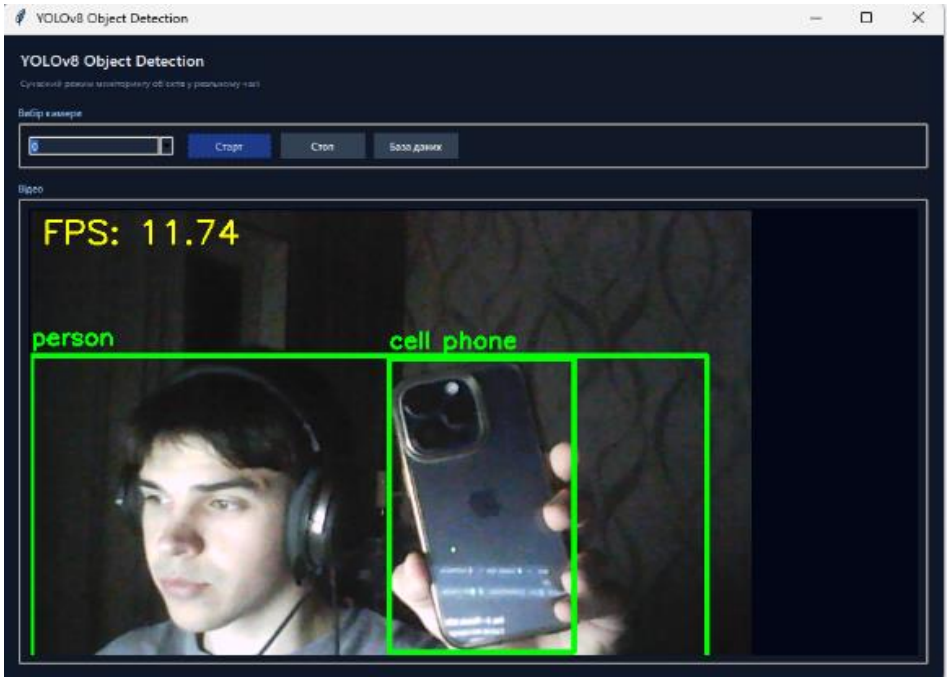

Рис. 3.3. Принцип роботи алгоритму відстеження об'єктів між кадрами

Після завершення інтервалу COORDINATE_TIMEOUT значення прапора saved автоматично змінюється на False. У результаті об’єкт, який повторно з’явиться у кадрі після певної паузи, буде оброблений як нове виявлення та повторно зареєстрований у системі. Така логіка є доцільною для систем моніторингу та відеоспостереження, оскільки дозволяє фіксувати повторні появи об’єктів у різний час.

Для реалізації допоміжних операцій обробки відеопотоку використовується бібліотека OpenCV. Відкриття та отримання кадрів із камери здійснюється за допомогою функції cv2.VideoCapture(). Побудова графічних позначень на зображенні реалізується функціями cv2.rectangle() та cv2.putText(), які використовуються для відображення обмежувальних рамок і текстових підписів об’єктів. Перед передачею зображення до графічного інтерфейсу Tkinter виконується конвертація кольорового простору з формату BGR у RGB засобами функції cv2.cvtColor(), що забезпечує коректне відображення кольорів у вікні застосунку.

3.5 Реалізація графічного інтерфейсу користувача

Графічний інтерфейс реалізований засобами стандартної Python-бібліотеки tkinter з розширеним набором тематизованих віджетів ttk. Обрання tkinter зумовлено його наявністю у стандартній поставці Python і нативною підтримкою на macOS без встановлення додаткових залежностей.

Метод configure_styles() встановлює темну кольорову схему через механізм ttk.Style. Використовується базова тема «clam» як найгнучкіша з точки зору кастомізації. Основна палітра: фон вікна #0f172a (темно-синій); фон карток #111827 (темно-сірий); основний колір кнопок #2563eb (синій); текст #e2e8f0 (світло-сірий); акцентні підписи заголовків карток #93c5fd (блакитний). Така палітра відповідає сучасним дизайн-системам темних інтерфейсів і забезпечує достатній контраст для тривалої роботи.

Метод create_widgets() будує ієрархію компонентів головного вікна. Вікно розміром 980×700 пікселів містить чотири секції.

Секція заголовку включає назву застосунку «YOLOv8 Object Detection» великим шрифтом і підзаголовок з описом призначення. Це забезпечує однозначну ідентифікацію застосунку без зайвих елементів.

Панель вибору камери (ttk.LabelFrame «Вибір камери») містить: комбобокс із списком індексів доступних камер, що автоматично заповнюється функцією get_available_cameras(); кнопку «Старт» (стиль Primary.TButton - синя); кнопку «Стоп» (стиль Secondary.TButton - темно-сіра, спочатку деактивована); кнопку «База даних» (Secondary.TButton).

Область відображення відео (ttk.LabelFrame «Відео») займає більшу частину вікна завдяки параметрам fill="both", expand=True. Всередині розміщено ttk.Label із чорним фоном (#020617), в якому оновлюється зображення кадру через механізм ImageTk.PhotoImage.

Журнал ідентифікацій (ttk.LabelFrame «Журнал ідентифікацій») - компактний текстовий блок висотою 4 рядки (параметр height=4) з прокруткою. Відображає рядки формату «{timestamp}: {class_name} збережено в {image_path}» у шрифті Consolas 9pt на темному фоні. Компактний розмір дозволяє максимально збільшити область відображення відео.

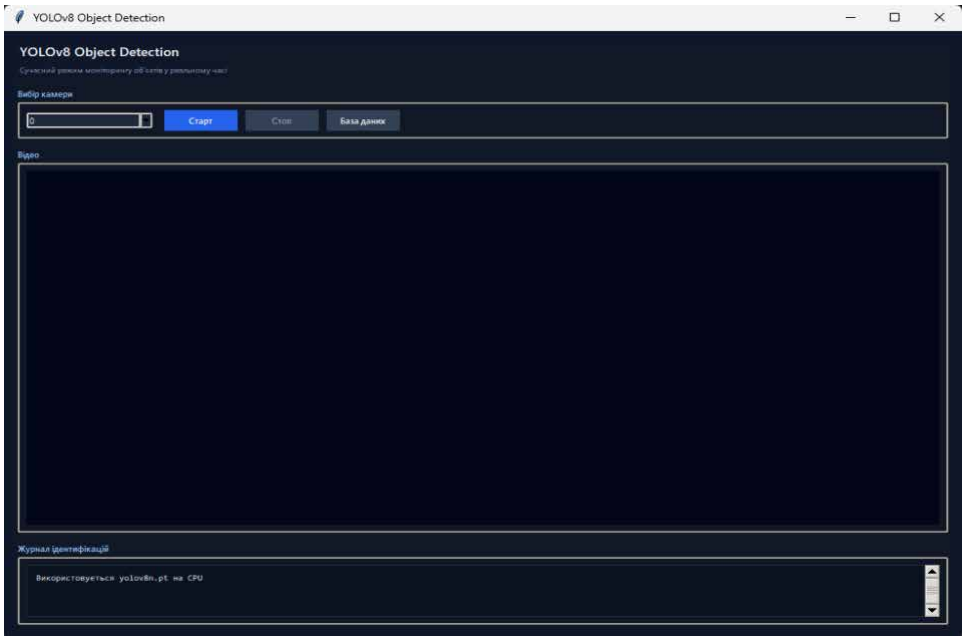

Рис. 3.4. Головне вікно застосунку YOLOv8 Object Detection

Метод show_database() відкриває окреме вікно Toplevel розміром 980×600 пікселів для перегляду результатів. Воно містить поле пошуку (ttk.Entry) з динамічним оновленням таблиці через механізм StringVar.trace_add(), таблицю ttk.Treeview з трьома колонками (Клас, Час, Шлях) і вертикальним скролбаром. Колонки є сортованими: клік на заголовку сортує записи за відповідним полем; повторний клік змінює порядок сортування на зворотній. Для поля timestamp сортування здійснюється парсингом рядка у datetime-об'єкт для коректного хронологічного впорядкування.

