ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МІСЬКОГО ГОСПОДАРСТВА ІМЕНІ О. М. БЕКЕТОВА

Пояснювальна записка
до кваліфікаційної роботи бакалавра

на тему:

«Розробка системи розпізнавання об'єктів у відеопотоці з використанням глибоких нейронних мереж»

Виконав: здобувач вищої освіти
групи КН 2021-1
спеціальності 122 – Комп’ютерні науки
Владислав БУГАРЕНКО
Керівник: Марія ВОЄВОДІНА
Рецензент: Микола КАРПЕНКО

м. Харків – 2025 рік

Харківський національний університет міського господарства імені О. М. Бекетова
(повне найменування закладу вищої освіти)
Навчально-науковий Інститут енергетичної, інформаційної та транспортної інфраструктури
Кафедра комп'ютерних наук та інформаційних технологій
Рівень вищої освіти перший (бакалаврський)
Спеціальність 122 Комп'ютерні науки
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри КНтаІТ

Марина НОВОЖИЛОВА
«26» 06 2025 року

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

Бугаренка Владислава Олександровича
(прізвище, ім'я, по батькові)

1. Тема роботи «Розробка системи розпізнавання об'єктів у відеопотоці з використанням глибоких нейронних мереж»

керівник роботи Воєводіна М.Ю., старший викладач кафедри КН та ІТ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від «09» травня 2025 р. № 341-03
2. Термін подання студентом роботи 21.06.2025р.
3. Вихідні дані до роботи Проєктування і реалізація концепції та розробки системи розпізнавання об'єктів у відеопотоці на базі нейромережі YOLOv8
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- провести аналіз предметної області та огляд аналогів;
- побудувати функціональну модель роботи користувача з ПЗ;
- визначити вимоги до функціональності клієнтської частини;
- розробити графічний інтерфейс користувача (GUI) для відеоаналізу та роботи з камерою;
- інтегрувати модуль обробки відео з системою візуального відображення результатів;
- реалізувати функції збереження результатів (графіки, координати, теплові карти);
- провести тестування та оцінити ефективність роботи клієнтської частини.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):

Презентація – 18 аркушів

6. Консультанти розділів роботи

Розділ	Ім'я та Прізвище, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Марія ВОЄВОДІНА, старший викладач кафедри КН та ІТ	15.05.2025	16.05.2025
2	Марія ВОЄВОДІНА, старший викладач кафедри КН та ІТ	24.05.2025	29.05.2025
3	Марія ВОЄВОДІНА, старший викладач кафедри КН та ІТ	03.06.2025	05.06.2025
4	Вікторія МАЛИШЕВА, к. т. н., доцент кафедри ОП та БЖ	07.06.2025	08.06.2025

7. Дата видачі завдання 15.05.2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вибір теми дипломної роботи	10.05.2025	Виконано
2	Затвердження тем, наукових керівників, завдань та календарного плану підготовки кваліфікаційної роботи	11.05.2025	Виконано
3	Написання I розділу	16.05.2025	Виконано
4	Написання II розділу	29.05.2025	Виконано
5	Написання III розділу	05.06.2025	Виконано
6	Написання IV розділу	08.06.2025	Виконано
7	Подання дипломної роботи керівнику	10.06.2025	Виконано
8	Робота по усуненню зауважень керівника, уточнення і доповнення практичного матеріалу, оформлення додатків до роботи	13.06.2025	Виконано
9	Подання доопрацьованого варіанту роботи керівнику	18.06.2025	Виконано
10	Офіційний захист матеріалів дипломної роботи на засіданні екзаменаційної комісії	27.06.2025	Виконано

Студент _____ Владислав БУГАРЕНКО
(підпис) (ім'я та прізвище)

Керівник роботи _____ Марія ВОЄВОДІНА
(підпис) (ім'я та прізвище)

АНОТАЦІЯ

Структура та обсяг роботи.

Пояснювальна записка кваліфікаційної роботи бакалавра студента групи КН 2021-1 спеціальності 122 Комп’ютерні науки Бугаренко Владислава Олександровича на тему «Розробка системи розпізнавання об'єктів у відеопотоці з використанням глибоких нейронних мереж» має 4 розділи, включає 21 рисунків, 8 таблиці, 24 джерел, 9 додатки.

Предмет дослідження: програмна система розпізнавання об'єктів на відео з використанням глибинного навчання.

Мета роботи: створення програмного забезпечення для розпізнавання об’єктів у відеопотоці з інтеграцією моделі YOLOv8, графічного інтерфейсу та забезпеченням режимів обробки відео в реальному часі та офлайн.

Методи дослідження: аналіз предметної області, побудова математичних моделей, об’єктно-орієнтоване проєктування, реалізація алгоритмів глибинного навчання, тестування програмного продукту.

Програмне забезпечення: Python 3.12, Visual Studio Code, Ultralytics YOLOv8, ByteTrack, OpenCV, PyTorch, Tkinter, OpenVINO.

Результати: реалізовано систему з двома режимами роботи (відеофайл / камера), з підтримкою розпізнавання та трекінгу об’єктів, інтегровано графічний інтерфейс для користувача, забезпечено стабільну роботу на апаратному забезпеченні середнього рівня.

Рекомендації щодо використання результатів: проєкт може використовуватись як базовий інструмент для досліджень у сфері відеоаналітики, комп’ютерного зору та правоохоронної діяльності.

Галузь застосування: системи безпеки, розумне відеоспостереження, аналіз поведінки, дослідницькі задачі в галузі ШІ.

Значущість роботи та висновки: проєкт поєднує сучасні підходи до розробки інтелектуальних систем на базі нейронних мереж, демонструє можливості Python-інтеграції зі спеціалізованими фреймворками, акцентує

увагу на практичному застосуванні методів глибинного навчання та модульному підході до проєктування ПЗ.

Ключові слова: РОЗПІЗНАВАННЯ ОБ’ЄКТІВ, ВІДЕОПОТІК, YOLOV8, PYTHON, DEEP LEARNING, NEURAL NETWORKS, ГРАФІЧНИЙ ІНТЕРФЕЙС.

ANNOTATION

Structure and scope of the work.

The explanatory note of the bachelor's qualification work by student of group KN 2021-1, specialty 122 Computer Science, Buharenko Vladyslav Oleksandrovych, titled «Development of an Object Detection System in a Video Stream Using Deep Neural Networks», consists of 4 sections, includes 21 figures, 8 tables, 24 references, and 9 appendices.

Subject of research: software system for object detection in video using deep learning.

Objective of the work: to develop software capable of detecting objects in a video stream with the integration of the YOLOv8 model, user-friendly graphical interface, and support for both real-time and offline video processing modes.

Research methods: domain analysis, mathematical modeling, object-oriented design, implementation of deep learning algorithms, and software testing.

Software used: Python 3.12, Visual Studio Code, Ultralytics YOLOv8, ByteTrack, OpenCV, PyTorch, Tkinter, OpenVINO.

Results: the system has been implemented with support for two operational modes (video file / camera), integrated object detection and tracking functionality, and a graphical user interface. Stable performance has been achieved on mid-level hardware.

Recommendations for the use of results: the project can be used as a foundational tool for research in video analytics, computer vision, or law enforcement activities.

Field of application: security systems, intelligent video surveillance, behavior analysis, and AI-based research tasks.

Significance and conclusions: the developed system integrates modern approaches to intelligent software design based on neural networks, demonstrates Python integration with specialized deep learning frameworks, and emphasizes practical application of modular software architecture in real-world conditions.

Keywords: OBJECT DETECTION, VIDEO STREAM, YOLOV8, PYTHON, DEEP LEARNING, NEURAL NETWORKS, GRAPHICAL INTERFACE.

ЗМІСТ

ВСТУП

Сучасний світ характеризується стрімким розвитком інформаційних технологій, які стали невід'ємною частиною повсякденного життя, економіки, безпеки та промисловості. Без автоматизованих систем, що базуються на комп'ютерному зорі, важко уявити функціонування багатьох сфер. Камери відеоспостереження, оснащені інтелектуальними алгоритмами, забезпечують розпізнавання об'єктів, відстеження їхнього руху, аналіз поведінки та прийняття рішень у реальному часі. Такі технології є основою для систем безпеки, інтелектуального транспорту, медичної діагностики, промислової автоматизації та концепції розумних міст. Відсутність подібних рішень суттєво ускладнила б контроль транспортних потоків, моніторинг громадських місць, забезпечення якості виробництва та швидке реагування на надзвичайні ситуації. Таким чином, системи розпізнавання об'єктів у відеопотоці стали критично важливими для сучасного суспільства, сприяючи підвищенню ефективності, безпеки та комфорту.

Перспективи розвитку систем розпізнавання об'єктів через відеокамери є надзвичайно широкими завдяки застосуванню глибоких нейронних мереж. У сфері безпеки такі системи дають змогу виявляти підозрілі об'єкти, розпізнавати обличчя та аналізувати поведінку, запобігаючи злочинам і терористичним загрозам. У транспортній галузі вони є ключовим елементом автономних автомобілів, забезпечуючи ідентифікацію пішоходів, транспортних засобів, дорожніх знаків і перешкод. У медицині технології комп'ютерного зору дозволяють аналізувати відеодані для виявлення патологій, наприклад, під час ендоскопічних досліджень. Промисловість активно використовує ці системи для автоматизованого контролю якості, моніторингу стану обладнання та забезпечення безпеки праці. Крім того, розпізнавання об'єктів у реальному часі відкриває можливості для інтеграції з технологіями доповненої реальності, створення інтерактивних інтерфейсів, розробки розумних побутових пристроїв і систем відеонагляду для розумних

міст. У контексті глобальних викликів, таких як урбанізація, зміна клімату та зростання населення, такі технології стають інструментом для оптимізації ресурсів, підвищення безпеки та покращення якості життя.

Особливу роль у реалізації цих систем відіграють глибокі нейронні мережі, зокрема модель YOLO (You Only Look Once), яка вирізняється високою точністю та швидкістю обробки відеопотоку. Завдяки здатності працювати в реальному часі та адаптуватися до різних умов, такі моделі є основою для створення ефективних і надійних систем розпізнавання. Актуальність розробки подібних систем зумовлена зростаючою потребою в автоматизації, обробці великих обсягів відеоданих і швидкому реагуванні на динамічні події. Усе це робить дослідження та створення систем розпізнавання об’єктів у відеопотоці важливим завданням, що має як теоретичне, так і практичне значення.

Мета роботи полягає у розробці системи розпізнавання об’єктів у відеопотоці з використанням глибоких нейронних мереж на базі моделі YOLOv8, яка забезпечує обробку відеофайлів і даних із камери в реальному часі, а також збереження результатів у зручному форматі.

Об’єкт дослідження – процес розробки системи розпізнавання об’єктів у відеопотоці на основі глибоких нейронних мереж.

Предмет дослідження – методи та алгоритми розпізнавання об’єктів, реалізовані за допомогою моделі YOLOv8 і мови програмування Python.

Методи дослідження включають аналіз літературних джерел, порівняльний аналіз аналогів, моделювання процесів обробки відеопотоку, програмування в середовищі Python, а також емпіричне тестування розробленої системи.

Завдання дослідження:

1. Провести аналіз предметної області, включаючи огляд сучасних систем розпізнавання об’єктів і підходів на основі глибоких нейронних мереж.

2. Розробити концептуальну модель системи, що включає обробку відеопотоку, детекцію та трекінг об’єктів.

3. Обґрунтувати вибір засобів розробки, зокрема мови Python, моделі YOLOv8 та відповідних бібліотек.

4. Реалізувати програмний продукт для розпізнавання об’єктів у відеопотоці з графічним інтерфейсом і можливістю збереження результатів.

5. Провести тестування розробленої системи, оцінити її ефективність і точність розпізнавання об’єктів.

Теоретична цінність роботи полягає у систематизації підходів до розробки систем розпізнавання об’єктів із застосуванням глибоких нейронних мереж.

Практична значущість роботи полягає у створенні програмного продукту, який може бути використаний для автоматизованого аналізу відеоданих у різних сферах, а також у розробці рекомендацій щодо його впровадження.

РОЗДІЛ 1
ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

Сучасний етап розвитку інформаційних технологій характеризується широким застосуванням комп’ютерного зору, який забезпечує автоматизовану обробку та аналіз візуальної інформації. Предметне середовище систем розпізнавання об’єктів у відеопотоці охоплює численні сфери, де відеодані є ключовим джерелом інформації: безпека, транспорт, медицина, промисловість, торгівля, побутові системи та концепція розумних міст. Відеопотік, отриманий із камер відеоспостереження, вебкамер, автомобільних відеореєстраторів чи спеціалізованих пристроїв, являє собою послідовність кадрів, що містять дані про об’єкти, їхнє розташування, рух і взаємодію в просторово-часовому контексті.

У сфері безпеки камери відеоспостереження фіксують людей, транспортні засоби чи підозрілі предмети, що вимагає швидкого аналізу для оперативного реагування (рис. 1.1).

Рисунок 1.1 – Система спостереження в будівлі

У транспортній галузі відеопотік із камер на дорогах чи в автомобілях містить пішоходів, інші транспортні засоби, дорожні знаки та перешкоди, що потребує високої точності розпізнавання (рис. 1.2).

Рисунок 1.2 – Розташування камер відеоспостереження на автомагістралі

У медицині відеодані з ендоскопів чи камер хірургічного обладнання використовуються для діагностики патологій (рис. 1.3).

Рисунок 1.3 – Системи камер спостереження в медицині

Промисловість застосовує камери для контролю якості продукції, моніторингу стану обладнання та забезпечення безпеки праці (рис. 1.4).

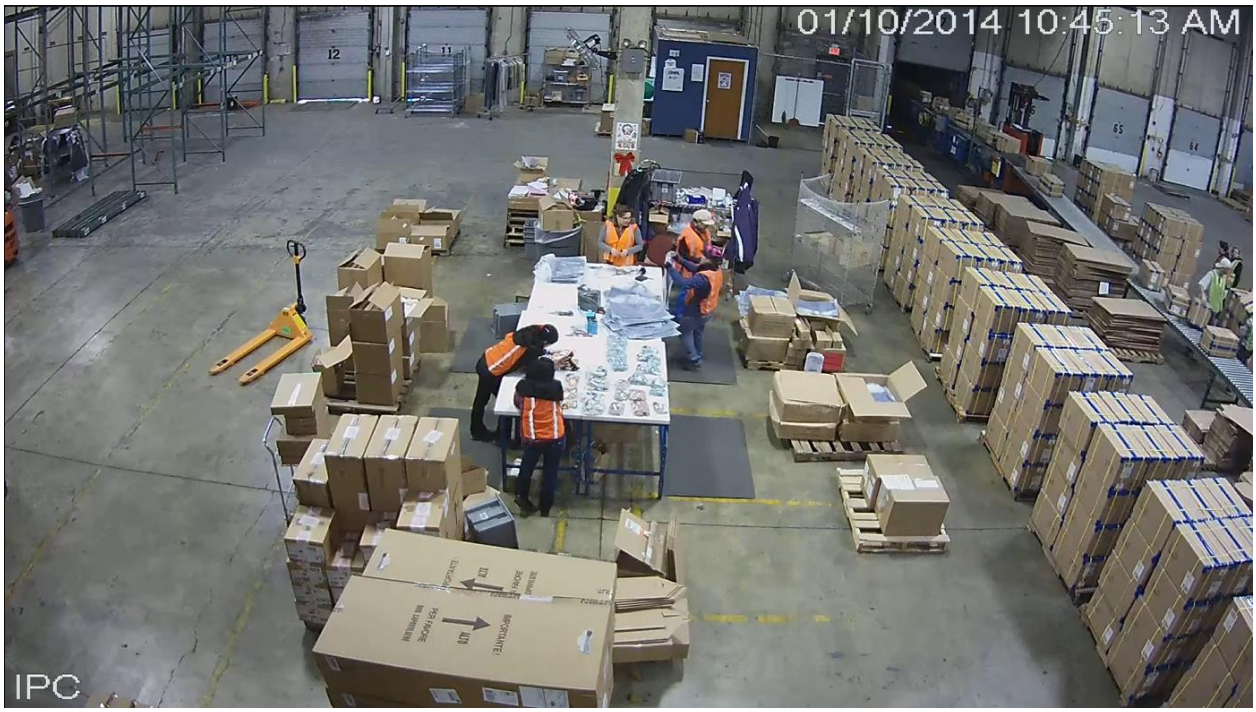


Рисунок 1.4 – Системи камер спостереження на підприємстві

Обсяг і різноманітність відеоданих створюють значні виклики. Відео з роздільною здатністю 1920×1080 пікселів і частотою 30 кадрів за секунду генерує великий потік інформації, що потребує ефективних алгоритмів обробки. Додатковими ускладненнями є низька якість зображення, мінливе освітлення (ніч, туман, дощ), а також потреба в розпізнаванні об’єктів у реальному часі. Це підкреслює необхідність використання передових технологій, зокрема глибоких нейронних мереж, які здатні адаптуватися до складних умов і забезпечувати високу точність і швидкість обробки.

За допомогою проведеного аналіза різних галузей, можна скласти таблицю пов’язаних сфер дослідження, особливостей середовищ та вимог до системи (табл. 1.1).