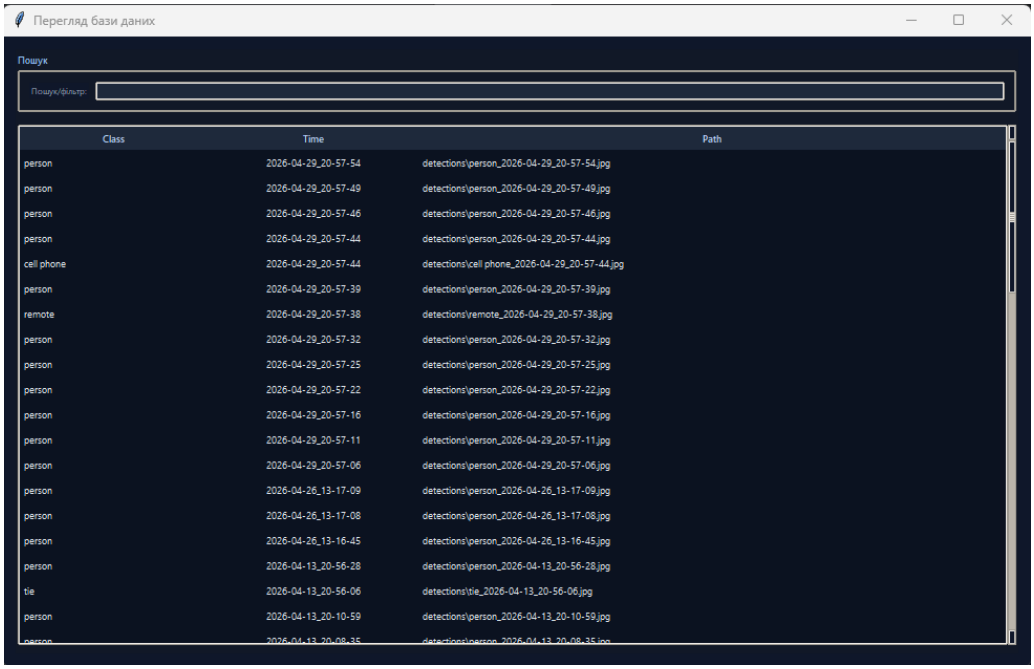

Рис. 3.5. Вікно перегляду бази даних ідентифікацій

Оновлення відеофрейму в реальному часі забезпечується таким ланцюжком: cv2.cvtColor(frame, cv2.COLOR_BGR2RGB) конвертує кадр із BGR в RGB; Image.fromarray(frame_rgb) перетворює numpy-масив у Pillow-зображення; ImageTk.PhotoImage(image=img) підготовлює зображення для Tkinter; video_label.config(image=imgtk) встановлює нове зображення у лейбл. Обов'язкове збереження посилання self.video_label.imgtk = imgtk запобігає збору сміття та зникненню зображення.

3.6 Реалізація модуля збереження та перегляду результатів

Модуль збереження та перегляду результатів реалізований через функцію save_detection() і метод show_database() класу ObjectDetectionApp. Ці компоненти забезпечують повний цикл роботи з даними: фіксацію результатів у момент ідентифікації та надання зручного інтерфейсу для їх перегляду.

Функція save_detection(class_name, frame, bbox) виконує такі дії. Отримує координати bounding box у вигляді кортежу (x1, y1, x2, y2). Створює копію поточного кадру за допомогою frame.copy(), щоб не модифікувати оригінальний кадр, що відображається в GUI. Наносить зелений прямокутник навколо об'єкта і текстовий підпис у лівому верхньому куті. Генерує мітку часу і формує унікальну назву файлу. Перевіряє наявність директорії detections/ і за потреби створює її. Зберігає анотований кадр у форматі JPEG через cv2.imwrite(). Відкриває з'єднання з SQLite, виконує INSERT і фіксує транзакцію. Повертає пару (image_path, timestamp) для відображення у журналі.

Код функції save_detection() наведено на рис. 3.6.

```
def save_detection(class_name, frame, bbox):
    x1, y1, x2, y2 = bbox

    annotated = frame.copy()
    cv2.rectangle(annotated, (x1, y1), (x2, y2), (0, 255, 0), 3)
    cv2.putText(annotated, class_name, (x1, y1 - 10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)

    timestamp = datetime.now().strftime('%Y-%m-%d_%H-%M-%S')
    filename = f"{class_name}_{timestamp}.jpg"

    if not os.path.exists(SAVE_DIR):
        os.makedirs(SAVE_DIR)

    image_path = os.path.join(SAVE_DIR, filename)
    cv2.imwrite(image_path, annotated)

    conn = sqlite3.connect(DB_NAME)
    c = conn.cursor()
    c.execute(
        'INSERT INTO detections (class_name, timestamp, image_path) VALUES (?, ?, ?)',
        (class_name, timestamp, image_path)
    )
    conn.commit()
    conn.close()

    return image_path, timestamp
```

Рис. 3.6. Код функції save_detection() - збереження результату ідентифікації

Функція get_available_cameras() виконує автоматичне виявлення підключених відеопристроїв шляхом послідовного перебору індексів, починаючи з 0. Для кожного індексу створюється об'єкт VideoCapture і здійснюється спроба читання кадру. Якщо кадр успішно зчитано, індекс додається до списку доступних камер. Цикл завершується при першому невдалому читанні. На MacBook Pro вбудована камера FaceTime HD отримує індекс 0.

Перегляд результатів у вікні бази даних забезпечується методом show_database(). Дані завантажуються з SQLite через SELECT і відображаються в ttk.Treeview. Пошукова функція реалізована на основі рядкового пошуку підрядка в об'єднаній рядковій репрезентації кожного запису. Відкриття зображення при подвійному кліку виконується через cv2.imshow(), що відкриває нативне вікно перегляду OpenCV із повнорозмірним анотованим зображенням.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Аналіз переваг, обмежень і напрямів розвитку системи

Розроблене програмне забезпечення демонструє ряд суттєвих переваг у порівнянні з розглянутими аналогами. Повністю локальна робота без підключення до мережі Інтернет є принциповою перевагою з точки зору конфіденційності та надійності: система не залежить від стабільності зовнішніх сервісів, не передає відеодані на сторонні сервери і функціонує навіть у середовищах з обмеженим або відсутнім доступом до мережі.

Використання виключно відкритого програмного забезпечення (Python, OpenCV, YOLOv8, SQLite) забезпечує мінімальну вартість впровадження і повну свободу модифікації під специфічні потреби. Проста установка через pip і запуск єдиним файлом робить систему доступною для широкої аудиторії без спеціальних технічних знань. Сучасний темний інтерфейс забезпечує зручну тривалу роботу без стомлення зору. Автоматичне завантаження моделі при першому запуску усуває необхідність ручного налаштування.