Таблиця 1.1 – Характеристики відеопотоку в різних сферах

Сфера	Джерело відеопотоку	Типи об’єктів	Особливості середовища	Вимоги до системи
Безпека	Камери відеоспостереження	Люди, транспорт, підозрілі предмети	Низьке освітлення, перекриття, шум	Реальний час, висока точність
Транспорт	Камери автомобілів, дорожні камери	Пішоходи, транспорт, знаки, перешкоди	Погодні умови, швидкий рух	Швидкість, надійність
Медицина	Ендоскопи, хірургічні камери	Тканини, органи, патології	Обмежена видимість, малий масштаб	Точність, чутливість
Промисловість	Камери на конвеєрах, роботах	Вироби, дефекти, обладнання	Висока швидкість, однотипність	Автоматизація, стабільність

1.2 Обґрунтування актуальності розробки

Актуальність розробки систем розпізнавання об’єктів у відеопотоці зумовлена зростаючою потребою в автоматизації аналізу відеоданих у різних сферах. У сучасному світі, де обсяг візуальної інформації стрімко зростає, ручна обробка стає неефективною через високу трудомісткість і схильність до помилок. Системи комп’ютерного зору, засновані на глибоких нейронних мережах, дозволяють автоматизувати виявлення, класифікацію та трекінг об’єктів, забезпечуючи швидкість, точність і надійність.

У сфері безпеки такі системи необхідні для моніторингу громадських місць, виявлення підозрілих об’єктів чи поведінки, а також запобігання злочинам і терористичним загрозам. У транспортній галузі вони є основою для автономних автомобілів, розпізнаючи пішоходів, транспортні засоби, дорожні знаки та перешкоди, що підвищує безпеку руху. У медицині технології розпізнавання об’єктів у відеопотоці застосовуються для аналізу ендоскопічних відео, виявлення патологій і підтримки хірургічних процедур. Промисловість використовує ці системи для автоматизованого контролю якості, моніторингу обладнання та оптимізації виробничих процесів (рис. 1.5).

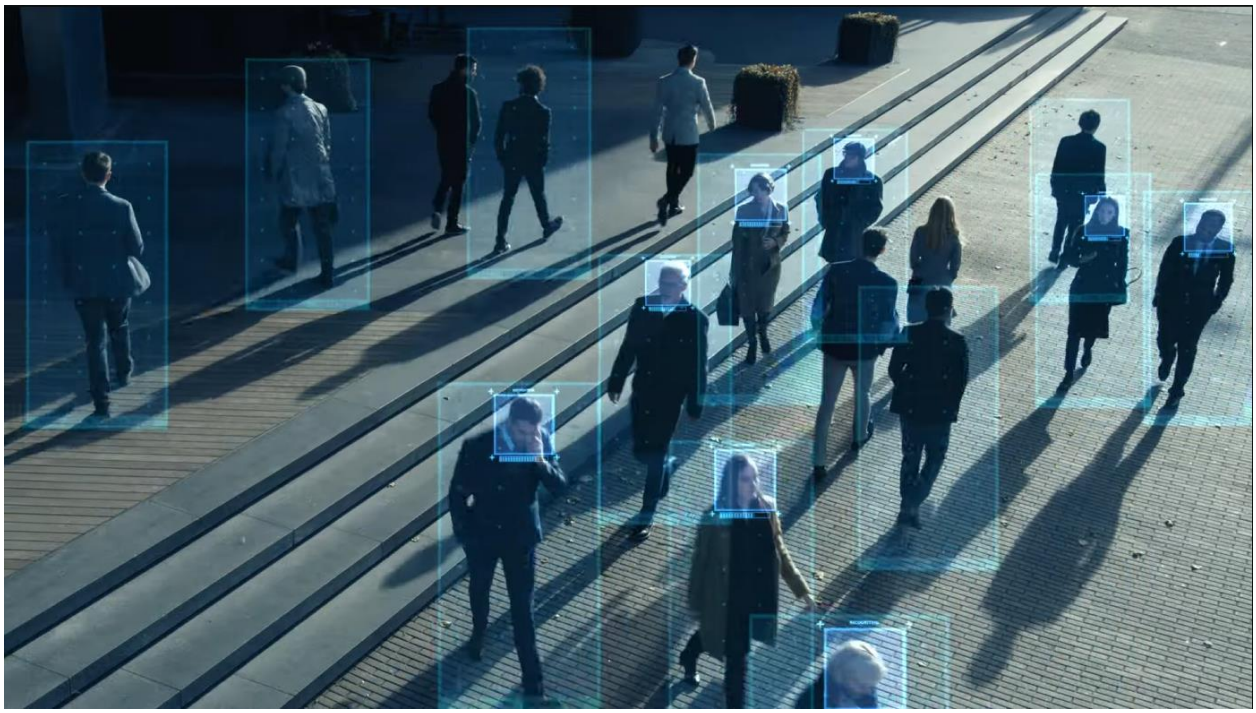


Рисунок 1.5 – Приклад застосування системи розпізнавання людей у відеопотоці

Розвиток глибоких нейронних мереж, таких як YOLO, відкрив нові можливості для обробки відеопотоку в реальному часі. Ці технології забезпечують високу точність і швидкість, що робить їх незамінними для динамічних середовищ. Актуальність розробки також підкреслюється глобальними викликами, такими як урбанізація, зростання населення та потреба в ефективному управлінні ресурсами. Система, що поєднує обробку відеофайлів і потоку з камери, гнучке збереження результатів і зручний графічний інтерфейс, має значний потенціал для практичного застосування, що обґрунтовує необхідність її створення.

1.3 Аналіз аналогів

Сучасні системи розпізнавання об’єктів у відеопотоці представлені численними рішеннями, що базуються на глибоких нейронних мережах. Одним із лідерів є модель YOLOv8, розроблена компанією Ultralytics [13].

Вона підтримує детекцію, класифікацію та трекінг об’єктів у реальному часі, вирізняючись високою швидкістю (до 100 кадрів за секунду на GPU) і точністю. YOLOv8 інтегрується в системи безпеки, такі як Milestone XProtect, де розпізнає людей, транспорт і підозрілі предмети. Її переваги – адаптивність, підтримка трекерів (наприклад, ByteTrack) і простота налаштування. Недоліки включають залежність від потужного апаратного забезпечення та зниження точності при поганій якості відео.

Інший аналог – Faster R-CNN, який застосовується в промислових системах, наприклад, Cognex VisionPro [2]. Ця модель використовує двоступеневу архітектуру: спочатку генерує регіональні пропозиції, а потім класифікує об’єкти, що забезпечує високу точність. Однак вона поступається YOLOv8 у швидкості, що обмежує її використання в реальному часі. SSD (Single Shot MultiBox Detector) [3], реалізований у платформах типу OpenVINO від Intel, є компромісом між швидкістю та точністю, але менш ефективний у складних сценаріях із перекриттям об’єктів [4].

Комерційні рішення, такі як Hikvision DeepinView, поєднують камери з нейронними мережами, пропонуючи готові продукти для безпеки та моніторингу [5]. Їхні переваги – простота впровадження та надійність, але недоліки включають високу вартість і обмежений SDG (Single shot detector) для детекції. DeepStream від NVIDIA забезпечує гнучкість і підтримку різних моделей, але потребує складного налаштування та кваліфікації користувача [6].

Проведений аналіз наявних аналогів сучасних систем дозволяє систематизувати ключові елементи. Кожна з розглянутих систем має свої переваги та недоліки. Для визначення оптимального варіанту була створена таблиця порівняльного аналізу систем розпізнавання об’єктів (табл. 1.2).

Таблиця 1.2 – Порівняльний аналіз аналогів систем розпізнавання об’єктів

Модель/Система	Сфера застосування	Переваги	Недоліки	Особливості
YOLOv8	Безпека, транспорт	Висока швидкість, точність, підтримка реального часу	Потреба в GPU, чутливість до якості відео	Трекінг (ByteTrack), реальний час
Faster R-CNN	Промисловість, медицина	Висока точність, детальний аналіз	Низька швидкість, складність	Двоступенева архітектура
SSD	Безпека, транспорт	Компроміс швидкості й точності	Менша точність у складних сценах	Одноступеневий підхід
Hikvision DeepinView	Безпека, моніторинг	Простота впровадження, надійність	Висока вартість, фіксовані класи	Інтеграція з камерами
DeepStream (NVIDIA)	Безпека, транспорт, промисловість	Гнучкість, підтримка моделей	Складне налаштування, потреба в кваліфікації	Хмарна обробка, масштабованість

1.4 Наявні та можливі проблеми

Незважаючи на прогрес, системи розпізнавання об’єктів у відеопотоці стикаються з низкою проблем. Перша проблема – висока обчислювальна складність глибоких нейронних мереж, що вимагає потужного апаратного забезпечення, такого як графічні процесори (GPU). Це ускладнює впровадження систем на малопотужних пристроях, таких як портативні камери чи вбудовані модулі. Друга проблема – зниження точності в умовах поганої видимості, зокрема при низькому освітленні, тумані, дощі чи перекритті об’єктів. Третя проблема – затримки під час обробки відеопотоку в реальному часі, що критично для застосувань, таких як автономне керування чи моніторинг безпеки.

Іншою складністю є брак уніфікованих підходів до збереження та аналізу результатів розпізнавання. Координати об’єктів, графіки кількості та теплові карти часто зберігаються в різних форматах, що ускладнює їх інтеграцію в інші системи. Можливі проблеми включають етичні та правові аспекти, зокрема захист даних і конфіденційність, особливо в системах відеоспостереження. Крім того, недостатня адаптивність моделей до нестандартних об’єктів чи сценаріїв може знижувати їхню ефективність.

На основі вищезгаданих проблем, було створено таблицю наявних та можливих проблем для визначення напрямку в якому слід рухатися для того, щоб уникнути найбільшої кількості проблем та вигадати рішення до них (табл. 1.3).

Таблиця 1.3 – Наявні та можливі проблеми систем розпізнавання об’єктів

Проблема	Опис	Наслідки	Можливі рішення
Обчислювальна складність	Високі вимоги до GPU, процесорів	Обмеження для малопотужних пристроїв	Оптимізація моделей, обмеження кількості кадрів
Погана видимість	Низьке освітлення, дощ, туман, перекриття	Зниження точності розпізнавання	Покращення фільтрації, адаптивні моделі
Затримки в реальному часі	Повільна обробка відеопотоку	Неможливість оперативного реагування	Прискорення алгоритмів, легші моделі
Збереження даних	Різні формати координат, графіків	Складність аналізу та інтеграції	Стандартизація форматів, уніфікація
Етичні аспекти	Конфіденційність, захист даних	Юридичні та соціальні обмеження	Шифрування, регулювання використання

Можливі рішення включають оптимізацію алгоритмів, використання хмарних обчислень, розробку легших моделей для малопотужних пристроїв, покращення фільтрації даних для умов поганої видимості та стандартизацію форматів зберігання результатів. Ці підходи потребують подальших досліджень і розробок.

Висновки до розділу

У розділі розглянуто предметне середовище систем розпізнавання об’єктів у відеопотоці, обґрунтовано актуальність розробки, проаналізовано аналоги та виявлено наявні й можливі проблеми. Встановлено, що відеопотік є складним і динамічним джерелом даних, застосовним у безпеці, транспорті, медицині та промисловості. Актуальність розробки підтверджена потребою в автоматизації, швидкості та точності аналізу. Аналіз аналогів показав переваги YOLOv8 у швидкості й адаптивності, але також підкреслив обмеження даної моделей. Виявлені проблеми – обчислювальна складність, погана видимість, затримки – потребують оптимізації. Розробка системи на базі YOLOv8 із графічним інтерфейсом є перспективною.

РОЗДІЛ 2
ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз предметної області

Аналіз предметної області є важливим етапом розробки системи розпізнавання об’єктів у відеопотоці, оскільки дозволяє визначити ключові характеристики, виклики та вимоги до інформаційного та математичного забезпечення. Предметна область охоплює обробку відеоданих, отриманих із різних джерел, і застосування глибоких нейронних мереж для виявлення та трекінгу об’єктів. У цьому підрозділі розглядаються особливості відеопотоку як джерела даних, класи об’єктів, які необхідно розпізнавати, основні виклики обробки та роль сучасних технологій, зокрема моделі YOLOv8, у розв’язанні цих завдань. Такий аналіз створює підґрунтя для концептуального проєктування та реалізації ефективної системи.

2.1.1 Характеристики відеопотоку як джерела даних

Системи розпізнавання об’єктів у відеопотоці базуються на обробці візуальної інформації, отриманої з різних джерел. Відеопотік являє собою послідовність кадрів, що надходять із камер відеоспостереження, вебкамер, автомобільних відеореєстраторів чи спеціалізованих пристроїв, таких як ендоскопи в медицині. Кожен кадр є двовимірним зображенням, що складається з пікселів, а їхня послідовність відображає динаміку об’єктів у часі. Основні технічні параметри відеопотоку включають роздільну здатність, частоту кадрів і формати. Роздільна здатність, наприклад, 1920×1080 (1080p), визначає деталізацію зображення. Частота кадрів (FPS, frames per second) варіюється від 30 до 60 кадрів за секунду, що впливає на плавність відображення руху. Поширені формати, такі як MP4 і AVI, забезпечують стиснення даних для ефективного зберігання та передачі [7].

Особливістю відеопотоку є значний обсяг даних. Наприклад, відео з роздільною здатністю 1920×1080 і частотою 30 кадрів за секунду генерує мільйони пікселів для обробки щосекунди, що створює високе навантаження на обчислювальні ресурси. Динамічність об’єктів у кадрі, їхній рух і зміна положення ускладнюють аналіз. Крім того, якість відеопотоку залежить від апаратного забезпечення: камери з низькою роздільною здатністю чи обмеженим кутом огляду можуть знижувати ефективність розпізнавання. Таким чином, відеопотік є складним об’єктом для обробки, що вимагає потужних алгоритмів і оптимізованих систем (рис. 2.1) [9].

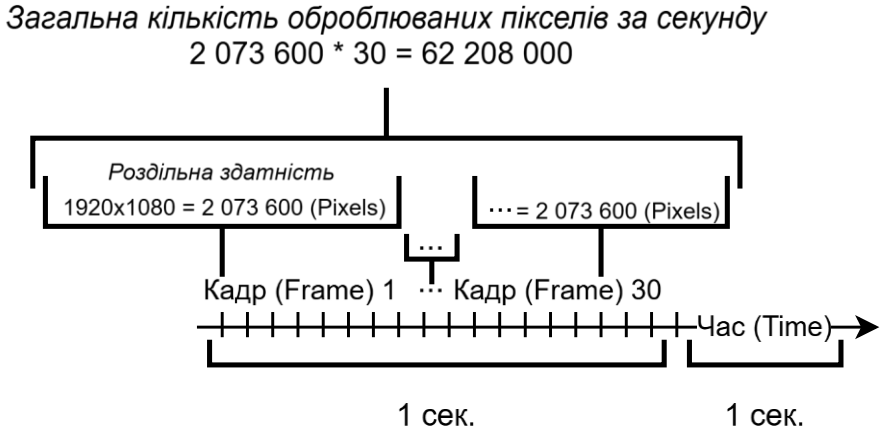


Рисунок 2.1 – Структура відеопотоку як джерела даних

2.1.2 Класи об’єктів для розпізнавання

Ефективність систем розпізнавання об’єктів залежить від чіткого визначення класів об’єктів, які планується виявляти в процесі розробки. У контексті підготовки до створення системи основну увагу приділено класам, які є релевантними для широкого спектра застосувань: люди та транспортні засоби, зокрема машини, мотоцикли, автобуси та вантажівки. Ці класи обрано через їхню значущість у різних сферах. У безпеці розпізнавання людей дасть змогу виявляти підозрілу поведінку чи присутність у заборонених зонах. У транспортній галузі ідентифікація машин, мотоциклів, автобусів і вантажівок

буде необхідною для управління дорожнім рухом, аналізу аварійності та підтримки автономних транспортних засобів.

Виклики розпізнавання пов’язані з варіативністю об’єктів. Люди можуть мати різний одяг, пози, розміри, а також частково перекриватися іншими об’єктами чи фоном. Транспортні засоби відрізняються за моделями, кольорами, кутами огляду та умовами руху. Наприклад, машина, що рухається на високій швидкості, або вантажівка, знята під незвичним ракурсом, ускладнюватимуть точну класифікацію. Вибір цих класів для майбутньої системи обґрунтований їхньою поширеністю та практичною цінністю, а також можливістю моделі YOLOv8 ефективно їх обробляти (табл. 2.1).

Таблиця 2.1 – Класи об’єктів для розпізнавання

Клас	Сфера застосування	Приклади	Виклики
Люди	Безпека, моніторинг громадських місць	Пішоходи, відвідувачі, персонал	Різноманітність одягу, пози, перекриття, зміна освітлення
Машини	Транспорт, дорожній рух, автономне керування	Легкові автомобілі, позашляховики	Різні моделі, кольори, швидкий рух, незвичні ракурси
Мотоцикли	Транспорт, дорожній рух	Спортивні мотоцикли, скутери	Малий розмір, висока швидкість, часткове перекриття
Автобуси	Транспорт, громадський транспорт	Міські автобуси, туристичні автобуси	Великий розмір, зміна орієнтації, погодні умови
Вантажівки	Транспорт, логістика	Вантажні автомобілі, фургони	Різноманітність розмірів, складні ракурси, низька видимість

2.1.3 Виклики обробки відеопотоку

Обробка відеопотоку для розпізнавання об’єктів супроводжується численними викликами, зумовленими природою даних і умовами середовища. Перший виклик – динамічність: об’єкти в кадрі рухаються, змінюють положення, масштаб і орієнтацію. Наприклад, людина, що біжить, або автомобіль, що швидко проїжджає, потребують швидкої та точної детекції (рис. 2.2).

Рисунок 2.2 – Приклад виклику у вигляді динамічності

Другий виклик – умови видимості. Низьке освітлення (нічний час), погодні фактори (дощ, туман, сніг) і перекриття об’єктів фоном чи іншими елементами знижують якість зображення та точність розпізнавання (рис. 2.3).

Рисунок 2.3 – Приклад виклику у вигляді умови видимості

Третій виклик – обчислювальна складність. Великий обсяг відеоданих, особливо при високій роздільній здатності та частоті кадрів, вимагає значних ресурсів для обробки в реальному часі. Затримки в аналізі можуть бути критичними, наприклад, для систем безпеки чи автономного транспорту, де потрібне миттєве реагування. Крім того, шум у відеопотоці, спричинений дефектами камер чи зовнішніми перешкодами, ускладнює виділення об’єктів. Ці виклики підкреслюють потребу в адаптивних і потужних алгоритмах для забезпечення ефективності системи (рис. 2.4).

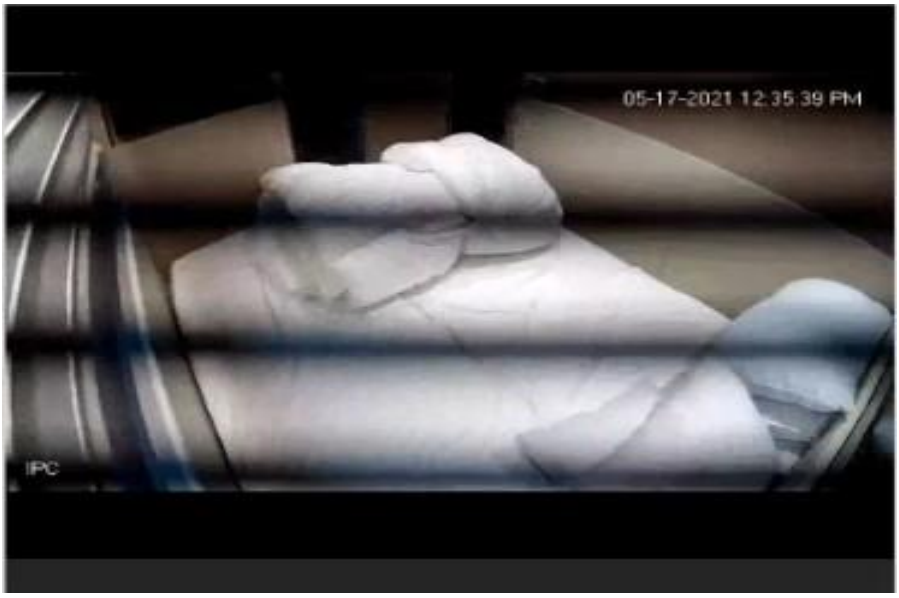


Рисунок 2.4 – Приклад виклику у вигляді дефекта камери

2.1.4 Вхідні та вихідні дані

Для розробки системи розпізнавання об’єктів у відеопотоці необхідно чітко визначити вхідні та вихідні дані, які будуть оброблятися на етапі реалізації. Вхідні дані системи включатимуть відеопотоки, отримані з двох джерел, що забезпечить гнучкість у застосуванні:

– Відеофайли у форматах MP4 або AVI, які містять попередньо записані відеодані, отримані, наприклад, із камер відеоспостереження, автомобільних відеореєстраторів або інших джерел. Параметри таких файлів включатимуть роздільну здатність (наприклад, 1280×720 або 1920×1080) і частоту кадрів (24–

60 FPS). Для оптимізації обробки планується масштабувати вхідні кадри до розміру 1280 пікселів, що забезпечить баланс між точністю детекції та обчислювальною ефективністю.

– Відеопотік у реальному часі, отриманий із вебкамери або іншої камери, підключеної до системи. Потік матиме аналогічні параметри роздільної здатності, а частота кадрів за замовчуванням складатиме 20 FPS, з можливістю оптимізації для обробки в реальному часі. Як і для відеофайлів, кадри масштабуватимуться до розміру 1280 пікселів.

Вхідні дані міститимуть зображення об’єктів класів, визначених раніше (люди, машини, і т.д.), які можуть з’являтися в різних умовах видимості (ніч, туман, перекриття) та динаміки (рух, зміна орієнтації). Для обробки вхідного відеопотоку планується використовувати налаштування, такі як поріг впевненості і максимальна кількість об’єктів у кадрі (до 1000), щоб оптимізувати продуктивність.

Вихідні дані системи будуть результатами обробки відеопотоку, які матимуть практичну цінність для користувачів у сферах безпеки, транспорту чи моніторингу. Планується генерувати такі типи вихідних даних:

1. Оброблене відео у форматі MP4 із нанесеними рамками навколо розпізнаних об’єктів. Рамки міститимуть мітки з назвою класу (наприклад, «person», «car»), унікальним ідентифікатором об’єкта і коефіцієнтом впевненості (0 – 1). Кольори відрізнятимуться залежно від класу (зелений для людей, синій для транспорту).

2. Файли з координатами об’єктів:

– Текстові файли (.txt) із записами у форматі «Кадр, Тип об’єкта, TrackID, Координати», де координати представлені як [x1, y1, x2, y2] (верхній лівий і нижній правий кути рамки).

– Табличні файли (.xlsx) з аналогічною структурою для зручного аналізу в табличних редакторах.

3. Графіки:

– Графік кількості об’єктів за кадрами, який відображатиме загальну кількість розпізнаних об’єктів у кожному кадрі.

– Графік кількості об’єктів за типами, який показуватиме розподіл об’єктів за класами (люди, машини тощо) у часі.

4. Теплова карта, яка відображатиме щільність об’єктів у кадрі за допомогою кольорової шкали (від холодних до гарячих тонів), де інтенсивність відповідатиме кількості виявлень у певній області.

Усі вихідні дані зберігатимуться у вказаній користувачем папці з унікальними іменами файлів, що включатимуть назву вхідного відео (для відеофайлів) або індекс запису (для камери). Наприклад, для відеофайлу «input.mp4» вихідні файли матимуть назви «input_out.mp4», «input_detections.txt», а для камери – «camera_out_0.mp4», «camera_detections_0.txt». Це забезпечить структуроване збереження та легкий доступ до результатів. Визначення вхідних і вихідних даних є основою для проєктування архітектури системи, вибору алгоритмів і розробки інтерфейсу користувача.

На основі проведеного аналізу вхідних та вихідних даних, було створено таблицю з вхідними/вихідними даними системи (табл. 2.2).

Таблиця 2.2 – Вхідні та вихідні дані системи

Тип даних	Джерело/формат	Опис	Призначення
1	2	3	4
Відеофайл	MP4, AVI	Попередньо записані відеодані з роздільною здатністю 1280×720 або 1920×1080, частотою кадрів 24–60 FPS. Кадри масштабується до 1280.	Вхідні дані для детекції та трекінгу об’єктів у записаних відео.
Потік із камери	Вебкамера	Відеопотік у реальному часі з роздільною здатністю до 1920×1080, частота кадрів 30 FPS за замовчуванням. Масштабується до 1280.	Вхідні дані для обробки в реальному часі, підходять для моніторингу.

Продовження таблиці 2.2

1	2	3	4
Оброблене відео	MP4	Відео з нанесеними рамками, мітками класу («person», «car» тощо), TrackID і коефіцієнтом впевненості. Кольори: зелений (люди), синій (транспорт).	Візуалізація результатів розпізнавання для аналізу користувачем.
Координати	Текстовий файл (.txt)	Записи у форматі: «Кадр, Тип об’єкта, TrackID, Координати [x1, y1, x2, y2]» для кожного об’єкта.	Збереження координат для подальшого аналізу чи інтеграції з іншими системами.
Координати	Табличний файл (.xlsx)	Таблиця з колонками: «Кадр, Тип об’єкта, TrackID, Координати [x1, y1, x2, y2]» для кожного об’єкта.	Зручний формат для аналізу в табличних редакторах.
Графіки (загальна к-сть)	Зображення (.png)	Графік кількості розпізнаних об’єктів за кадрами.	Аналіз динаміки кількості об’єктів у відеопотоці.
Графіки (за типами)	Зображення (.png)	Графік кількості об’єктів за класами (люди, машини тощо) у часі.	Аналіз розподілу об’єктів за типами для оцінки сценаріїв.
Теплова карта	Зображення (.png)	Зображення щільності об’єктів у кадрі з кольоровою шкалою.	Візуалізація зон активності об’єктів для моніторингу чи безпеки.

2.2 Проєктування системи

Концептуальне проєктування системи розпізнавання об’єктів у відеопотоці є ключовим етапом, який визначає її архітектуру, функціональні компоненти та принципи їхньої взаємодії. У цьому підрозділі буде розроблено концептуальну модель системи, описано її основні модулі, спроєктовано потоки даних і змодельовано поведінку за допомогою UML-діаграм. Це забезпечить чітке розуміння структури системи перед її реалізацією та дозволить узгодити вимоги предметної області, визначені в підрозділі 2.1, з технічними рішеннями.

2.2.1 Загальна концепція системи

Проєктування програмного забезпечення для системи розпізнавання об’єктів у відеопотоці передбачає створення гнучкої та модульної архітектури,

яка забезпечуватиме ефективне виконання основних завдань системи: обробку відеоданих, виявлення та відстеження об’єктів, збереження результатів та взаємодію з користувачем через графічний інтерфейс. В умовах зростаючих вимог до обчислювальної ефективності, адаптивності та зручності використання, обґрунтованим є підхід, що ґрунтується на розділенні функціональних компонентів відповідно до їхніх ролей у загальному процесі роботи.

На етапі проєктування планується визначити три основні функціональні компоненти системи: інтерфейс користувача, модуль обробки відеоданих та механізм збереження результатів. Такий розподіл дозволяє досягти високого рівня ізольованості відповідальності між частинами програми, що спрощує тестування, масштабування та подальший розвиток системи.

Інтерфейс користувача має забезпечувати інтуїтивну взаємодію зі сторони оператора: вибір відеофайлів або активацію режиму зчитування потоку з камери, налаштування параметрів обробки, вибір типів результатів, які потрібно зберігати, та відображення інформації про хід роботи системи. Передбачається створення окремих графічних оболонок для режимів офлайн-обробки та роботи в реальному часі, кожна з яких буде реалізована у вигляді окремого класу.

Модуль обробки відеоданих планується реалізувати як ядро системи, відповідальне за детекцію об’єктів у відеопотоці з використанням попередньо натренованої глибокої нейронної мережі YOLOv8. У рамках цього модуля також передбачається впровадження механізмів трекінгу (зокрема, алгоритму ByteTrack), побудови теплових карт присутності об’єктів та формування кадрів із візуалізацією результатів.

Компонент збереження результатів повинен забезпечити збереження структурованих даних у зручних для аналізу форматах, таких як .txt, .xlsx, .png, а також фінального обробленого відео у форматі .mp4. Для реалізації цього функціоналу доцільно інтегрувати спеціалізовані бібліотеки для роботи з табличними даними (наприклад, pandas) і для побудови графіків (matplotlib).

Запропонована архітектурна модель передбачає взаємодію компонентів на основі виклику методів, передачі аргументів через публічні атрибути класів, а також оновлення графічного інтерфейсу через механізм зворотнього виклику (callback). У подальших підпунктах буде детально розглянуто принципи взаємодії між окремими класами, їхню структуру та особливості функціонування в обох режимах роботи системи.

2.2.2 Класи проєктованої системи

У рамках проєктування програмного забезпечення для системи розпізнавання об’єктів у відеопотоці передбачається реалізація архітектури, заснованої на об’єктно-орієнтованому підході. Така модель дозволяє структурувати функціональність системи у вигляді взаємодіючих класів, кожен з яких виконує чітко визначені обов’язки. Принципи інкапсуляції, модульності та слабкого зв’язку між класами забезпечують гнучкість і масштабованість при реалізації та подальшому розвитку програмного продукту.

Згідно з проєктною структурою, основними класами системи мають стати:

– YOLOApp – клас, відповідальний за графічний інтерфейс у режимі обробки відеофайлів. Передбачається, що він забезпечуватиме користувачу можливість обрати вхідний відеофайл, вказати директорію збереження результатів, активувати/деактивувати окремі типи вихідних даних (координати, графіки, теплову карту), запускати процес обробки та спостерігати за його прогресом.

– CameraWindow – окремий інтерфейсний клас, який планується застосовувати для реалізації режиму обробки відеопотоку в реальному часі з підключеної камери. Його функціональність частково збігається з класом YOLOApp, проте доповнюється можливістю відображення живого відео на екрані, запуску та зупинки запису, а також обробки потоку кадрів у циклі.

– VideoProcessor – основний обчислювальний клас, що реалізує всі ключові етапи обробки відеоданих: зчитування кадрів, виконання детекції

об’єктів за допомогою моделі YOLOv8, реалізацію трекінгу, нанесення візуальних позначень, накопичення координат, побудову теплових карт, формування графіків та збереження отриманих результатів. Передбачається, що саме цей клас отримуватиме параметри від інтерфейсних модулів та повертатиме проміжні або фінальні дані для виводу.

Взаємодія між зазначеними класами реалізовуватиметься за принципом ініціалізації екземплярів та виклику публічних методів. Зокрема, YOLOApp і CameraWindow створюватимуть об’єкт класу VideoProcessor, передаючи йому набір параметрів, після чого відповідні методи виконуватимуть операції з обробки даних. Зворотний зв’язок із користувачем (зокрема, оновлення прогрес-бару) передбачається реалізовувати через механізм callback-функцій, що передаються від інтерфейсного модуля до обчислювального.

На основі цього, було наведено UML-діаграму класів, яка ілюструє структуру взаємозв’язків між основними компонентами системи, їх атрибути та ключові методи. Діаграма слугує візуальним доповненням до описаної логіки взаємодії та дає цілісне уявлення про архітектурну модель системи (додаток А).

Таким чином, обрана структура класів забезпечує відповідність вимогам до системи, розділення відповідальності та передумови для ефективної реалізації функціональності в обох режимах роботи – з відеофайлом та камерою.

2.2.3 Логіка роботи системи

При проєктуванні системи розпізнавання об'єктів у відеопотоці важливим аспектом є визначення послідовності взаємодії компонентів у двох основних режимах роботи – обробки попередньо записаного відеофайлу та роботи з відеопотоком у реальному часі. Незважаючи на певні відмінності у специфіці джерел відеоданих, загальна логіка роботи системи у цих режимах має низку спільних кроків, що забезпечують уніфікацію підходів до їх реалізації.

На початковому етапі роботи в обох режимах користувач здійснює налаштування параметрів через відповідний графічний інтерфейс. У випадку роботи з відеофайлами це включає вибір самого файлу, визначення папки для збереження результатів і активацію необхідних типів вихідних даних. В режимі реального часу додатково передбачається вибір папки збереження, а також активація та припинення запису відеопотоку безпосередньо під час роботи.

Після налаштування користувачем усіх параметрів і запуску процесу, графічний інтерфейс здійснює передачу введених даних та параметрів до модуля обробки (VideoProcessor), який стає центральною складовою логіки обох режимів.

Під час роботи із записаним відеофайлом модуль обробки відкриває його за допомогою функціональних можливостей бібліотеки OpenCV [11], після чого розпочинається послідовна обробка кадрів. У режимі реального часу аналогічна процедура виконується зі зчитуванням потоку з камери. В обох випадках кожен отриманий кадр передається до моделі YOLOv8, яка забезпечує детекцію об'єктів, визначаючи їх типи, координати, а також рівні впевненості класифікації. Наступним етапом є застосування алгоритму ByteTrack, що забезпечує трекінг об'єктів між сусідніми кадрами [12].

Після детекції та трекінгу система формує візуалізацію результатів, яка включає нанесення прямокутників навколо знайдених об'єктів, зазначення класу, ідентифікатора та впевненості. Паралельно відбувається накопичення координат для побудови теплової карти, яка відображатиме густину знаходження об'єктів у кадрі за весь час спостереження. Отримані кадри з візуальними позначеннями послідовно записуються у новий відеофайл (у випадку роботи з попередньо записаним відео), або здійснюється потокова візуалізація у вікні програми (при роботі в реальному часі).

У рамках роботи з камерою додатковою важливою характеристикою є вимога до швидкості обробки кадрів. Це зумовлює проєктні рішення щодо оптимізації роботи алгоритмів детекції та трекінгу для забезпечення

мінімальних затримок між отриманням кадру з камери і виведенням результатів на екран.

В обох сценаріях після завершення обробки кадрів здійснюється формування файлів із координатами виявлених об'єктів у форматах .txt та .xlsx, генерація графіків, які відображають кількість об'єктів за типами й у часі, а також побудова остаточної теплової карти у форматі .png.