Водночас система має низку обмежень, що є природним наслідком вибраних проєктних рішень. Підтримка лише 80 класів об'єктів датасету COCO обмежує застосовність у вузькоспеціалізованих доменах, де потрібне виявлення нестандартних об'єктів. Базовий центроїдний алгоритм відстеження між кадрами є менш точним порівняно з повноцінними системами трекінгу (DeepSORT, ByteTrack), що може призводити до помилкового спрацювання при швидкому переміщенні або перекритті об'єктів.

Відсутність підтримки MPS (Metal Performance Shaders) - API апаратного прискорення Apple Silicon - є технічним обмеженням, оскільки PyTorch починаючи з версії 1.12 підтримує MPS, однак реалізація Ultralytics YOLOv8 потребує явної вказівки device="mps" для його використання. Система підтримує лише одну камеру одночасно і не надає можливості аналізу відеофайлів.

Перспективні напрями подальшого розвитку системи охоплюють такі аспекти. Інтеграція підтримки Apple MPS дозволить використовувати нейронний движок M2 і 10-ядерний GPU для GPU-прискореної інференції YOLOv8, що потенційно збільшить швидкість обробки у 3–5 разів. Додавання підтримки відеофайлів як вхідного джерела розширить застосовність системи для аналізу записаного відео.

Реалізація інтерфейсу для навчання кастомної моделі YOLOv8 на власному датасеті дозволить виявляти специфічні для предметної області об'єкти. Підтримка декількох камер одночасно з агрегацією результатів у єдину базу даних відкриє можливості для побудови комплексних систем відеоспостереження. Функція експорту результатів у форматах CSV та PDF і формування звітів надасть додаткову цінність для аналітичних задач. Реалізація REST API дозволить інтегрувати систему як компонент у більші автоматизовані рішення.

4.2 Тестування системи

Тестування програмного забезпечення є важливим етапом розробки, що дозволяє перевірити коректність функціонування системи, стабільність її роботи та відповідність поставленим вимогам. Основною метою тестування було виявлення можливих помилок, перевірка правильності обробки даних, оцінювання роботи алгоритмів детектування та відстеження об’єктів, а також підтвердження працездатності графічного інтерфейсу й підсистеми збереження інформації. У процесі дослідження проведено комплексне функціональне тестування, яке охоплювало перевірку отримання відеопотоку з камери, роботи алгоритмів розпізнавання об’єктів, механізмів їх відстеження між кадрами, збереження результатів у базі даних і файловій системі, а також відображення інформації у графічному інтерфейсі користувача. Додатково виконувалась перевірка стабільності роботи системи під час тривалої обробки відеопотоку в режимі реального часу.

Тестування проводилось на MacBook Pro 2023 (Apple M2, 16 ГБ RAM) під управлінням macOS Ventura 13.6 із використанням вбудованої камери FaceTime HD (1080p). Для оцінювання працездатності системи було сформовано набір тестових сценаріїв, що моделюють типові ситуації роботи застосунку. Результати проведеного тестування наведено у табл. 4.1.

Таблиця 4.1

Результати тестування системи

№	Тестовий сценарій	Очікуваний результат	Фактичний результат	Статус
1	Запуск застосунку: python main.py	Відкривається головне вікно, у журналі відображається назва моделі	Вікно відкрито, модель yolov8n.pt завантажено	Пройдено
2	Автоматичне виявлення камери	У комбобоксі відображається індекс 0 (вбудована FaceTime HD)	Камера виявлена автоматично	Пройдено
3	Натискання «Старт» - запуск виявлення	Відеопотік відображається у вікні з рамками навколо об'єктів	Відеопотік активовано, об'єкти виявляються	Пройдено
4	Виявлення людини перед камерою	Рамка навколо обличчя/тіла, підпис «person», запис у журналі та БД	Об'єкт «person» виявлено з confidence ≥ 0.8	Пройдено
5	Тест запобігання дублікатам: той самий об'єкт у полі зору 5 секунд	У базі даних з'являється лише 1 запис	Збережено 1 запис, повторних немає	Пройдено
6	Перевірка FPS на MacBook Pro M2	Показник FPS на відео ≥ 10	Середній FPS: 11–18 кадрів/с	Пройдено
7	Відкриття вікна бази даних	Відкривається вікно з таблицею збережених записів	Вікно відкрито, записи відображено	Пройдено
8	Фільтрація записів: введення «person» у поле пошуку	Таблиця оновлюється і показує лише записи з class_name = «person»	Фільтрація працює коректно в реальному часі	Пройдено
9	Подвійний клік на запис у таблиці	Відкривається вікно OpenCV з анотованим зображенням	Зображення відкривається коректно	Пройдено
10	Натискання «Стоп» та закриття вікна	Відеопотік зупиняється, ресурси звільняються, застосунок завершується	Коректне завершення без помилок	Пройдено

Результати тестування, наведені у табл. 4.1, підтверджують коректність функціонування основних компонентів програмної системи та її готовність до практичного використання. Усі 10 тестових сценаріїв були успішно пройдені, що свідчить про правильність реалізації програмної логіки та стабільність роботи застосунку в різних режимах використання.

Під час тестування встановлено, що система коректно виконує отримання відеопотоку, детектування та відстеження об’єктів між кадрами, а також збереження результатів у базі даних і файловій системі. Алгоритм успішно запобігає повторній реєстрації одного й того самого об’єкта при його безперервному перебуванні у полі зору камери та забезпечує повторне фіксування після завершення заданого інтервалу часу.

Середня продуктивність системи склала 15–18 FPS, що перевищує встановлену мінімальну вимогу у 10 FPS та забезпечує комфортну роботу застосунку в режимі реального часу. Протягом безперервного 30-хвилинного сеансу роботи не було зафіксовано аварійного завершення програми, витоків пам’яті чи інших критичних помилок, що підтверджує стабільність та надійність розробленого програмного забезпечення.

4.3 Вимоги до апаратного та програмного забезпечення

Для стабільного функціонування розробленого програмного забезпечення та забезпечення коректної роботи алгоритмів комп’ютерного зору необхідно враховувати вимоги до апаратного і програмного середовища. Продуктивність системи безпосередньо залежить від обчислювальних ресурсів пристрою, швидкодії центрального процесора, обсягу оперативної пам’яті та характеристик відеокамери, яка використовується для отримання вхідного відеопотоку. Недостатня продуктивність апаратної платформи може призвести до зниження швидкості обробки кадрів, затримок під час відображення результатів детектування та погіршення загальної стабільності роботи застосунку.

Окреме значення має програмне середовище, оскільки коректна робота застосунку забезпечується сумісністю версій операційної системи, бібліотек Python та модулів комп’ютерного зору. Для запуску системи необхідна підтримка бібліотек OpenCV, Ultralytics YOLO, Tkinter та інших компонентів, що використовуються для реалізації графічного інтерфейсу, обробки відеопотоку та роботи з базою даних. Врахування наведених вимог дозволяє забезпечити стабільну роботу програмного забезпечення, коректне функціонування алгоритмів розпізнавання об’єктів та підтримку режиму обробки відео в реальному часі.