Процес взаємодії всіх компонентів у межах описаних режимів роботи ілюструє UML-діаграма послідовності, представлена в додатку Б, яка демонструє єдність і послідовність кроків виконання завдань системою.

Таким чином, запропонована логіка роботи системи забезпечує реалізацію єдиного механізму обробки відеоданих, що враховує специфіку різних джерел відеопотоків і створює передумови для ефективної реалізації у проєктованому програмному забезпеченні.

2.3 Математичне та алгоритмічне забезпечення

2.3.1 Математичні моделі

Розробка системи розпізнавання об'єктів у відеопотоці базується на кількох фундаментальних математичних моделях, що дозволяють виконати завдання детекції, відстеження об’єктів та формування теплових карт щільності.

Для завдання детекції об'єктів у кадрі застосовується модель глибокої нейронної мережі YOLOv8, яка працює на основі згорткових шарів та забезпечує ефективну локалізацію об'єктів у зображенні. Основою цієї моделі є розрахунок коефіцієнта перетину рамок, що відомий як показник IoU (Intersection over Union). IoU є критично важливим показником, що визначає ступінь перекриття рамок, які утворюються навколо виявлених об'єктів:

$$IoU = \frac{Area(B_1 \cap B_2)}{Area(B_1 \cup B_2)}, \quad (2.1)$$

де *B1* та *B2* – відповідно перша та друга рамки, які порівнюються; *Area(B1∩B2)* – площа їхнього перетину; *Area(B1∪B2)* – площа їхнього об'єднання.

Окрім того, для роботи мережі YOLOv8 застосовується перетворення координат рамок із формату центру з шириною і висотою до формату двох кутів, що описують прямокутник. Цей процес відбувається за такою формулою:

$$(x_{center}, y_{center}, width, height) \rightarrow (x_1, y_1, x_2, y_2,), \quad (2.2)$$

де (x_{center}, y_{center}) – координати центру рамки; *width, height* – її ширина та висота; (x_1,y_1), (x_2,y_2) – координати верхнього лівого та нижнього правого кутів відповідно.

Наступною математичною моделлю, що застосовується у системі, є алгоритм трекінгу ByteTrack [12], який базується на класичному угорському алгоритмі (Hungarian algorithm). Цей алгоритм дозволяє ефективно встановлювати відповідності між детектованими об’єктами на послідовних кадрах. Розв’язок цієї задачі формалізується таким чином:

$$\text{minimize} \sum_{i=1}^{n} \sum_{j=1}^{m} C_{ij} X_{ij}, \quad (2.3)$$

за умовами:

$$\sum_{j=1}^{m} X_{ij} = 1(i = 1,2, \dots, n), \sum_{i=1}^{n} X_{ij} = 1(j = 1,2, \dots, m), \quad (2.4)$$

де C_{ij} – матриця вартості, сформована на основі значень IoU між рамками, а X_{ij} – матриця призначення, яка вказує на відповідність об'єктів між кадрами.

Для формування теплових карт щільності об'єктів застосовується модель, що базується на накопиченні координат знайдених об'єктів. Значення щільності у кожному пікселі розраховується за формулою:

$$H(x,y) = \sum_{f=1}^{F} \sum_{i=1}^{N_f} 1\left[(x,y) \in bbox_{i,f}\right] \quad (2.5)$$

де $H(x,y)$ – значення щільності у пікселі з координатами (x,y), F – загальна кількість кадрів у відео, N_f – кількість об'єктів у кадрі з номером f, а $bbox_{i,f}$ – рамка навколо i-го об'єкта на кадрі номер f.

Таким чином, описані математичні моделі є основою для розв’язання всіх поставлених задач системи.

2.3.2 Алгоритми реалізації

Для ефективної практичної реалізації математичних моделей, описаних вище, необхідно чітко визначити алгоритмічні рішення, що забезпечуватимуть узгоджену взаємодію компонентів системи. Алгоритмічне забезпечення передбачає три взаємопов’язані рівні: рівень інтерфейсу користувача, рівень обчислювальних модулів та загальний алгоритм обробки відеоданих.

Алгоритм роботи графічного інтерфейсу реалізує початковий етап взаємодії системи з користувачем, що включає налаштування параметрів обробки, запуск процесів та відображення поточного стану системи. Інтерфейс має працювати у двох режимах: обробка відеофайлів та робота з камерою в реальному часі.

У першому режимі користувач вибирає файл, директорію збереження, типи даних, що генеруються, і запускає процес обробки. У другому режимі користувач визначає лише директорію збереження та здійснює керування записом потоку з камери. Алгоритм взаємодії інтерфейсу з користувачем можна представити у вигляді блок-схеми, яка містить етапи налаштування, запуску процесів та виведення результатів (рис. 2.5).

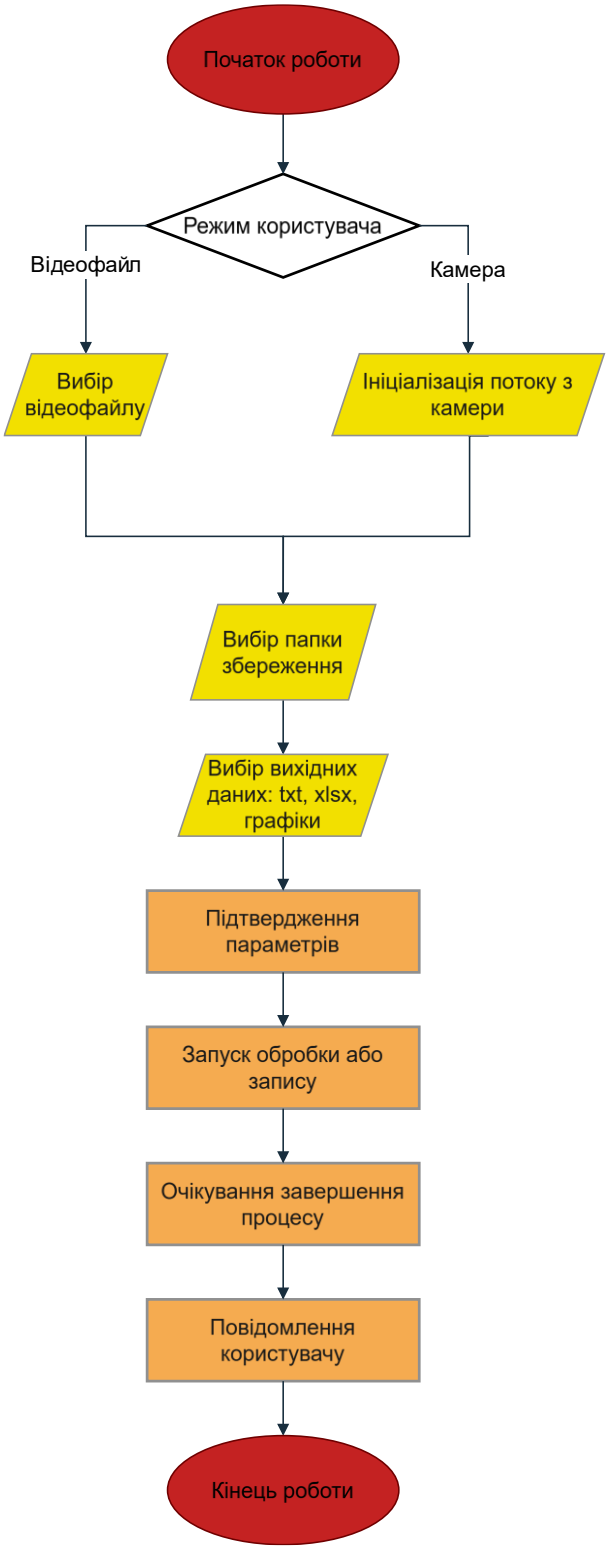


Рисунок 2.5 – Алгоритм роботи графічного інтерфейсу

Після того, як користувач ініціює процес обробки, керування передається на рівень обчислювальних модулів системи. Цей рівень

представлений центральним класом VideoProcessor, що реалізує алгоритми детекції, трекінгу та візуалізації об’єктів.

Алгоритм роботи обчислювальних модулів включає в себе послідовність процедур, необхідних для реалізації математичних моделей, описаних у пункті 2.3.1 (рис. 2.6).

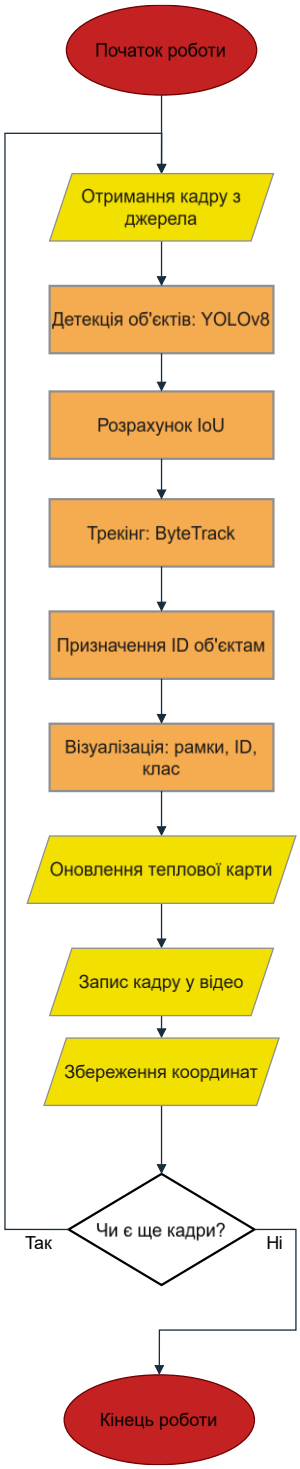


Рисунок 2.6 – Алгоритм роботи обчислювальних модулів системи

Основними етапами цього алгоритму є:

– отримання кадру з джерела відеопотоку;

– виконання детекції за допомогою YOLOv8 (формула 2.1);

– трекінг детектованих об’єктів (формули 2.3 та 2.4);

– візуалізація знайдених об’єктів (формула 2.2);

– накопичення інформації для теплової карти (застосування формули накопичення щільності (2.5));

– передача результатів до механізму збереження.

Для забезпечення цілісного функціонування системи необхідний загальний алгоритм, який координує роботу інтерфейсу та обчислювальних модулів, враховуючи особливості обох режимів роботи системи (рис. 2.7).

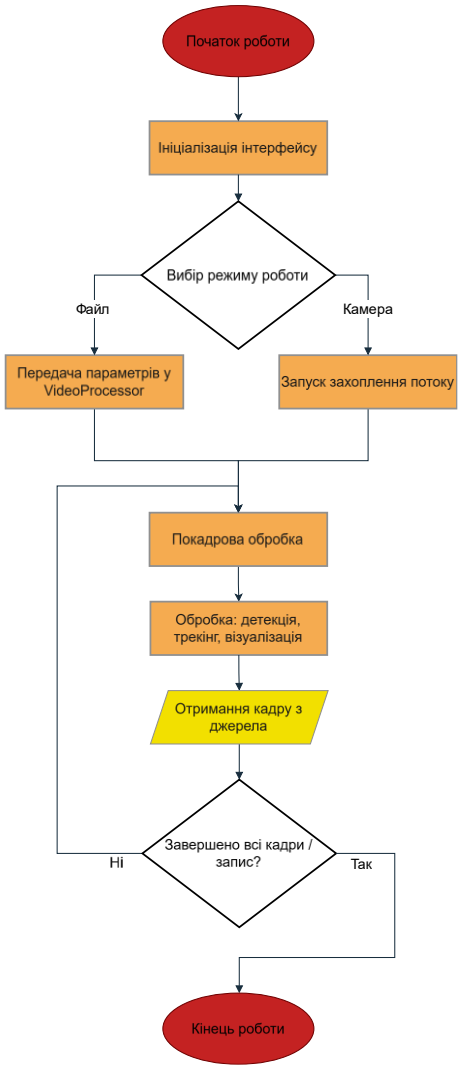


Рисунок 2.7 – Загальний алгоритм роботи системи

Загальний алгоритм роботи системи складається з таких етапів:

– ініціалізація системи та вибір режиму роботи через графічний інтерфейс;

– запуск обробки відповідно до обраного режиму (файл/камера);

– виклик алгоритму роботи обчислювальних модулів для кожного отриманого кадру;

– періодичне оновлення графічного інтерфейсу для відображення стану процесу;

– формування та збереження результатів після завершення роботи (координати, графіки, теплова карта).

Для повноти картини також доречно описати алгоритм роботи модуля формування теплових карт, який працює на основі накопичення координат виявлених об’єктів (рис. 2.8).

Рисунок 2.8 – Алгоритм формування теплової карти щільності об’єктів

Він включає такі кроки:

– створення початкової матриці щільності (нульова матриця);

– накопичення щільності за координатами bounding boxes (формула 2.5);

– нормалізація матриці щільності після завершення всіх кадрів;

– формування візуального представлення теплової карти за допомогою колірної шкали.

Таким чином, алгоритмічне забезпечення представлене комплексом узгоджених процедур, що забезпечують виконання всіх функцій проєктованої системи. Чітко визначені переходи між етапами обробки та відповідність використаних математичних моделей дозволяють забезпечити логічну та ефективну реалізацію запланованого програмного забезпечення.

Висновки до розділу

У розділі виконано аналіз інформаційної структури задачі, спроєктовано архітектуру системи та обґрунтовано вибір основних математичних моделей. Визначено ключові компоненти програмного забезпечення: інтерфейс користувача, модуль обробки відео та підсистема збереження результатів. Описано логіку взаємодії між модулями в обох режимах роботи системи. Наведено математичні моделі, зокрема функцію IoU, угорський алгоритм для трекінгу та модель побудови теплової карти. Розроблено алгоритми реалізації та подано блок-схеми основних процедур. Отримані результати є основою для подальшої реалізації програмної частини системи.

РОЗДІЛ 3
ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Засоби розробки

Розробка програмного забезпечення для системи розпізнавання об’єктів у відеопотоці здійснювалася з використанням мови програмування Python версії 3.12 [9], яка забезпечує високий рівень виразності, широке поширення бібліотек для роботи з нейронними мережами, комп’ютерним зором, візуалізацією та графічним інтерфейсом. У якості середовища розробки було обрано Visual Studio Code [10], що дозволяє зручно керувати проєктною структурою, інтегрувати віртуальне середовище Python, здійснювати відлагодження та контроль версій.

Архітектура програмного продукту реалізована у вигляді двох основних логічних модулів, розміщених у відповідних папках:

– Program – містить основний вихідний код програми: скрипти інтерфейсу, модуль обробки відео та головний виконавчий файл (main.py);

– Data – використовується для розміщення вхідних відеофайлів, збереження вихідних результатів (оброблені відео, координати, графіки, теплові карти).

Для реалізації глибокої детекції об’єктів використано модель YOLOv8 [1], яка інтегрується через бібліотеку ultralytics. Ця модель дозволяє виявляти об’єкти у відеопотоці з високою точністю та забезпечує роботу в реальному часі. Детекція виконується за допомогою попередньо навчених ваг без потреби локального тренування.

Бібліотека OpenCV [11] використовується для захоплення відео з камери, читання відеофайлів, обробки кадрів та збереження результатів у форматі .mp4. За допомогою ByteTrack [12], вбудованого в Ultralytics [13], реалізовано трекінг об’єктів між кадрами.

Графічний інтерфейс побудований з використанням Tkinter [14], що дозволяє створювати інтуїтивні вікна, кнопки, поля введення та інші віджети для взаємодії з користувачем. Для виводу зображень у форматі, сумісному з Tkinter, застосовується Pillow.

Для збереження координат виявлених об’єктів у табличному форматі використовується Pandas [17], яка забезпечує експорт даних у .xlsx. Для побудови графіків кількості об’єктів і теплових карт використано бібліотеку Matplotlib [15].

У табл. 3.1 наведено узагальнену інформацію про використані засоби розробки.

Таблиця 3.1 – Засоби розробки програмного забезпечення

Компонент	Засіб / Бібліотека	Призначення
Мова програмування	Python 3.12	Основна мова реалізації
Середовище розробки	Visual Studio Code	Створення та налагодження проєкту
Обробка відео	OpenCV	Захоплення, обробка, збереження відео
Детекція об’єктів	Ultralytics / YOLOv8	Виявлення об’єктів у кадрі
Трекінг	ByteTrack (Ultralytics)	Відстеження об’єктів між кадрами
Інтерфейс	Tkinter, Pillow	Графічне середовище користувача
Збереження координат	Pandas	Табличні дані у форматі .xlsx
Побудова графіків	Matplotlib	Графіки та теплові карти
Структура проєкту	Program / Data	Вихідний код / Вхідні та вихідні дані

Таким чином, використані програмні засоби забезпечують реалізацію всіх функціональних компонентів системи, адаптовані до завдань комп’ютерного зору та дають змогу ефективно поєднати алгоритмічну частину з користувацьким інтерфейсом.

3.2 Вимоги до технічного та програмного забезпечення

Для коректного функціонування програмного забезпечення системи розпізнавання об'єктів у відеопотоці користувач повинен мати комп’ютер із відповідною апаратною конфігурацією та встановленим необхідним програмним забезпеченням. Зважаючи на використання глибоких нейронних

мереж, обробки відеоданих і візуалізації, система має певні мінімальні та рекомендовані вимоги.

Програму можливо запускати як на комп'ютерах із графічними прискорювачами (GPU), так і без них, оскільки фреймворк torch [16], який використовується для обробки глибоких моделей, автоматично перемикається між CUDA та CPU. Проте відсутність GPU суттєво знижує продуктивність та швидкість обробки відео, тому у рекомендованій конфігурації передбачено наявність дискретної відеокарти.

На основі проведеного аналізу було складено таблицю 3.2, в якій подано порівняння мінімальних і рекомендованих вимог до апаратного забезпечення (табл. 3.2).

Таблиця 3.2 – Технічні вимоги до системи

Параметр	Мінімальні вимоги	Рекомендована конфігурація
Операційна система	Windows 10 / Linux	Windows 10 (64-bit)
Процесор	4 ядра / 8 потоків	16 ядер / 32 потоки
Оперативна пам'ять	8 ГБ	32 ГБ DDR5
Графічний процесор (GPU)	Відсутній (CPU-режим)	NVIDIA RTX 4070 або аналог (8 ГБ VRAM)
Камера	Необов'язкова	Вбудована або зовнішня вебкамера
Відеоінтерфейс	Стандартний	Роздільна здатність екрана 1080p
Вільне місце на диску	2–3 ГБ	5+ ГБ

Крім апаратної складової, система потребує встановлення відповідного програмного середовища. У таблиці 3.3 наведено перелік необхідного програмного забезпечення.

Таблиця 3.3 – Програмне забезпечення для запуску системи

Компонент	Параметри / Версія
Інтерпретатор Python	Python 3.12
Менеджер пакетів	pip
Обов'язкові бібліотеки	ultralytics, torch, opencv-python, pandas, matplotlib, numpy, openpyxl
Середовище розробки	Visual Studio Code (опціонально)

Програмне забезпечення не потребує сторонніх сервісів, драйверів чи підключення до мережі, за винятком етапу первинної інсталяції бібліотек. Робота з камерою передбачає наявність вбудованого або зовнішнього відеопристрою, який розпізнається бібліотекою OpenCV [11].

Таким чином, вимоги до апаратного та програмного забезпечення залишаються помірними для сучасних комп'ютерних систем і дають змогу використовувати програму як у локальному режимі, так і для подальшого перенесення на інші пристрої.

3.3 Опис програмної реалізації

3.3.1 Загальна структура програмного забезпечення

Програмне забезпечення системи розпізнавання об'єктів у відеопотоці реалізовано як консольно-графічний додаток із двома режимами роботи: обробка відеофайлів та обробка потоку з камери в реальному часі. Структура реалізації базується на модульному підході, що дозволяє забезпечити розмежування відповідальності між різними компонентами системи.

Уся програмна логіка організована у вигляді трьох основних модулів, які розміщено в окремих скриптах каталогу Program:

– main.py – головний модуль запуску програми, що ініціалізує відповідний режим роботи (з відеофайлом або камерою) та викликає відповідний графічний інтерфейс;

– gui.py – містить реалізацію графічного інтерфейсу користувача у вигляді двох класів: YOLOApp (режим відеофайлу) та CameraWindow (режим камери);

– video_processor.py – реалізує основну логіку обробки відеопотоку, включаючи детекцію об'єктів, трекінг, візуалізацію, збереження результатів і формування теплових карт.

Крім того, у каталозі Data зберігаються вхідні відеофайли, а також вихідні дані: оброблені відео, текстові й табличні координати, графіки та теплові карти.

Загальна логіка взаємодії компонентів проєкту полягає в тому, що користувач через інтерфейс (класи YOLOApp або CameraWindow) задає параметри обробки, які передаються до модуля VideoProcessor, що виконує основні обчислювальні операції. Після обробки результати повертаються інтерфейсному модулю та зберігаються в обраній директорії.

Всі ключові класи, їхні методи та зв’язки наведено у діаграмі класів (див. додаток А), а загальний сценарій роботи системи подано на UML-діаграмі послідовності (див. додаток Б).

Таким чином, реалізація системи ґрунтується на розділенні логіки за функціональними ознаками:

- main.py – керування режимами запуску;
- gui.py – взаємодія з користувачем;
- video_processor.py – обробка, аналіз і збереження відеоданих.

Це забезпечує гнучкість структури, полегшує відлагодження, підтримку та масштабування програмного продукту.

3.3.2 Модуль графічного інтерфейсу (gui.py)

Графічний інтерфейс користувача є ключовим компонентом, що забезпечує доступність, зручність і наочність використання системи. Реалізація інтерфейсу виконана за допомогою стандартної бібліотеки Tkinter, що входить до складу Python, та доповнена функціональністю бібліотеки Pillow для роботи з візуальним контентом. Модуль інтерфейсу зосереджено в окремому файлі gui.py, який містить два основних класи: YOLOApp та CameraWindow.

Клас YOLOApp відповідає за інтерфейс у режимі офлайн-обробки попередньо записаних відеофайлів. Після запуску користувач бачить вікно з полями вибору відеофайлу, директорії для збереження результатів, а також

перемикачами (чекбоксами) для вибору форматів вихідних даних: координати об’єктів у .txt та .xlsx, графіки та теплова карта.

Ключові методи класу:

– browse_video() – відкриває вікно вибору відеофайлу;

– browse_directory() – відкриває вікно вибору папки для збереження;

– process_video() – викликає обробку відео, ініціюючи створення об’єкта VideoProcessor та передаючи всі параметри;

– update_progress(progress) – оновлює прогрес-бар під час обробки;

– switch_to_camera() – перемикає користувача до вікна CameraWindow;

– on_closing() – обробляє подію закриття вікна, завершуючи роботу без помилок.

Інтерфейс також містить прогрес-бар для відображення стану обробки та повідомлення про завершення процесу. Усі елементи інтерфейсу прив’язані до змінних типу StringVar та BooleanVar, що забезпечує реактивність GUI на дії користувача.

Клас CameraWindow реалізує інтерфейс для роботи з вебкамерою в режимі реального часу. Його структура подібна до YOLOApp, однак він додатково містить полотно (Canvas) для виведення відеопотоку та кнопки для керування записом: початок, зупинка та повернення до режиму відеофайлу.

Ключові методи класу:

– start_recording() – ініціює запис і обробку відеопотоку;

– stop_recording() – припиняє обробку та зберігає результати;

– update_camera() – періодично оновлює зображення з камери у вікні;

– switch_to_video() – повертає користувача до YOLOApp;

– browse_directory() – вибір директорії для збереження файлів;

– on_closing() – обробляє закриття вікна.

Клас використовує змінні стану, такі як is_recording, для контролю логіки обробки в реальному часі, а також графічні елементи Label та PhotoImage для відображення відео.

Обидва класи реалізують взаємодію з модулем VideoProcessor через виклик відповідних методів (process_video, start_recording) і передають до нього параметри обробки, обрані користувачем. Зворотний зв’язок реалізовано через функції зворотного виклику (callback), що дозволяє оновлювати інтерфейс під час виконання обчислювальних операцій у фоновому потоці.

Таким чином, модуль gui.py забезпечує зручну взаємодію між користувачем та системою, дозволяючи легко налаштовувати параметри обробки, запускати алгоритми та переглядати результати. Роль інтерфейсу в системі є не лише сервісною, а інтегруючою, оскільки він координує обмін даними між усіма іншими модулями (з повною версією можна ознайомитись в додатку В).

3.3.3 Модуль обробки відео (video_processor.py)

Модуль video_processor.py реалізує основну функціональність системи розпізнавання об’єктів. Саме в ньому зосереджено всі обчислювальні процеси: від зчитування відеопотоку до формування графічних результатів. У модулі реалізовано клас VideoProcessor, який використовується як у режимі обробки відеофайлів, так і при роботі з потоком із камери. Цей клас взаємодіє з бібліотеками Ultralytics, OpenCV, Numpy, Pandas та Matplotlib.

3.3.3.1 Структура класу VideoProcessor

Об’єкти класу VideoProcessor ініціалізуються під час запуску обробки з GUI. До основних атрибутів класу належать:

– model – екземпляр моделі YOLOv8, завантаженої з ultralytics;

– cap, out – об’єкти для зчитування та запису відео (cv2.VideoCapture, cv2.VideoWriter);

– detections – список об’єктів, виявлених у процесі роботи;

– heatmap – матриця щільності, що накопичує координати для побудови теплової карти;

– is_recording, is_running – логічні прапори, що керують виконанням потокової обробки;

– transport_classes, class_names – словники з класами об’єктів і кольорами для візуалізації.

Ці атрибути використовуються під час обробки відео у відповідних методах. З модулем відповідного класа можна ознайомитись в додатку Г.

3.3.3.2 Метод process_video()

Цей метод використовується для обробки відеофайлів. Основна логіка полягає в такому:

1. Відкривається відеофайл за допомогою cv2.VideoCapture.

2. Кадри зчитуються в циклі до завершення відео.

3. Кожен кадр передається моделі YOLOv8 для детекції об’єктів:

– координати рамок та типи об’єктів зберігаються;

– розраховується IoU для фільтрації результатів (див. формулу 2.1).

4. Після детекції застосовується вбудований ByteTrack, що виконує трекінг між кадрами (з використанням угорського алгоритму, формули 2.3–2.4).

5. Візуалізація: на кадр накладаються прямокутники, класи, унікальні ID.

6. Для кожного кадру оновлюється матриця щільності heatmap, на основі координат рамок (див. формулу 2.5).

7. Оброблені кадри записуються до нового відеофайлу.

8. По завершенню:

– координати експортуються у .txt та .xlsx;

– генеруються графіки та теплова карта.

Цей метод приймає параметри, обрані користувачем, включно з режимами збереження результатів (типи даних, директорія, формат).

3.3.3.3 Метод process_camera_stream()

Цей метод активується при роботі з камерою. Його логіка аналогічна до process_video(), однак має кілька відмінностей:

– кадри зчитуються у реальному часі;

– використовується таймер для періодичного оновлення в GUI (update_camera() у CameraWindow);

– відео записується лише під час активного запису (натискання кнопки «Start»);

– всі інші етапи: детекція, трекінг, оновлення heatmap і збереження – реалізовано аналогічно.

3.3.3.4 Інші методи класу

Інші методи класів, що використовується лише для ініціалізації дій користувача:

- start_recording() – ініціалізує збереження потоку з камери;
- stop_recording() – зупиняє запис і викликає збереження результатів;
- stop_camera() – вивільняє ресурси OpenCV (звільнення cap, out).

3.3.3.5 Збереження результатів

Після завершення обробки система автоматично виконує:

- експорт координат в .txt та .xlsx (через pandas.DataFrame);
- побудову графіка кількості об’єктів за кадрами;
- побудову графіка розподілу об’єктів за класами;
- генерацію теплової карти (matplotlib.imshow із колірною шкалою hot).

Усі файли зберігаються у вибраній директорії з унікальними назвами, пов’язаними з початковим відео або серією запису з камери.

Таким чином, модуль video_processor.py виступає ядром системи. Його реалізація забезпечує повноцінну обробку відеоданих у двох режимах, поєднуючи можливості глибокого навчання, трекінгу об’єктів та візуальної аналітики. Взаємодія цього модуля з GUI здійснюється через параметри та механізм зворотних викликів, що дозволяє підтримувати повноцінну інтеграцію між інтерфейсом та обчислювальним ядром програми.

3.3.4 Головний модуль запуску (main.py)

Файл main.py є точкою входу до програми та виконує функцію запуску графічного інтерфейсу користувача залежно від вибраного режиму роботи. Його структура є відносно простою, однак відіграє критично важливу роль у логіці функціонування всього програмного забезпечення, оскільки координує

ініціалізацію класів, які реалізовано в модулі gui.py. У цьому модулі передбачено два можливі режими роботи: режим обробки відеофайлу та режим обробки відеопотоку з камери.

За замовчуванням при запуску програми відкривається вікно класу YOLOApp, яке відповідає за взаємодію з відеофайлами. Якщо користувач у межах GUI натискає кнопку переходу до камери (реалізовано через метод switch_to_camera()), то поточне вікно закривається, і викликається інтерфейс CameraWindow. Аналогічно, у CameraWindow реалізовано можливість повернення до режиму відеофайлу.

Таким чином, main.py виконує такі дії:

1. Імпортує бібліотеку tkinter та класи YOLOApp і CameraWindow з модуля gui.py.
2. Створює головне вікно через tk.Tk().
3. Запускає клас YOLOApp як початковий інтерфейс.
4. Визначає функції зворотного виклику (callback) для перемикання між вікнами, які передаються як аргументи під час створення екземплярів класів.
5. Запускає головний цикл подій графічного інтерфейсу (mainloop()).

Оскільки tkinter не підтримує одночасне існування декількох активних вікон у межах одного Tk(), для переходу між режимами реалізовано закриття поточного вікна та створення нового екземпляра Tk() [14].

Фрагмент логіки перемикання між режимами (повний код див. додаток Д):

```
def switch_to_camera():
    root.destroy()
    new_root = tk.Tk()
    CameraWindow(new_root, switch_to_video)
    new_root.mainloop()

def switch_to_video():
    root.destroy()
    new_root = tk.Tk()
    YOLOApp(new_root, switch_to_camera)
    new_root.mainloop()
```

Головна перевага такого підходу – централізоване керування сценаріями запуску, що дозволяє уникнути дублювання коду між режимами та забезпечити гнучкість масштабування. Файл main.py не виконує жодної обробки відео напряму, однак є координатором роботи всіх ключових модулів системи.

3.3.5 Обробка результатів і візуалізація

Після завершення основного процесу обробки відео система формує набір вихідних даних, які мають практичну цінність для подальшого аналізу або збереження. Обробка результатів включає збереження координат виявлених об'єктів, побудову графіків, формування теплової карти, а також запис обробленого відео з візуалізацією детекцій.

Усі вихідні файли зберігаються у директорії, яку вказує користувач у графічному інтерфейсі. Іменування файлів відбувається на основі назви відео або умовного індексу (у разі роботи з камерою).

3.3.5.1 Координати виявлених об’єктів

Після проходження кожного кадру система фіксує наступну інформацію:

- номер кадру;
- тип об’єкта (наприклад, «person», «car»);
- унікальний ідентифікатор об’єкта (track ID);
- координати рамки у форматі [x_1, y_1, x_2, y_2].

Ці дані записуються у два формати: текстовий файл .txt, який структурований построчно та табличний файл .xlsx – аналогічний за змістом, створений за допомогою pandas.DataFrame, зручний для подальшого аналізу у табличних редакторах.

3.3.5.2 Побудова графіків

Для оцінки динаміки присутності об’єктів система автоматично створює два графіки за допомогою matplotlib [15]:

– Графік кількості об’єктів за кадрами – відображає зміну загальної кількості виявлень у кожному кадрі.

– Графік кількості об’єктів за класами – відображає окремо частотність появи об’єктів кожного типу (люди, транспорт тощо) у часі.

Ці графіки генеруються у форматі .png та зберігаються з назвами «detections_plot.png» та «types_plot.png».

3.3.5.3 Побудова теплової карти

У ході обробки кадрів координати виявлених об’єктів накопичуються у матриці heatmap, де кожна клітинка відповідає кількості поява об’єктів у відповідному пікселі зображення.

Після завершення обробки:

1. матриця нормалізується;
2. візуалізується як зображення з використанням кольорової шкали hot;
3. результат зберігається у форматі .png під назвою «heatmap.png».

Теплова карта дозволяє наочно побачити, в яких частинах кадру найчастіше з’являлися об’єкти – це корисно, зокрема, для аналізу поведінки у просторі (наприклад, у системах відеоспостереження).

3.3.5.4 Оброблене відео

Всі кадри, у яких виконувалась візуалізація результатів детекції та трекінгу (рамки, мітки класів, ID), послідовно записуються у новий відеофайл у форматі .mp4, де:

- кольори рамок залежать від класу об’єкта;
- інформація про клас і ідентифікатор виводиться над кожним об’єктом;
- відео створюється через cv2.VideoWriter.

Ім’я вихідного відео формується автоматично на основі вхідного, наприклад «input_out.mp4».

Усі результати зберігаються в окремій папці, що забезпечує чітке структурування виводу. Такий підхід дозволяє користувачу легко інтерпретувати отримані результати, повторно аналізувати дані, вбудовувати їх у інші системи чи просто візуально оцінювати ефективність виявлення об’єктів.

3.4 Керівництво користувача

У даному підрозділі наведено покрокову інструкцію щодо встановлення, запуску та практичного використання розробленого програмного забезпечення. Інтерфейс системи побудований з урахуванням доступності для користувача, що не володіє спеціальними знаннями у галузі комп'ютерного зору або програмування. Програма підтримує два режими роботи – офлайн-обробка відеофайлів та робота з вебкамерою в реальному часі, які реалізовано через два взаємопов'язані графічні інтерфейси.

3.4.1 Установка необхідних бібліотек та мови програмування

Перед запуском програми необхідно мати встановлену мову програмування Python версії 3.12. Рекомендується використовувати середовище розробки Visual Studio Code, однак запуск можливий і з командного рядка.