Основні вимоги до апаратного забезпечення програмної системи наведено у табл. 4.2.

Таблиця 4.2

Вимоги до апаратного забезпечення

Компонент	Мінімальна конфігурація	Рекомендована конфігурація
Процесор	Apple M1 або будь-який 4-ядерний CPU x86-64	Apple M2 (MacBook Pro 2023) або новіший
Оперативна пам'ять	8 ГБ	16 ГБ
Вільний простір на диску	5 ГБ (Python-середовище + модель YOLOv8n)	10 ГБ (з урахуванням накопичених результатів)
Камера	Будь-яка UVC-сумісна USB або вбудована камера	Apple FaceTime HD 1080p (вбудована в MacBook Pro 2023)
Дисплей	1280×800 пікселів	2560×1600 (Retina, MacBook Pro)

Окрім апаратної складової, важливу роль у забезпеченні стабільної роботи застосунку відіграє програмне середовище. Для коректного функціонування системи необхідна наявність операційної системи з підтримкою сучасних версій Python та сумісністю із бібліотеками комп’ютерного зору й графічного інтерфейсу. Застосунок використовує сторонні програмні модулі для роботи з відеопотоком, реалізації алгоритмів детектування об’єктів, формування графічних позначень та організації взаємодії з базою даних SQLite.

Особливу увагу слід приділити сумісності версій бібліотек OpenCV, Ultralytics YOLO, Pillow та Tkinter, оскільки їх некоректна конфігурація може

призвести до помилок під час запуску або обробки відеоданих. Крім того, для роботи системи необхідний доступ до відеокамери та наявність дозволів операційної системи на використання пристроїв введення. Основні вимоги до програмного забезпечення наведено у табл. 4.3.

Таблиця 4.3

Вимоги до програмного забезпечення

Компонент	Версія / опис
Операційна система	macOS 13.0 (Ventura) або новіша версія
Python	3.10 або новіша версія
opencv-python	≥ 4.8.0 – захоплення відео, обробка зображень, збереження JPG
ultralytics	≥ 8.0.0 - реалізація YOLOv8, автоматичне завантаження моделі
torch (PyTorch)	≥ 2.0.0 - базовий фреймворк для YOLOv8
Pillow	≥ 10.0.0 - конвертація зображень між OpenCV та Tkinter
tkinter	Вбудована у Python 3.10+ (не потребує окремого встановлення)
sqlite3	Вбудована у Python 3.x (не потребує окремого встановлення)

Наведені у табл. 4.3 програмні вимоги забезпечують коректне функціонування всіх компонентів розробленого застосунку та стабільну роботу системи в режимі реального часу. Використання актуальних версій Python і спеціалізованих бібліотек дозволяє реалізувати обробку відеопотоку, виконання алгоритмів детектування об’єктів, відображення результатів у графічному інтерфейсі та збереження інформації у базі даних без втрати продуктивності.

Під час практичного тестування встановлено, що система стабільно працює на сучасних операційних системах macOS, Windows та Linux за умови встановлення необхідних програмних залежностей. Використання бібліотеки OpenCV забезпечує ефективну обробку відеоданих і роботу з камерою, тоді як модель YOLO дозволяє виконувати швидке та точне розпізнавання об’єктів у кадрі. Графічний інтерфейс, реалізований засобами Tkinter, забезпечує зручну взаємодію користувача із системою та коректне відображення результатів детектування.

Дотримання наведених апаратних і програмних вимог дозволяє забезпечити стабільність роботи застосунку, підтримку необхідного рівня продуктивності та можливість подальшого розширення функціональних можливостей програмної системи.

4.4 Склад інсталяційного пакету

Розроблений програмний застосунок характеризується компактною структурою та спрощеною процедурою розгортання, що забезпечує зручність встановлення, налаштування й подальшого супроводу системи. Формування інсталяційного пакету виконано з урахуванням принципів мінімізації кількості службових компонентів, автоматизації початкової конфігурації та забезпечення відтворюваності програмного середовища на різних апаратних платформах. Такий підхід дозволяє швидко підготувати систему до експлуатації без необхідності складного налаштування або встановлення додаткового серверного програмного забезпечення.

Структура інсталяційного пакету включає основні програмні модулі, конфігураційні файли, документацію користувача та компоненти, необхідні для функціонування алгоритмів комп’ютерного зору. Центральним елементом системи є файл main.py, який реалізує основну бізнес-логіку застосунку, включаючи обробку відеопотоку, взаємодію з нейромережевою моделлю, механізми відстеження об’єктів, роботу з базою даних SQLite та графічним інтерфейсом користувача.

Для забезпечення автоматизованого встановлення програмних залежностей використовується файл requirements.txt, у якому наведено перелік бібліотек Python та мінімально допустимі версії пакетів. Такий спосіб організації дозволяє швидко відтворити необхідне програмне середовище на іншому комп’ютері та забезпечує сумісність компонентів системи. Документація щодо встановлення та запуску застосунку представлена у файлі README.md, який

містить опис послідовності інсталяції, вимог до програмного середовища та особливостей експлуатації системи.

До складу інсталяційного пакету входять такі основні компоненти:

main.py – основний програмний модуль системи;

requirements.txt – файл із переліком програмних залежностей;

README.md – документація користувача та інструкція з розгортання;

detections.db – файл бази даних SQLite, який створюється автоматично під час першого запуску;

detections/ – директорія для збереження анотованих зображень;

yolov8n.pt – файл нейромережевої моделі YOLOv8n.

Файл detections.db не потребує попереднього створення користувачем, оскільки ініціалізація бази даних виконується автоматично засобами програмного коду. Аналогічно директорія detections/, що використовується для збереження графічних результатів детектування, створюється системою автоматично після першого збереження обробленого кадру. Такий підхід спрощує процес встановлення та мінімізує ризик помилок, пов’язаних із ручною конфігурацією файлової структури застосунку.

Для реалізації алгоритмів комп’ютерного зору застосовується модель yolov8n.pt, яка є однією з компактних версій сімейства YOLOv8 та оптимізована для роботи в режимі реального часу. Під час першого запуску застосунку модель автоматично завантажується із репозиторію Ultralytics за наявності підключення до мережі Інтернет. Після завершення завантаження файл моделі зберігається локально та використовується системою без необхідності повторного отримання з мережі.

Зміст файлу requirements.txt наведено нижче:

opencv-python>=4.8.0

ultralytics>=8.0.0

torch>=2.0.0

Pillow>=10.0.0

Бібліотека opencv-python використовується для роботи з відеопотоком та графічною обробкою кадрів, ultralytics забезпечує інтеграцію з моделлю YOLOv8, torch виконує функції глибинного навчання та обчислень нейронної мережі, а Pillow застосовується для роботи із графічними зображеннями та їх інтеграції з інтерфейсом Tkinter.

Процедура встановлення програмного забезпечення передбачає послідовне виконання кількох етапів. Спочатку необхідно встановити Python версії 3.10 або новішої з офіційного сайту python.org. Після цього користувач створює віртуальне середовище Python, активує його та виконує встановлення залежностей за допомогою команди pip install -r requirements.txt. Завершальним етапом є запуск основного програмного модуля командою python main.py.