Для встановлення всіх необхідних бібліотек достатньо відкрити термінал у кореневій директорії проєкту та виконати команду: «pip install -r requirements.txt» (рис. 3.1). Ця команда інсталює такі бібліотеки: ultralytics, torch, opencv-python, pandas, matplotlib, numpy, openpyxl.

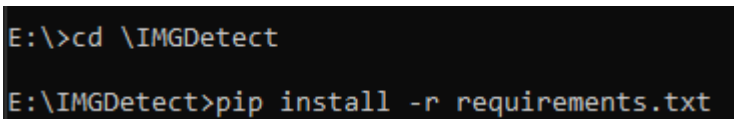


Рисунок 3.1 – Використання CMD для встановлення необхідних бібліотек

Примітка: у разі відсутності GPU програма автоматично працюватиме у CPU-режимі, однак швидкість обробки знизиться.

3.4.2 Запуск програми

Для запуску системи необхідно відкрити файл «main.py». Це можна зробити двома способами:

1. через середовище розробки (наприклад, Visual Studio Code);

2. через командний рядок, виконавши команду:

Використовуючи перший спосіб, необхідно відкрити файл за допомогою будь-якого середовища розробки (рис. 3.2).

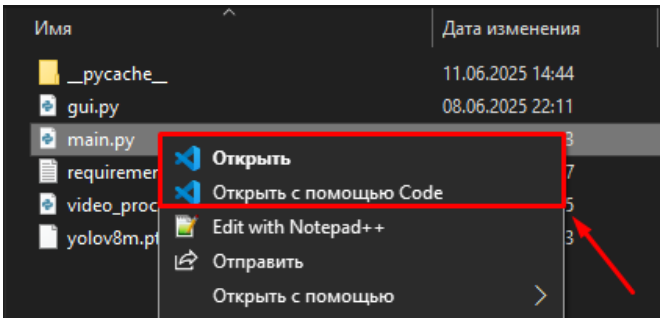


Рисунок 3.2 – Приклад використання VS Code для відкривання файлу

Далі безпосередньо запустити файл Python (рис. 3.3).

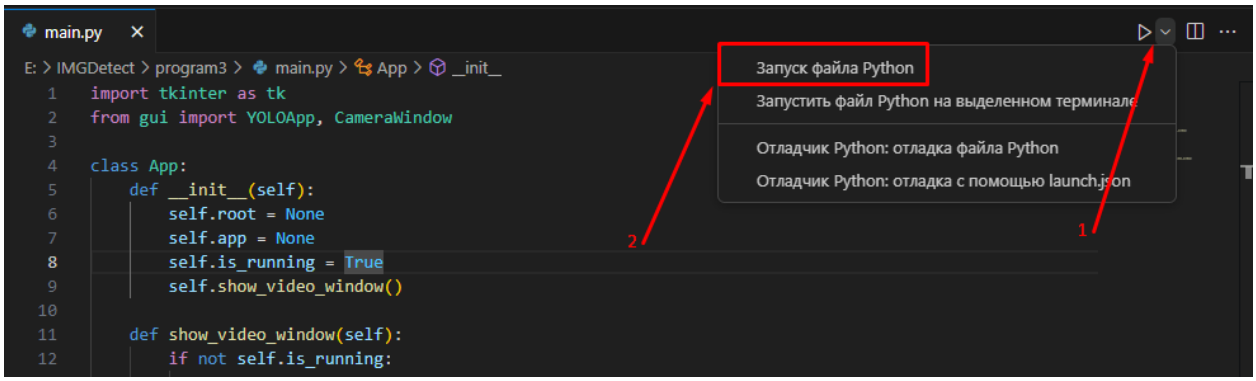


Рисунок 3.3 – Запуск програми через VS Code

В разі необхідності, можна також використати термінал CMD для того, щоб запустити програму. Для цього, знаходячись всередині каталогу ввести команду запуску: «python main.py» (рис. 3.4).

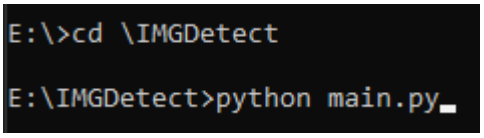


Рисунок 3.4 – Запуск програми через CMD

Після запуску програми відкривається графічне вікно, яке відповідає за обробку відеофайлів.

3.4.3 Робота в режимі обробки відеофайлу

У режимі обробки відеофайлу користувач бачить графічне вікно з наступними елементами (рис. 3.5):

1. кнопка «Обрати відео» – вибір відеофайлу;
2. кнопка «Обрати папку» – вибір директорії для збереження результатів;
3. прапорці – вибір типів вихідних файлів: .txt, .xlsx, графіки, теплова карта;
4. прогрес-бар – відображає прогрес процесу обробки;
5. кнопка «Обробити відео» – запуск обробки;
6. кнопка «Перейти до режиму камери» – перехід до режиму камери.

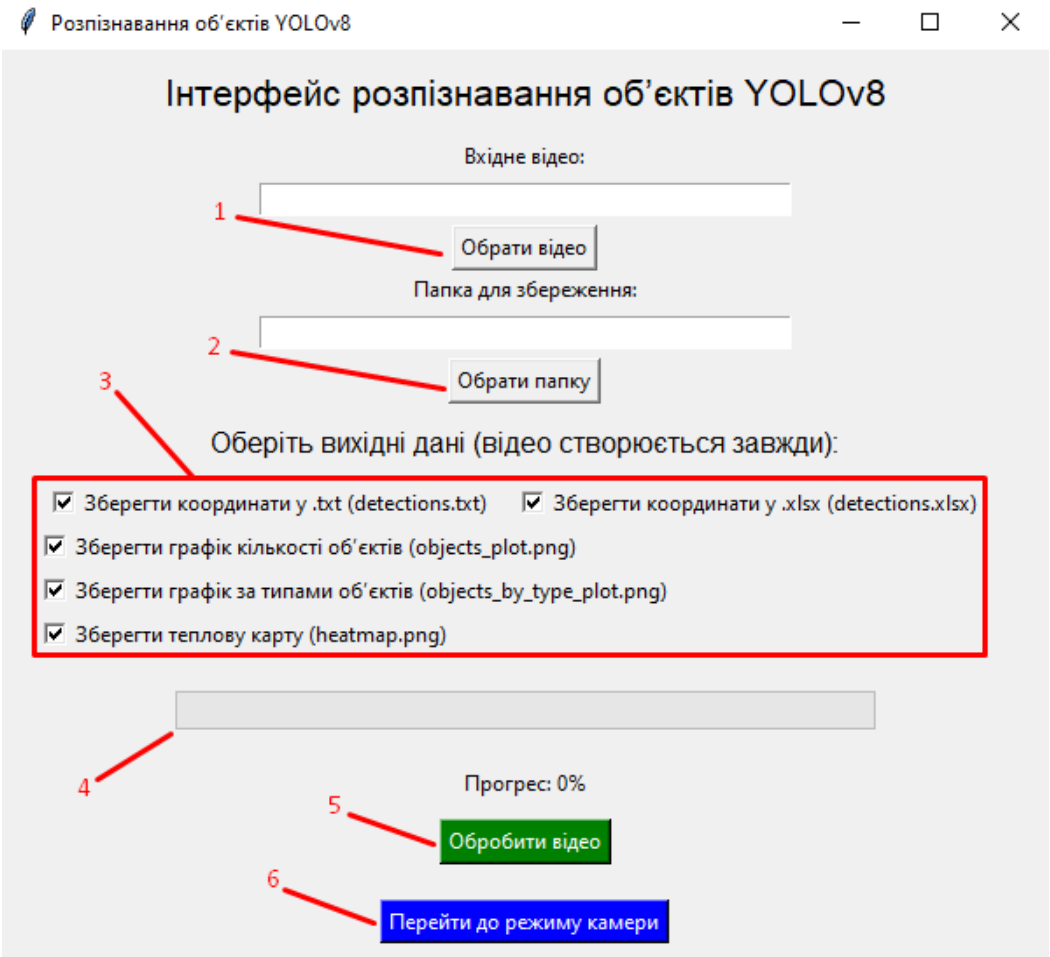


Рисунок 3.5 – Інтерфейс режиму відеофайлу з підписаними елементами

Послідовність дій користувача:

1. обрати відеофайл для обробки;
2. вибрати папку, в яку буде збережено результати;
3. позначити необхідні типи вихідних даних;
4. натиснути кнопку «Обробити відео»;
5. дочекатися завершення обробки (за потреби – слідкувати за прогрес-баром).

Після завершення обробки у вказаній директорії з’являться файли з результатами (рис. 3.6):

– оброблене відео з детекцією;
– координати об’єктів;
– графіки та теплова карта (якщо обрано).

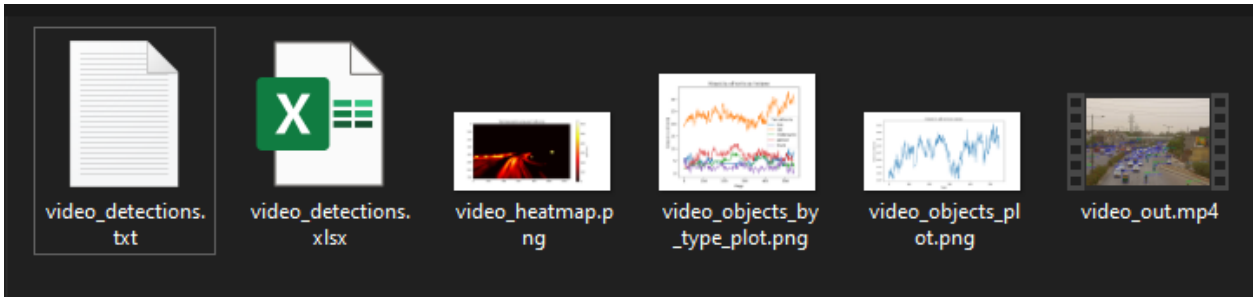


Рисунок 3.6 – файли-результати, які з’являються після обробки відео

З детальними результатами, що були отримані в ході обробки відео, можна ознайомитися в додатках Е, Ж, З, И.

3.4.4 Робота в режимі камери

Для переходу до режиму камери необхідно натиснути кнопку «Перейти до режиму камери». Програма автоматично відкриє нове вікно CameraWindow, яке містить елементи для взаємодії з потоком з камери (рис. 3.7):

1. поле (canvas) відображення відеопотоку з камери;
2. кнопка «Обрати папку» – вибір місця для збереження записаного відео;
3. напис (label), що відображає поточний статус роботи програми;

4. прапорці – вибір типів вихідних файлів: .txt, .xlsx, графіки, теплова карта;

5. кнопка «Почати запис» – запуск обробки потоку з камери;

6. кнопка «Зупинити запис» – зупинка запису та збереження результатів;

7. кнопка «Повернутися до режиму відеофайлів» – повернення до попереднього режиму.

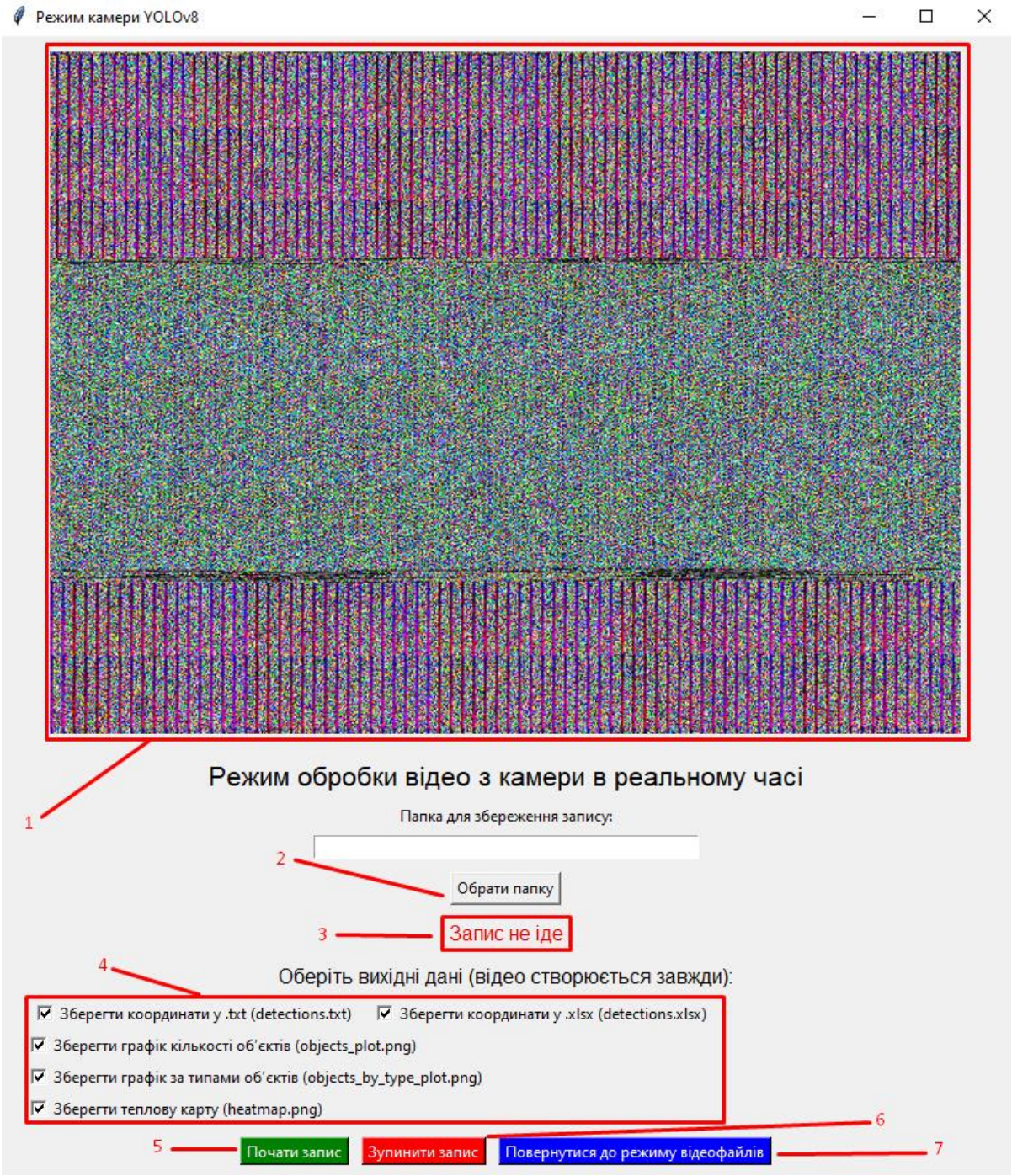


Рисунок 3.7 – Інтерфейс режиму камери з підписаними елементами

Послідовність дій користувача:

1. обрати папку збереження результатів;
2. натиснути «Почати запис»;
3. дочекатися завершення потрібного періоду зйомки;
4. натиснути «Зупинити запис» – програма автоматично обробить зняті кадри та збереже результати схожого характеру до тих, що були отримані в ході обробки відео в офлайн режимі.

Такий підхід до реалізації інтерфейсу дозволяє зручно перемикатися між режимами, зберігаючи єдину логіку обробки даних і забезпечуючи користувачу простий, інтуїтивний процес взаємодії з системою.

Висновки до розділу

У розділі розглянуто програмну реалізацію системи розпізнавання об’єктів у відеопотоці. Визначено засоби розробки, програмні бібліотеки та технічні вимоги до запуску системи. Побудовано модульну архітектуру з розподілом відповідальності між основними компонентами: головним модулем запуску, графічним інтерфейсом та ядром обробки відео.

Здійснено детальний опис логіки роботи класів, алгоритмів обробки, а також механізмів збереження результатів. Реалізовано підтримку двох режимів: офлайн-обробки відеофайлів і реального часу через камеру. Окремо описано принципи побудови графіків, теплових карт та формування координат у зручних форматах.

У підрозділі також надано інструкції для користувача, які дозволяють швидко встановити необхідне програмне забезпечення, запустити систему та отримати результати без додаткової технічної підготовки.

Отримані результати демонструють функціональну повноту реалізованого програмного забезпечення та його придатність до практичного застосування.

РОЗДІЛ 4
ОХОРОНА ПРАЦІ

4.1 Регулювання питань охорони праці на законодавчому рівні

Охорона праці є ключовим елементом організації робочих процесів на підприємствах та в організаціях, зокрема тих, що здійснюють правоохоронну діяльність і аналітику дорожнього руху. Відповідно до Закону України «Про охорону праці» № 2694-XII від 14 жовтня 1992 року, охорона праці спрямована на створення безпечних і нешкідливих умов праці, запобігання професійним захворюванням і нещасним випадкам на виробництві [18]. Основним завданням законодавства у сфері охорони праці є встановлення прав і обов'язків роботодавців, працівників та державних органів для забезпечення безпеки на робочих місцях.

Кодекс законів про працю України (КЗпП) визначає обов'язки роботодавця щодо створення безпечних умов праці, проведення інструктажів і навчання працівників з питань охорони праці [19]. Державні санітарні норми і правила при роботі з джерелами електромагнітних полів (ДСанПіН 3.3.6.096-2002) встановлюють вимоги до організації робочих місць, де використовується комп'ютерна техніка, що є актуальним для розробки та експлуатації системи розпізнавання об'єктів у відеопотоці [20]. Крім того, ДСТУ ISO 45001:2019 «Системи управління охороною здоров'я та безпекою праці» визначає стандарти для впровадження ефективних заходів з управління ризиками на робочих місцях [21].

Роль посадових осіб у реалізації законодавчих положень з охорони праці є визначальною. Роботодавець несе відповідальність за організацію безпечних умов праці, забезпечення працівників засобами індивідуального захисту та регулярну перевірку технічного стану обладнання. Фахівець з охорони праці контролює дотримання нормативних вимог, проводить аналіз потенційних ризиків і розробляє рекомендації щодо їх зниження. Працівники зобов'язані

дотримуватися інструкцій з охорони праці, повідомляти про виявлені небезпеки та брати участь у навчаннях з безпеки. Державні органи, зокрема Державна служба України з питань праці, здійснюють нагляд за дотриманням законодавства, проводять інспекції та розробляють нові нормативні акти для підвищення рівня безпеки.

З соціально-економічної точки зору, охорона праці сприяє підвищенню продуктивності праці, зниженню витрат на лікування професійних захворювань і компенсацію нещасних випадків, а також формуванню позитивного іміджу організації. У контексті правоохоронної діяльності, де використовується розроблена система, забезпечення безпеки працівників є пріоритетом, оскільки від їхньої ефективності залежить якість виконання службових завдань, таких як моніторинг дорожнього руху та виявлення правопорушень.

4.2 Виявлення потенційних небезпек стосовно об’єкта проєктування

Об’єктом проєктування є інформаційна система розпізнавання об’єктів у відеопотоці з використанням глибоких нейронних мереж, призначена для використання у правоохоронній діяльності та аналітиці дорожнього руху. Аналіз потенційних небезпечних і шкідливих виробничих факторів проведено відповідно до класифікації [22] з урахуванням особливостей діяльності працівників, які розробляють, тестують і експлуатують систему, а також специфіки правоохоронного контексту, де система застосовується.

Основні небезпеки, пов’язані з розробкою та експлуатацією системи, поділено на фізичні, психофізіологічні, біологічні та загальні фактори:

1. Фізичні фактори:

– Недостатня освітленість робочої зони, що виникає під час тривалої роботи з моніторами для перегляду та аналізу відеопотоку. Це може спричинити втому зорового апарату, зниження концентрації та підвищення ймовірності помилок у виявленні правопорушень.

– Підвищений рівень електромагнітного випромінювання від комп’ютерів, серверів, відеокамер та іншої апаратної інфраструктури, необхідної для обробки даних нейронних мереж. Тривалий вплив цього фактора може негативно позначитися на здоров’ї працівників, викликаючи головний біль або порушення сну.

– Ураження електричним струмом через несправність електричного обладнання, наприклад, серверів, камер спостереження чи кабельних мереж, що використовуються в системі.

– Загроза виникнення пожежі внаслідок короткого замикання електропроводки, перегрівання серверного обладнання або неправильного використання електроприладів у приміщеннях, де розміщена система.

– Підвищений рівень шуму, спричинений роботою охолоджувальних систем серверів, що може створювати дискомфорт і знижувати продуктивність працівників.

– Підвищена температура повітря в серверних приміщеннях через інтенсивну роботу обладнання, що може призводити до теплового стресу.

2. Психофізіологічні фактори:

– Розумове перенапруження, пов’язане з тривалою обробкою великих обсягів відеоданих і прийняттям рішень у реальному часі, що є типовим для операторів у правоохоронній діяльності, наприклад, під час моніторингу дорожнього руху чи виявлення підозрілих об’єктів.

– Монотонність праці, викликана повторюваними завданнями аналізу відеопотоку, що знижує рівень уваги та підвищує ризик пропуску критичних подій.

– Емоційні перевантаження, зумовлені високою відповідальністю за точність розпізнавання об’єктів, особливо в ситуаціях, пов’язаних із запобіганням злочинам або реагуванням на надзвичайні події.

– Фізичні перевантаження, зокрема статичні, через тривале перебування в сидячому положенні під час роботи за комп’ютером, що може спричинити порушення постави або болі в спині.

3. Біологічні фактори:

– Ризик поширення патогенних мікроорганізмів у закритих приміщеннях, де працюють оператори системи, особливо за умов недостатньої вентиляції або скупчення людей. Цей фактор є актуальним у контексті епідеміологічних загроз, таких як вірусні інфекції.

– Контакт із продуктами життєдіяльності мікроорганізмів, наприклад, пліснявою, яка може утворюватися в приміщеннях із підвищеною вологістю, де розміщене серверне обладнання, що впливає на якість повітря та може викликати алергічні реакції.

– Вплив біологічно активних речовин, що можуть виділятися під час технічного обслуговування обладнання (наприклад, дезінфікуючих засобів, які використовуються для очищення поверхонь камер чи серверів), що може спричинити подразнення шкіри або дихальних шляхів.

– Ризик укусів або контакту з комахами чи гризунами, які можуть проникати в технічні приміщення, де розташовані камери спостереження або сервери, особливо якщо ці приміщення розташовані в польових умовах або мають недостатній рівень санітарного захисту.

4. Загальні небезпеки:

– Військова загроза, що є актуальною в умовах сучасної геополітичної ситуації в Україні. У разі воєнних дій, таких як обстріли чи диверсії, функціонування системи може бути порушено, а працівники можуть опинитися в небезпеці.

– Надзвичайні ситуації природного характеру, наприклад, повені чи землетруси, які можуть пошкодити інфраструктуру системи, зокрема серверні приміщення або камери спостереження, та загрожувати безпеці персоналу.

Виявлені небезпеки можуть бути частково або повністю усунуті шляхом впровадження організаційних і технічних заходів. Для цього необхідно провести оцінку ризиків, пов’язаних із зазначеними факторами, та розробити рекомендації щодо їх мінімізації, що буде розглянуто в наступному підрозділі.

4.3 Дослідження ризику реалізації потенційних небезпек на об’єкті проєктування та розробка заходів щодо їх попередження

Процедура оцінки ризику є важливим інструментом забезпечення безпеки праці, спрямованим на ідентифікацію, аналіз та мінімізацію потенційних небезпек на робочих місцях. Згідно з ДСТУ ISO 45001:2019, оцінка ризиків передбачає систематичний процес, який включає виявлення небезпечних факторів, визначення ймовірності їх реалізації та оцінку можливих наслідків для здоров’я і безпеки працівників [21]. Основними завданнями цієї процедури є: запобігання нещасним випадкам і професійним захворюванням, підвищення ефективності управління безпекою та відповідність вимогам законодавства, зокрема Закону України «Про охорону праці» № 2694-XII від 14 жовтня 1992 року [18]. Особливості оцінки ризиків полягають у необхідності врахування специфіки об’єкта проєктування, умов його експлуатації та категорій працівників, що підлягають впливу небезпек.

Для аналізу однієї з ключових небезпек, виявлених у підрозділі 4.2, а саме «ураження електричним струмом», застосовано метод «дерева відмов» (рис. 4.1). Цей метод дозволяє графічно зобразити послідовність подій, що можуть призвести до небажаної події, та визначити можливі причини її виникнення.

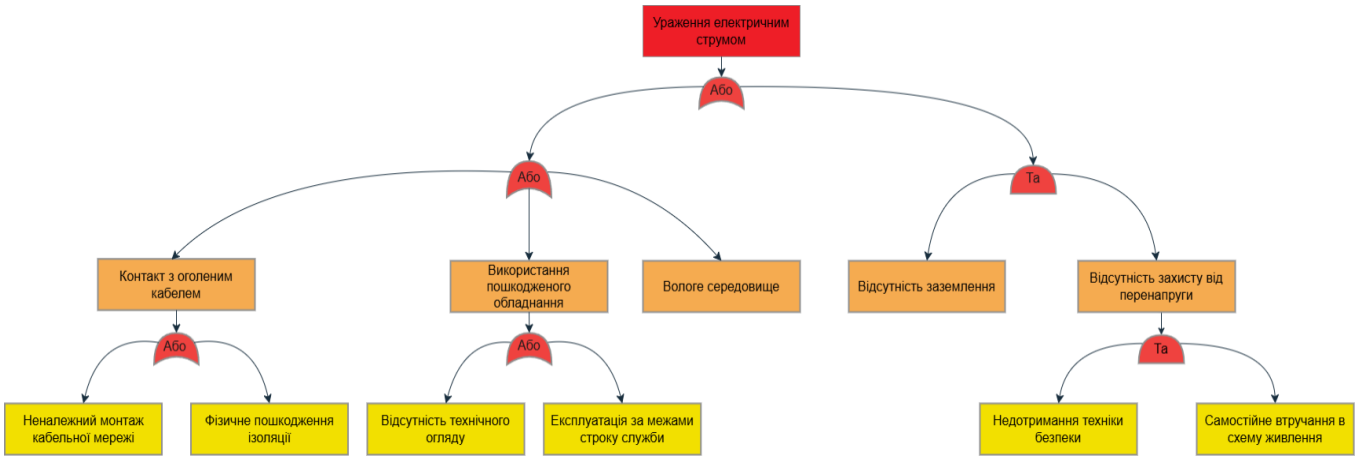


Рисунок 4.1 – Дерево відмов небезпечної події «ураження електричним струмом»

Небажаною подією (кінцевою подією) визначено «Ураження електричним струмом», яке може статися під час експлуатації системи розпізнавання об’єктів у відеопотоці через несправність електричного обладнання (серверів, камер спостереження чи кабельних мереж). Аналіз причин поділено на дві основні категорії: «Або» (умова або умова) і «Та» (умова + умова), що відображають логічні зв’язки між подіями [23].

1. Категорія «Або»:

– Контакт із пошкодженим кабелем як основна причина ураження. Ця подія може виникнути через неналежний монтаж електропроводки або фізичне пошкодження кабелю під час експлуатації.

– Використання пошкодженого обладнання, наприклад, серверів або камер із пошкодженою ізоляцією, що може бути результатом зносу або неправильного технічного обслуговування.

– Вологе середовище може виникати внаслідок конденсації вологи на поверхнях через підвищений рівень вологості повітря або внаслідок розливу рідин поблизу обладнання, що може спричинити коротке замикання електричного обладнання.

2. Категорія «Та»:

– Використання технічного обладнання без належного заземлення, що підвищує ризик ураження струмом у разі замикання.

– Експлуатація за межами струму захисту, що може статися через недотримання техніки безпеки працівниками (наприклад, відсутність використання захисних засобів або ігнорування інструкцій) та самостійне втручання в систему живлення, що може включати несанкціонований ремонт або підключення обладнання без відповідної кваліфікації.

На основі аналізу «дерева відмов» встановлено, що ймовірність реалізації небезпеки «ураження електричним струмом» залежить від сукупності технічних і організаційних факторів. Зокрема, комбінація пошкодженого обладнання та відсутності заземлення значно підвищує ризик.

Для попередження реалізації зазначеної небезпеки пропонуються наступні заходи:

– Проведення регулярного технічного обслуговування електрообладнання з перевіркою стану кабелів, ізоляції та заземлення відповідно до ДСТУ EN 60950-1:2014 [24].

– Забезпечення працівників засобами індивідуального захисту (діелектричними рукавицями, взуттям) та проведення інструктажів з техніки безпеки відповідно до Типового положення про службу охорони праці (НПАОП 0.00-4.21-04) [24].

– Встановлення пристроїв захисного відключення (ПЗВ) для автоматичного відключення електромережі при виявленні замикання.

– Обмеження доступу некваліфікованого персоналу до електрообладнання шляхом впровадження чітких процедур і контролю за виконанням робіт.

– Організація навчання працівників щодо правильного використання обладнання та реагування на потенційні небезпеки.

Запропоновані заходи спрямовані на усунення основних причин ураження електричним струмом, таких як технічні несправності та порушення

правил експлуатації. Регулярне обслуговування та використання ПЗВ зменшать технічні ризики, тоді як навчання та контроль персоналу сприятимуть зниженню організаційних факторів. Ефективність цих заходів буде підтверджена шляхом моніторингу стану обладнання та аналізу інцидентів протягом експлуатації системи.

Висновки до розділу

У розділі розглянуто законодавчі основи забезпечення охорони праці, із врахуванням специфіки правоохоронної діяльності. Виявлено потенційні небезпеки, пов’язані з експлуатацією системи розпізнавання об’єктів у відеопотоці, включаючи ураження електричним струмом, психофізіологічні перевантаження, біологічні фактори та військові загрози. Проведено оцінку ризику ураження електричним струмом за методом «дерева відмов» [23]. Запропоновано технічні (техобслуговування, ПЗВ) та організаційні (навчання, контроль доступу) заходи для зниження ризиків, що сприятиме створенню безпечних умов праці в правоохоронному контексті.

ЗАГАЛЬНІ ВИСНОВКИ

У межах кваліфікаційної роботи було розроблено програмну систему для розпізнавання об’єктів у відеопотоці з використанням глибоких нейронних мереж. В ході дослідження було здійснено аналіз предметної області, виявлено актуальні проблеми відеоаналітики, обґрунтовано вибір сучасних методів комп’ютерного зору, серед яких особливу увагу приділено архітектурі YOLOv8.

У теоретичній частині сформульовано вимоги до майбутньої системи, побудовано математичні моделі та алгоритми роботи, що охоплюють процеси виявлення об’єктів, їх відстеження та взаємодії з користувачем через графічний інтерфейс. Було проведено проєктування архітектури ПЗ із застосуванням структурних діаграм, зокрема діаграми класів і послідовності.

У практичному розділі реалізовано основні функціональні модулі системи: обробка відеопотоку в реальному часі та збережених відеофайлів, інтеграція моделі YOLOv8, використання трекера ByteTrack, а також GUI-інтерфейс із підтримкою перемикання режимів. Програму протестовано на власному комп’ютері з підтримкою GPU, що забезпечило стабільну роботу з високою продуктивністю.

Також у роботі розглянуто питання охорони праці під час експлуатації системи. Виявлено потенційні небезпеки для користувача та розробника, зокрема пов’язані з електробезпекою та тривалим перебуванням за комп’ютером. На основі методу дерева небезпеки розроблено рекомендації щодо мінімізації ризиків.

У результаті виконаної роботи досягнуто поставленої мети – реалізовано систему, що поєднує сучасні методи глибинного навчання, модулі відеоаналітики та користувацький інтерфейс, що придатна до практичного використання й подальшого розвитку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. YOLOv8 – офіційний сайт // Ultralytics. – 2024. – URL: https://yolov8.com/ (дата звернення: 12.06.2025).

2. Faster R-CNN // Papers With Code. – 2024. – URL: https://paperswithcode.com/method/faster-r-cnn (дата звернення: 12.06.2025).

3. Ren S. et al. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks // arXiv.org. – 2015. – URL: https://arxiv.org/abs/1512.02325 (дата звернення: 12.06.2025).

4. OpenVINO Documentation Index // Intel OpenVINO. – 2025. – URL: https://docs.openvino.ai/2025/index.html (дата звернення: 12.06.2025).

5. DeepinView Series Cameras // Hikvision. – 2024. – URL: https://www.hikvision.com/en/products/IP-Products/Network-Cameras/DeepinView-Series (дата звернення: 12.06.2025).

6. DeepStream SDK // NVIDIA Developer. – 2024. – URL: https://developer.nvidia.com/deepstream-sdk (дата звернення: 12.06.2025).

7. Gehrig D., Scaramuzza D. Low-latency automotive vision with event cameras // Nature. – 2024. – Режим доступу: https://doi.org/10.1038/s41586-024-07409-w (дата звернення: 12.06.2025).

8. Wang Z., Jin L., Wang S., Xu H. Apple stem/calyx real-time recognition using YOLO-v5 algorithm for fruit automatic loading system // Postharvest Biology and Technology. – 2022. – Vol. 185, 111808. – DOI: https://doi.org/10.1016/j.postharvbio.2021.111808

9. Python 3.12.8 Release Notes // Python.org. – 2024. – URL: https://www.python.org/downloads/release/python-3128/ (дата звернення: 12.06.2025).

10. Visual Studio Code – Version 1.96 // Visual Studio Code. – 2024. – URL: https://code.visualstudio.com/updates/v1_96 (дата звернення: 12.06.2025).

11. OpenCV – Open Source Computer Vision Library // OpenCV.org. – 2024. – URL: https://opencv.org/ (дата звернення: 12.06.2025).

12. ByteTrack – PyPI Documentation // PyPI. – 2024. – URL: https://pypi.org/project/bytetracker/#description (дата звернення: 12.06.2025).

13. Ultralytics – YOLO Family Models // Ultralytics. – 2024. – URL: https://www.ultralytics.com/ (дата звернення: 12.06.2025).

14. Tkinter Standard Library // Python Documentation. – 2024. – URL: https://docs.python.org/3/library/tkinter.html (дата звернення: 12.06.2025).

15. Matplotlib User Guide // Matplotlib.org. – 2024. – URL: https://matplotlib.org/stable/users/index.html (дата звернення: 12.06.2025).

16. PyTorch Documentation // PyTorch.org. – 2024. – URL: https://docs.pytorch.org/docs/stable/index.html#pytorch-documentation (дата звернення: 12.06.2025).