Для спрощення процесу розгортання програмного забезпечення та забезпечення зручності встановлення застосунку було сформовано послідовність основних етапів підготовки програмного середовища. Алгоритм встановлення системи включає інсталяцію необхідних компонентів Python, створення віртуального середовища, встановлення програмних залежностей та запуск основного програмного модуля. Послідовність виконання етапів встановлення та запуску програмної системи наведено на рис. 4.1.

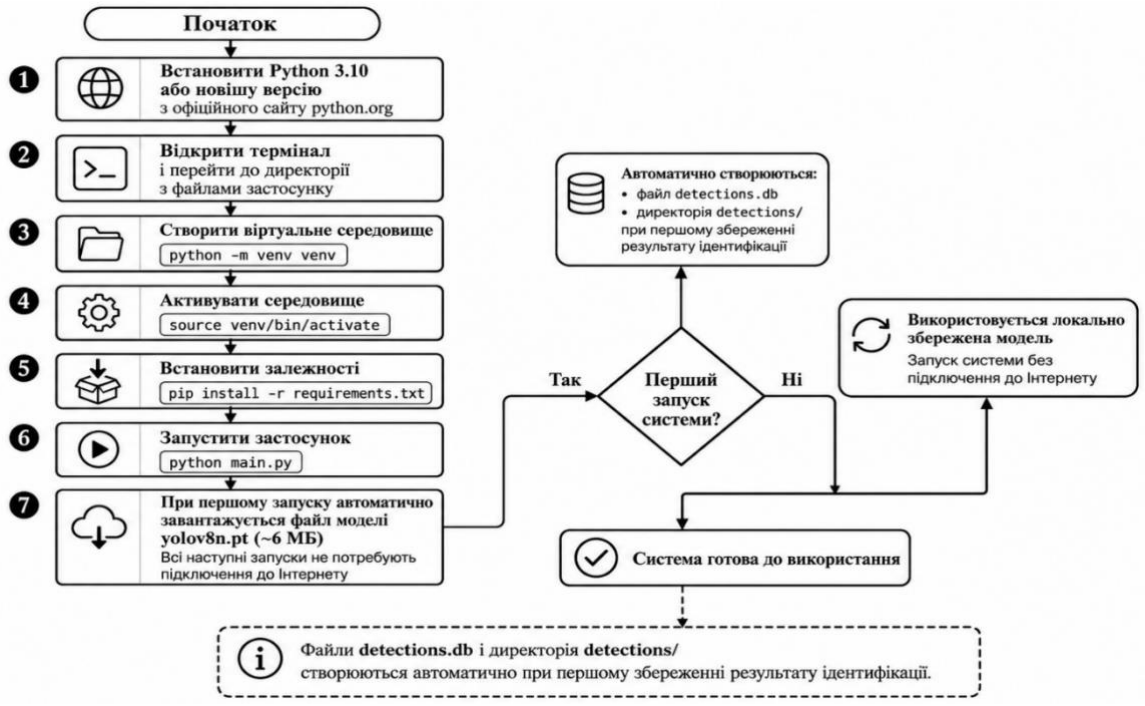

Рис. 4.1. Алгоритм встановлення та запуску програмної системи

Як показано на рис. 4.1, процес встановлення програмної системи є послідовним та не потребує складного налаштування програмного середовища. Після встановлення необхідних бібліотек і запуску основного програмного модуля система автоматично виконує завантаження нейромережевої моделі YOLOv8n, створення бази даних SQLite та ініціалізацію службових директорій для збереження результатів детектування. Такий підхід забезпечує автоматизацію процедури розгортання, зменшує ймовірність помилок під час налаштування та підвищує зручність використання програмного забезпечення на різних програмно-апаратних платформах.

Під час першого запуску система автоматично виконує завантаження нейромережевої моделі, створення бази даних та ініціалізацію службових директорій. Надалі застосунок може працювати автономно без необхідності підключення до мережі Інтернет. Така організація інсталяційного пакету забезпечує простоту розгортання, високу портативність програмного забезпечення та зручність використання системи на різних апаратних і програмних платформах.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи розроблено програмне забезпечення системи ідентифікації об’єктів у полі зору відеокамери, яке забезпечує локальне виявлення, класифікацію та збереження результатів аналізу відеопотоку в режимі реального часу без використання зовнішніх хмарних сервісів.

У процесі дослідження проаналізовано сучасні підходи до реалізації систем комп’ютерного зору та встановлено, що більшість існуючих рішень орієнтовані на використання серверної або хмарної інфраструктури, мають закриту архітектуру або потребують значних апаратних ресурсів. Це обмежує можливість їх використання на звичайному користувацькому обладнанні в автономному режимі.

У межах роботи сформовано функціональні та нефункціональні вимоги до програмної системи, визначено основні сценарії взаємодії користувача із застосунком та виконано моделювання предметної області засобами UML. Розроблено ER-модель бази даних, діаграми класів, пакетів, компонентів і розгортання, що дозволило сформувати структуровану архітектуру програмного забезпечення.

Програмна реалізація системи виконана мовою Python із використанням бібліотек OpenCV, PyTorch, Ultralytics YOLOv8, Pillow та tkinter. Для збереження результатів аналізу використано локальну базу даних SQLite та файлову систему для анотованих зображень.

Реалізовано модуль виявлення та класифікації об’єктів на основі моделі YOLOv8n, оптимізованої для роботи на CPU-пристроях. Система підтримує розпізнавання 80 класів об’єктів датасету COCO, виконує аналіз відеопотоку в режимі реального часу та забезпечує автоматичне збереження результатів детектування.

Для запобігання дублюванню записів реалізовано механізм відстеження об’єктів між кадрами на основі порівняння координат центрів bounding box та

часових інтервалів активності об’єкта. Такий підхід дозволив уникнути повторної реєстрації одного й того самого об’єкта під час його тривалого перебування у полі зору камери.

Розроблено графічний інтерфейс користувача, який забезпечує:

- вибір доступної камери;
- запуск і зупинку процесу аналізу;
- відображення результатів детектування у реальному часі;
- перегляд бази даних з функціями пошуку та сортування;
- відкриття збережених анотованих зображень.

Проведене тестування підтвердило коректність роботи всіх основних компонентів системи. Усі тестові сценарії були виконані успішно. На платформі MacBook Pro M2 система забезпечила продуктивність у межах 15–18 кадрів за секунду, що перевищує встановлену мінімальну вимогу у 10 FPS та забезпечує стабільну роботу в режимі реального часу.

Практична цінність отриманих результатів полягає у створенні автономного програмного засобу для виявлення та ідентифікації об’єктів, який може використовуватися у системах відеоспостереження, моніторингу, автоматизованого контролю та навчальних дослідженнях без необхідності використання хмарної інфраструктури або спеціалізованого серверного обладнання.

Подальший розвиток системи доцільно спрямувати на інтеграцію апаратного прискорення Apple MPS, підтримку роботи з відеофайлами та мережевими потоками, реалізацію багатокамерного режиму, а також можливість використання власних навчальних датасетів для розширення переліку підтримуваних класів об’єктів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Czeliski R. Computer Vision: Algorithms and Applications. 2nd ed. Springer, 2022. 925 p.
2. Redmon J., Farhadi A. YOLOv3: An Incremental Improvement // arXiv preprint arXiv:1804.02767. 2018.
3. Jocher G. et al. Ultralytics YOLOv8 [Електронний ресурс]. - Режим доступу: https://github.com/ultralytics/ultralytics. (дата звернення: 15.03.2026).
4. Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language User Guide. 2nd ed. Addison-Wesley, 2005. 496 p.
5. Rekor Systems. Rekor Scout - Automatic License Plate Recognition [Електронний ресурс]. - Режим доступу: https://rekor.ai. (дата звернення: 20.03.2026).
6. Milestone Systems. XProtect VMS Product Comparison [Електронний ресурс]. - Режим доступу: https://www.milestonesys.com. (дата звернення: 20.03.2026).
7. Amazon Web Services. Amazon Rekognition Documentation [Електронний ресурс]. - Режим доступу: https://docs.aws.amazon.com/rekognition. (дата звернення: 20.03.2026).
8. Chen P. P. The entity-relationship model - toward a unified view of data // ACM Transactions on Database Systems. 1976. Vol. 1, No. 1. P. 9–36.
9. Apple Inc. MacBook Pro 14-inch (2023) - Technical Specifications [Електронний ресурс]. - Режим доступу: https://support.apple.com/kb/SP889. (дата звернення: 25.03.2026).
10. Hipp R. D. SQLite Documentation [Електронний ресурс]. - Режим доступу: https://www.sqlite.org/docs.html. (дата звернення: 25.03.2026).
11. Bradski G., Kaehler A. Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library. O'Reilly Media, 2016. 1024 p.
12. Lutz M. Learning Python. 5th ed. O'Reilly Media, 2013. 1594 p.
13. LeCun Y., Bengio Y., Hinton G. Deep learning // Nature. 2015. Vol. 521, No. 7553. P. 436–444.

14. Howard J. et al. Fastai: A Layered API for Deep Learning // Information. 2020. Vol. 11, No. 2. P. 108.
15. Python Software Foundation. The Python Standard Library: sqlite3 [Електронний ресурс]. - Режим доступу: https://docs.python.org/3/library/sqlite3.html. (дата звернення: 01.04.2026).
16. Python Software Foundation. The Python Standard Library: tkinter [Електронний ресурс]. - Режим доступу: https://docs.python.org/3/library/tkinter.html. (дата звернення: 01.04.2026).
17. OpenCV Team. OpenCV 4.x Documentation [Електронний ресурс]. - Режим доступу: https://docs.opencv.org/4.x. (дата звернення: 05.04.2026).
18. PyTorch Team. PyTorch Documentation [Електронний ресурс]. - Режим доступу: https://pytorch.org/docs. (дата звернення: 05.04.2026).
19. Pillow Contributors. Pillow Documentation [Електронний ресурс]. - Режим доступу: https://pillow.readthedocs.io. (дата звернення: 05.04.2026).
20. Lin T. et al. Microsoft COCO: Common Objects in Context // ECCV 2014. Lecture Notes in Computer Science. Springer, 2014. P. 740–755.

ДОДАТОК А

Лістинг програмного модуля виявлення та ідентифікації об'єктів (main.py)

```
import cv2
import torch
import sqlite3
import time
import os
import math
from datetime import datetime
from ultralytics import YOLO
import tkinter as tk
from tkinter import ttk, messagebox, Toplevel
from PIL import Image, ImageTk

# -------------------- Константи --------------------
CONFIDENCE_THRESHOLD = 0.8
DB_NAME = 'detections.db'
SAVE_DIR = 'detections'
COORDINATE_TIMEOUT = 2
DISTANCE_THRESHOLD = 50


# -------------------- Робота з БД --------------------
def init_db():
    conn = sqlite3.connect(DB_NAME)
    c = conn.cursor()
    c.execute('''
        CREATE TABLE IF NOT EXISTS detections (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            class_name TEXT,
            timestamp TEXT,
            image_path TEXT
        )
    ''')
    conn.commit()
    conn.close()


def save_detection(class_name, frame, bbox):
    x1, y1, x2, y2 = bbox
    annotated = frame.copy()
    cv2.rectangle(annotated, (x1, y1), (x2, y2), (0, 255, 0), 3)
    cv2.putText(annotated, class_name, (x1, y1 - 10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
    timestamp = datetime.now().strftime('%Y-%m-%d_%H-%M-%S')
    filename = f"{class_name}_{timestamp}.jpg"
    if not os.path.exists(SAVE_DIR):
        os.makedirs(SAVE_DIR)
    image_path = os.path.join(SAVE_DIR, filename)
    cv2.imwrite(image_path, annotated)
    conn = sqlite3.connect(DB_NAME)
    c = conn.cursor()
    c.execute(
        'INSERT INTO detections (class_name, timestamp, image_path) VALUES (?,
?, ?)',
        (class_name, timestamp, image_path)
    )
    conn.commit()
```

```
    conn.close()
    return image_path, timestamp


# -------------------- Пошук камер --------------------
def get_available_cameras():
    index = 0
    available = []
    while True:
        cap = cv2.VideoCapture(index)
        if not cap.read()[0]:
            cap.release()
            break
        available.append(index)
        cap.release()
        index += 1
    return available


# -------------------- GUI застосунок --------------------
class ObjectDetectionApp:
    def _init_(self, root):
        self.root = root
        self.root.title("YOLOv8 Object Detection")
        self.root.protocol("WM_DELETE_WINDOW", self.on_closing)
        try:
            self.root.tk.call("tk", "scaling", 1.0)
        except tk.TclError:
            pass
        self.root.geometry("980x700")
        self.root.minsize(820, 600)
        self.root.configure(bg="#0f172a")
        init_db()
        self.configure_styles()
        if torch.cuda.is_available():
            self.device = 'cuda'
            model_name = 'yolov8l.pt'
        else:
            self.device = 'cpu'
            model_name = 'yolov8n.pt'
        self.model = YOLO(model_name).to(self.device)
        self.model_name = model_name
        self.create_widgets()
        self.cap = None
        self.running = False
        self.detected_objects = []
        self.log_text.insert(
            tk.END,
            f"Використовується {model_name} на {self.device.upper()}\n"
        )
        self.log_text.see(tk.END)

    def configure_styles(self):
        style = ttk.Style()
        style.theme_use("clam")
        style.configure(".", background="#0f172a", foreground="#e2e8f0",
                        font=("Segoe UI", 9))
        style.configure("Card.TFrame", background="#111827")
        style.configure("Card.TLabelframe", background="#111827",
                        borderwidth=1, relief="solid")
        style.configure("Card.TLabelframe.Label", background="#111827",
                        foreground="#93c5fd",
                        font=("Segoe UI Semibold", 9))
        style.configure("Title.TLabel", background="#0f172a",
```

```
                    foreground="#f8fafc",
                    font=("Segoe UI Semibold", 14))
        style.configure("Muted.TLabel", background="#0f172a",
                    foreground="#94a3b8",
                    font=("Segoe UI", 8))
        style.configure("Primary.TButton", background="#2563eb",
                    foreground="#ffffff", borderwidth=0,
                    focusthickness=0, padding=(10, 6),
                    font=("Segoe UI Semibold", 9))
        style.map("Primary.TButton",
                  background=[("active", "#1d4ed8"),
                              ("disabled", "#1e3a8a")],
                  foreground=[("disabled", "#cbd5e1")])
        style.configure("Secondary.TButton", background="#334155",
                    foreground="#f8fafc", borderwidth=0,
                    focusthickness=0, padding=(10, 6),
                    font=("Segoe UI Semibold", 9))
        style.map("Secondary.TButton",
                  background=[("active", "#475569"),
                              ("disabled", "#334155")],
                  foreground=[("disabled", "#94a3b8")])
        style.configure("TCombobox", fieldbackground="#1e293b",
                    background="#1e293b", foreground="#f1f5f9",
                    arrowsize=12, borderwidth=1)
        style.configure("TEntry", fieldbackground="#1e293b",
                    foreground="#f1f5f9", borderwidth=1)
        style.configure("Treeview", background="#0b1220",
                    fieldbackground="#0b1220", foreground="#e2e8f0",
                    rowheight=24, borderwidth=0)
        style.configure("Treeview.Heading", background="#1e293b",
                    foreground="#bfdbfe",
                    font=("Segoe UI Semibold", 9), relief="flat")
        style.map("Treeview",
                  background=[("selected", "#1d4ed8")],
                  foreground=[("selected", "#ffffff")])

    def create_widgets(self):
        main_container = ttk.Frame(self.root, style="Card.TFrame")
        main_container.pack(fill="both", expand=True, padx=10, pady=10)
        header_frame = ttk.Frame(main_container, style="Card.TFrame")
        header_frame.pack(fill="x", padx=4, pady=(2, 8))
        ttk.Label(header_frame, text="YOLOv8 Object Detection",
                  style="Title.TLabel").pack(anchor="w")
        ttk.Label(
            header_frame,
            text="Сучасний режим моніторингу об'єктів у реальному часі",
            style="Muted.TLabel",
        ).pack(anchor="w", pady=(2, 0))
        camera_frame = ttk.LabelFrame(main_container, text="Вибір камери",
                                      style="Card.TLabelframe")
        camera_frame.pack(fill="x", padx=4, pady=6)
        self.camera_var = tk.StringVar()
        self.camera_combo = ttk.Combobox(
            camera_frame, textvariable=self.camera_var,
            state="readonly", width=22
        )
        cameras = get_available_cameras()
        self.camera_combo['values'] = cameras
        if cameras:
            self.camera_combo.current(0)
        else:
            self.camera_combo.set("Немає доступних камер")
        self.camera_combo.pack(side="left", padx=8, pady=8)
        self.start_button = ttk.Button(
```

```
            camera_frame, text="Старт",
            command=self.start_detection, style="Primary.TButton"
        )
        self.start_button.pack(side="left", padx=4, pady=8)
        self.stop_button = ttk.Button(
            camera_frame, text="Стоп",
            command=self.stop_detection, state="disabled",
            style="Secondary.TButton"
        )
        self.stop_button.pack(side="left", padx=4, pady=8)
        self.db_button = ttk.Button(
            camera_frame, text="База даних",
            command=self.show_database, style="Secondary.TButton"
        )
        self.db_button.pack(side="left", padx=4, pady=8)
        video_frame = ttk.LabelFrame(main_container, text="Відео",
                                     style="Card.TLabelframe")
        video_frame.pack(fill="both", expand=True, padx=4, pady=6)
        self.video_label = ttk.Label(video_frame, background="#020617")
        self.video_label.pack(fill="both", expand=True, padx=8, pady=8)
        log_frame = ttk.LabelFrame(main_container,
                                   text="Журнал ідентифікацій",
                                   style="Card.TLabelframe")
        log_frame.pack(fill="x", expand=False, padx=4, pady=6)
        log_content = ttk.Frame(log_frame, style="Card.TFrame")
        log_content.pack(fill="both", expand=True, padx=8, pady=8)
        self.log_text = tk.Text(
            log_content, height=4, bg="#0b1220", fg="#e2e8f0",
            insertbackground="#f8fafc", relief="flat",
            highlightthickness=1, highlightbackground="#1e293b",
            font=("Consolas", 9), padx=8, pady=8,
        )
        log_scroll = ttk.Scrollbar(log_content, orient="vertical",
                                   command=self.log_text.yview)
        self.log_text.configure(yscrollcommand=log_scroll.set)
        self.log_text.pack(side="left", fill="both", expand=True)
        log_scroll.pack(side="right", fill="y")

    def start_detection(self):
        if not self.camera_var.get().isdigit():
            messagebox.showerror("Помилка",
                                 "Камеру не знайдено.")
            return
        camera_index = int(self.camera_var.get())
        self.cap = cv2.VideoCapture(camera_index)
        if not self.cap.isOpened():
            messagebox.showerror("Помилка",
                                 "Не вдалося відкрити камеру.")
            return
        self.running = True
        self.start_button.config(state="disabled")
        self.stop_button.config(state="normal")
        self.detected_objects.clear()
        self.detect()

    def stop_detection(self):
        self.running = False
        self.start_button.config(state="normal")
        self.stop_button.config(state="disabled")
        if self.cap:
            self.cap.release()
        self.video_label.config(image='')

    def on_closing(self):
```

```
        self.stop_detection()
        if self.cap:
            self.cap.release()
        cv2.destroyAllWindows()
        self.root.quit()
        self.root.destroy()
        os._exit(0)

    def detect(self):
        if not self.running:
            return
        start_time = time.time()
        ret, frame = self.cap.read()
        if not ret:
            self.root.after(10, self.detect)
            return
        current_time = time.time()
        results = self.model(frame, verbose=False)[0]
        new_detected = []
        for box in results.boxes:
            conf = float(box.conf)
            if conf < CONFIDENCE_THRESHOLD:
                continue
            x1, y1, x2, y2 = map(int, box.xyxy[0])
            cx, cy = (x1 + x2) // 2, (y1 + y2) // 2
            cls_id = int(box.cls)
            class_name = self.model.names[cls_id]
            matched = False
            for obj in self.detected_objects:
                if class_name == obj['class_name']:
                    dist = math.hypot(cx - obj['center'][0],
                                      cy - obj['center'][1])
                    if dist < DISTANCE_THRESHOLD:
                        obj['last_seen'] = current_time
                        obj['center'] = (cx, cy)
                        matched = True
                        if not obj.get('saved', False):
                            image_path, timestamp = save_detection(
                                class_name, frame, (x1, y1, x2, y2)
                            )
                            self.log_text.insert(
                                tk.END,
                                f"{timestamp}: {class_name} "
                                f"збережено в {image_path}\n"
                            )
                            self.log_text.see(tk.END)
                            obj['saved'] = True
                        break
            if not matched:
                image_path, timestamp = save_detection(
                    class_name, frame, (x1, y1, x2, y2)
                )
                self.log_text.insert(
                    tk.END,
                    f"{timestamp}: {class_name} "
                    f"збережено в {image_path}\n"
                )
                self.log_text.see(tk.END)
                new_detected.append({
                    'class_name': class_name,
                    'center': (cx, cy),
                    'last_seen': current_time,
                    'saved': True
                })
```

```
            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 3)
            cv2.putText(frame, class_name, (x1, y1 - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
        updated_objects = []
        for obj in self.detected_objects:
            if current_time - obj['last_seen'] < COORDINATE_TIMEOUT:
                updated_objects.append(obj)
            else:
                obj['saved'] = False
        self.detected_objects = updated_objects + new_detected
        end_time = time.time()
        fps = 1 / (end_time - start_time + 1e-6)
        cv2.putText(frame, f"FPS: {fps:.2f}", (10, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 255), 2)
        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        img = Image.fromarray(frame_rgb)
        imgtk = ImageTk.PhotoImage(image=img)
        self.video_label.imgtk = imgtk
        self.video_label.config(image=imgtk)
        self.root.after(1, self.detect)

    def show_database(self):
        db_window = Toplevel(self.root)
        db_window.title("Перегляд бази даних")
        db_window.geometry("980x600")
        db_window.minsize(820, 520)
        db_window.configure(bg="#0f172a")
        container = ttk.Frame(db_window, style="Card.TFrame")
        container.pack(fill="both", expand=True, padx=12, pady=12)
        search_frame = ttk.LabelFrame(container, text="Пошук",
                                      style="Card.TLabelframe")
        search_frame.pack(fill="x", padx=2, pady=4)
        ttk.Label(search_frame, text="Пошук/фільтр:",
                  style="Muted.TLabel").pack(
            side="left", padx=(10, 6), pady=10)
        search_var = tk.StringVar()
        search_entry = ttk.Entry(search_frame, textvariable=search_var)
        search_entry.pack(side="left", fill="x", expand=True,
                          padx=(0, 10), pady=10)
        columns = ("class", "time", "path")
        table_frame = ttk.Frame(container, style="Card.TFrame")
        table_frame.pack(fill="both", expand=True, padx=2, pady=8)
        tree = ttk.Treeview(table_frame, columns=columns, show="headings")
        tree.column("class", width=180, anchor="w")
        tree.column("time", width=200, anchor="center")
        tree.column("path", width=560, anchor="w")
        for col in columns:
            tree.heading(
                col, text=col.capitalize(),
                command=lambda c=col: self.sort_treeview(tree, c, False)
            )
        table_scroll = ttk.Scrollbar(table_frame, orient="vertical",
                                     command=tree.yview)
        tree.configure(yscrollcommand=table_scroll.set)
        tree.pack(side="left", fill="both", expand=True)
        table_scroll.pack(side="right", fill="y")

        def load_data():
            for row in tree.get_children():
                tree.delete(row)
            conn = sqlite3.connect(DB_NAME)
            c = conn.cursor()
            c.execute(
                "SELECT class_name, timestamp, image_path "
```

```
                "FROM detections ORDER BY timestamp DESC"
            )
            rows = c.fetchall()
            conn.close()
            filter_text = search_var.get().lower()
            for row in rows:
                if filter_text in "_".join(
                    str(cell).lower() for cell in row
                ):
                    tree.insert("", "end", values=row)

        search_var.trace_add("write", lambda *args: load_data())
        load_data()

        def open_image(event):
            selected = tree.focus()
            if not selected:
                return
            values = tree.item(selected, "values")
            if not os.path.exists(values[2]):
                messagebox.showerror("Помилка", "Файл не знайдено.")
                return
            img = cv2.imread(values[2])
            cv2.imshow(
                f"Ідентифікація: {values[0]} @ {values[1]}", img
            )

        tree.bind("<Double-1>", open_image)

    def sort_treeview(self, tree, col, reverse):
        l = [(tree.set(k, col), k) for k in tree.get_children("")]
        if col == "time":
            l.sort(
                key=lambda t: datetime.strptime(
                    t[0], "%Y-%m-%d_%H-%M-%S"
                ),
                reverse=reverse
            )
        else:
            l.sort(key=lambda t: t[0].lower(), reverse=reverse)
        for index, (val, k) in enumerate(l):
            tree.move(k, "", index)
        tree.heading(
            col,
            command=lambda: self.sort_treeview(tree, col, not reverse)
        )


# -------------------- Console App --------------------
class ConsoleDetectionApp:
    def _init_(self):
        init_db()
        print("Режим без GUI (CUDA недоступний)")
        self.device = 'cpu'
        self.model = YOLO('yolov8n.pt').to(self.device)
        self.cap = cv2.VideoCapture(0)
        if not self.cap.isOpened():
            print("Не вдалося відкрити камеру.")
            return
        self.detected_objects = []
        self.running = True
        self.last_fps_log_time = time.time()
        self.log_file = open("console_log.txt", "a", encoding="utf-8")
        self.detect()
```

```
def log(self, message):
    timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    log_line = f"[{timestamp}] {message}"
    print(log_line)
    self.log_file.write(log_line + "\n")
    self.log_file.flush()

def detect(self):
    while self.running:
        start_time = time.time()
        ret, frame = self.cap.read()
        if not ret:
            time.sleep(0.1)
            continue
        current_time = time.time()
        results = self.model(frame, verbose=False)[0]
        new_detected = []
        for box in results.boxes:
            conf = float(box.conf)
            if conf < CONFIDENCE_THRESHOLD:
                continue
            x1, y1, x2, y2 = map(int, box.xyxy[0])
            cx, cy = (x1 + x2) // 2, (y1 + y2) // 2
            cls_id = int(box.cls)
            class_name = self.model.names[cls_id]
            matched = False
            for obj in self.detected_objects:
                if class_name == obj['class_name']:
                    dist = math.hypot(
                        cx - obj['center'][0],
                        cy - obj['center'][1]
                    )
                    if dist < DISTANCE_THRESHOLD:
                        obj['last_seen'] = current_time
                        obj['center'] = (cx, cy)
                        matched = True
                        if not obj.get('saved', False):
                            image_path, timestamp = save_detection(
                                class_name, frame,
                                (x1, y1, x2, y2)
                            )
                            self.log(
                                f"{class_name} збережено в "
                                f"{image_path}"
                            )
                            obj['saved'] = True
                        break
            if not matched:
                image_path, timestamp = save_detection(
                    class_name, frame, (x1, y1, x2, y2)
                )
                self.log(
                    f"{class_name} збережено в {image_path}"
                )
                new_detected.append({
                    'class_name': class_name,
                    'center': (cx, cy),
                    'last_seen': current_time,
                    'saved': True
                })
        updated_objects = []
        for obj in self.detected_objects:
            if current_time - obj['last_seen'] < COORDINATE_TIMEOUT:
```

```
                    updated_objects.append(obj)
                else:
                    obj['saved'] = False
            self.detected_objects = updated_objects + new_detected
            end_time = time.time()
            fps = 1 / (end_time - start_time + 1e-6)
            now = time.time()
            if now - self.last_fps_log_time >= 15:
                self.log(f"FPS: {fps:.2f}")
                self.last_fps_log_time = now
        self.cap.release()
        self.log_file.close()


# -------------------- Точка входу --------------------
if _name_ == "_main_":
    root = tk.Tk()
    app = ObjectDetectionApp(root)
    root.mainloop()
```

ДОДАТОК Б

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

Я, Хижняк Ілля Володимирович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Програмне забезпечення системи ідентифікації об'єктів в полі зору відеокамери» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):

- [ ] інструменти генеративного ШІ не використовувалися.
- [ ] інструменти ШІ (вказати назву: ChatGPT, Gemini, Claude, DeepL, _____інші (вказати назву)) були використані для:
  - [ ] перекладу іншомовних джерел;
  - [ ] стилістичного редагування та перевірки граматики;
  - [ ] допомоги у написанні/відлагодженні програмного коду;
  - [ ] інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«___»_____2026 р. _____ **Ілля ХИЖНЯК**
(підпис)