17. Pandas Documentation // pandas.pydata.org. – 2024. – URL: https://pandas.pydata.org/docs/ (дата звернення: 12.06.2025).

18. Закон України «Про охорону праці» [Електронний ресурс] // Офіційний сайт Верховної Ради України. – URL: https://zakon.rada.gov.ua/laws/show/2694-12 (дата звернення: 12.06.2025).

19. Кодекс цивільного захисту України [Електронний ресурс] // Офіційний сайт Верховної Ради України. – URL: https://zakon.rada.gov.ua/laws/show/z1494-18#n109 (дата звернення: 12.06.2025).

20. Державні санітарні норми і правила при роботі з джерелами електромагнітних полів : ДСанПіН 3.3.6.096-2002. – [Чинний від 2002-12-18] . – Київ : МОЗ, 2002. – 24 с.

21. Системи управління охороною здоров’я та безпекою праці. Вимоги та настанови щодо застосування : ДСТУ ISO 45001:2019. – [Чинний від 01.01.2021]. – Київ : ДП «УкрНДНЦ», 2019. – 23 с.

22. Бедрій Я.І. Основи охорони праці користувачів персональних комп’ютерів / Я.І. Бедрій : навч. посіб. – Тернопіль. : Навчальна книга – Богдан, 2014. – 144 с

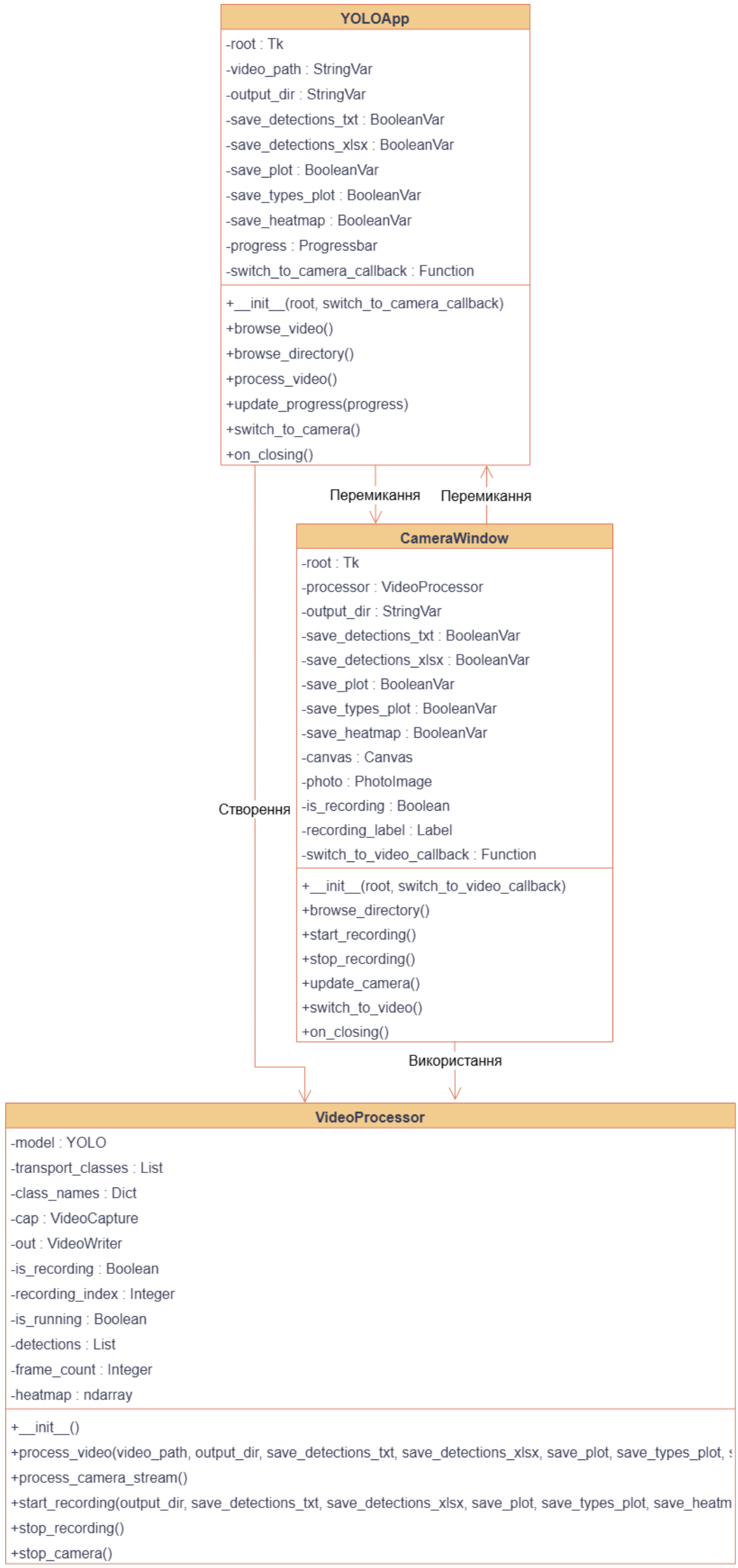
23. Малишева В. В. Методичні вказівки до виконання розділу «Охорона праці» в дипломних роботах бакалаврів для студентів 4 курсу галузі знань 12 «Інформаційні технології» спеціальностей 122, 126, галузі знань 15 «Автоматизація та приладобудування» спеціальності 151 / В. В. Малишева. – Харків : ХНУМГ ім. О. М. Бекетова, 2021. – 13 с.

24. Типове положення про службу охорони праці : НПАОП 0.00-4.21-04 [Електронний ресурс] // Офіційний сайт Верховної Ради України. – URL: https://zakon.rada.gov.ua/laws/show/z1526-04#Text (дата звернення: 12.06.2025).

ДОДАТКИ

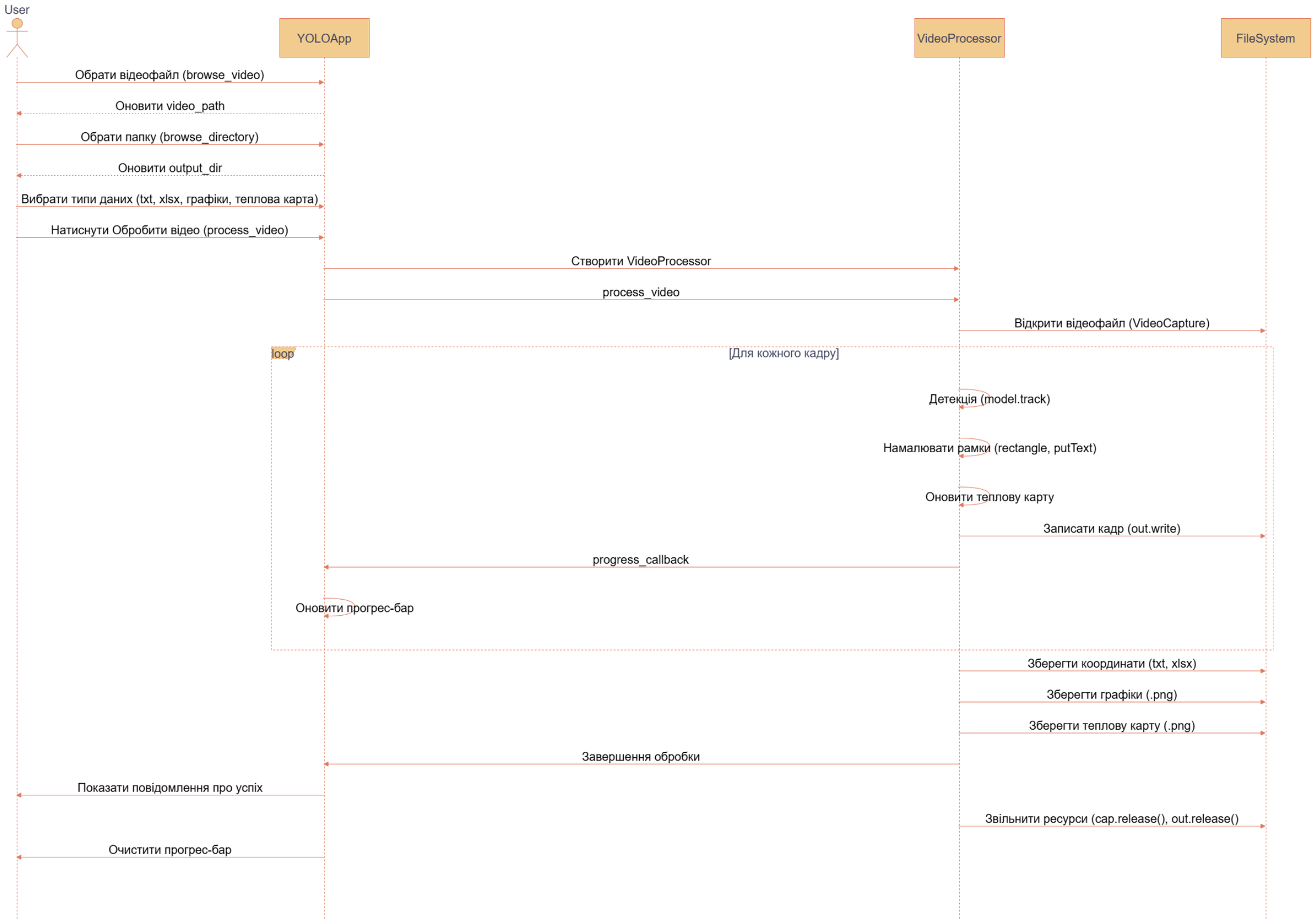
Додаток А

Діаграма потоків даних між основними компонентами системи



Додаток Б

UML-діаграма послідовності взаємодії компонентів системи



Додаток В

Лістинг коду допоміжного модуля (gui.py)

```
import tkinter as tk
from tkinter import filedialog, messagebox, ttk
from video_processor import VideoProcessor
from PIL import Image, ImageTk
import cv2
import numpy as np

class YOLOApp:
    def __init__(self, root, switch_to_camera_callback):
        self.root = root
        self.root.title("Розпізнавання об’єктів YOLOv8")
        self.root.geometry("600x520")
        self.switch_to_camera_callback = switch_to_camera_callback

        # Змінні для зберігання шляхів і вибору
        self.video_path = tk.StringVar()
        self.output_dir = tk.StringVar()
        self.save_detections_txt = tk.BooleanVar(value=True)
        self.save_detections_xlsx = tk.BooleanVar(value=True)
        self.save_plot = tk.BooleanVar(value=True)
        self.save_types_plot = tk.BooleanVar(value=True)
        self.save_heatmap = tk.BooleanVar(value=True)

        # GUI елементи
        tk.Label(root, text="Інтерфейс розпізнавання об’єктів YOLOv8",
font=("Arial", 16)).pack(pady=10)

        tk.Label(root, text="Вхідне відео:").pack()
        tk.Entry(root, textvariable=self.video_path, width=50).pack(pady=5)
        tk.Button(root, text="Обрати відео", command=self.browse_video).pack()

        tk.Label(root, text="Папка для збереження:").pack()
        tk.Entry(root, textvariable=self.output_dir, width=50).pack(pady=5)
        tk.Button(root, text="Обрати папку",
command=self.browse_directory).pack()

        # Чекбокси для вибору вихідних даних
        tk.Label(root, text="Оберіть вихідні дані (відео створюється завжди):",
font=("Arial", 12)).pack(pady=10)

        # Фрейм для чекбоксів .txt і .xlsx
        detections_frame = tk.Frame(root)
        detections_frame.pack(anchor="w", padx=20)
        tk.Checkbutton(detections_frame, text="Зберегти координати у .txt
(detections.txt)", variable=self.save_detections_txt).pack(side="left", padx=5)
```

```
        tk.Checkbutton(detections_frame, text="Зберегти координати у .xlsx
(detections.xlsx)", variable=self.save_detections_xlsx).pack(side="left", padx=5)

        tk.Checkbutton(root, text="Зберегти графік кількості об’єктів
(objects_plot.png)", variable=self.save_plot).pack(anchor="w", padx=20)
        tk.Checkbutton(root, text="Зберегти графік за типами об’єктів
(objects_by_type_plot.png)", variable=self.save_types_plot).pack(anchor="w",
padx=20)
        tk.Checkbutton(root, text="Зберегти теплову карту (heatmap.png)",
variable=self.save_heatmap).pack(anchor="w", padx=20)

        # Прогрес-бар
        self.progress = ttk.Progressbar(root, orient="horizontal", length=400,
mode="determinate")
        self.progress.pack(pady=20)
        self.progress_label = tk.Label(root, text="Прогрес: 0%")
        self.progress_label.pack()

        tk.Button(root, text="Обробити відео", command=self.process_video,
bg="green", fg="white").pack(pady=10)
        tk.Button(root, text="Перейти до режиму камери",
command=self.switch_to_camera, bg="blue", fg="white").pack(pady=10)

        # Обробка закриття вікна
        self.root.protocol("WM_DELETE_WINDOW", self.on_closing)

    def browse_video(self):
        file_path = filedialog.askopenfilename(filetypes=[("Відео файли", "*.mp4
*.avi")])
        if file_path:
            self.video_path.set(file_path)

    def browse_directory(self):
        dir_path = filedialog.askdirectory()
        if dir_path:
            self.output_dir.set(dir_path)

    def update_progress(self, progress):
        self.progress["value"] = progress
        self.progress_label.config(text=f"Прогрес: {int(progress)}%")
        self.root.update()

    def process_video(self):
        video_path = self.video_path.get()
        output_dir = self.output_dir.get()

        if not video_path or not output_dir:
            messagebox.showerror("Помилка", "Будь ласка, оберіть відео та папку
для збереження!")
            return
```

```
        try:
            processor = VideoProcessor()
            processor.process_video(
                video_path=video_path,
                output_dir=output_dir,
                save_detections_txt=self.save_detections_txt.get(),
                save_detections_xlsx=self.save_detections_xlsx.get(),
                save_plot=self.save_plot.get(),
                save_types_plot=self.save_types_plot.get(),
                save_heatmap=self.save_heatmap.get(),
                progress_callback=self.update_progress
            )
            messagebox.showinfo("Успіх", f"Обробка завершена!\nРезультати
збережено до: {output_dir}")
        except Exception as e:
            messagebox.showerror("Помилка", f"Сталася помилка: {str(e)}")
        finally:
            self.progress["value"] = 0
            self.progress_label.config(text="Прогрес: 0%")

    def switch_to_camera(self):
        try:
            self.switch_to_camera_callback()
        except Exception as e:
            messagebox.showerror("Помилка", f"Не вдалося перейти до режиму
камери: {str(e)}")
        finally:
            self.root.destroy()

    def on_closing(self):
        self.root.destroy()

class CameraWindow:
    def __init__(self, root, switch_to_video_callback):
        self.root = root
        self.root.title("Режим камери YOLOv8")
        self.root.geometry("800x900")
        self.switch_to_video_callback = switch_to_video_callback
        self.processor = VideoProcessor()
        self.is_recording = False
        self.output_dir = tk.StringVar()
        self.photo = None

        # Змінні для чекбоксів
        self.save_detections_txt = tk.BooleanVar(value=True)
        self.save_detections_xlsx = tk.BooleanVar(value=True)
        self.save_plot = tk.BooleanVar(value=True)
        self.save_types_plot = tk.BooleanVar(value=True)
        self.save_heatmap = tk.BooleanVar(value=True)
```

```
        # Полотно для відео
        self.canvas = tk.Canvas(root, width=720, height=540)
        self.canvas.pack(pady=10)

        # GUI елементи
        tk.Label(root, text="Режим обробки відео з камери в реальному часі",
font=("Arial", 16)).pack(pady=5)

        tk.Label(root, text="Папка для збереження запису:").pack()
        tk.Entry(root, textvariable=self.output_dir, width=50).pack(pady=5)
        tk.Button(root, text="Обрати папку",
command=self.browse_directory).pack(pady=5)

        # Статус запису
        self.recording_label = tk.Label(root, text="Запис не іде", font=("Arial",
12), fg="red")
        self.recording_label.pack(pady=5)

        # Чекбокси для вибору вихідних даних
        tk.Label(root, text="Оберіть вихідні дані (відео створюється завжди):",
font=("Arial", 12)).pack(pady=5)

        # Фрейм для чекбоксів .txt і .xlsx
        detections_frame = tk.Frame(root)
        detections_frame.pack(anchor="w", padx=20)
        tk.Checkbutton(detections_frame, text="Зберегти координати у .txt
(detections.txt)", variable=self.save_detections_txt).pack(side="left", padx=5)
        tk.Checkbutton(detections_frame, text="Зберегти координати у .xlsx
(detections.xlsx)", variable=self.save_detections_xlsx).pack(side="left", padx=5)

        tk.Checkbutton(root, text="Зберегти графік кількості об’єктів
(objects_plot.png)", variable=self.save_plot).pack(anchor="w", padx=20)
        tk.Checkbutton(root, text="Зберегти графік за типами об’єктів
(objects_by_type_plot.png)", variable=self.save_types_plot).pack(anchor="w",
padx=20)
        tk.Checkbutton(root, text="Зберегти теплову карту (heatmap.png)",
variable=self.save_heatmap).pack(anchor="w", padx=20)

        # Фрейм для кнопок
        button_frame = tk.Frame(root)
        button_frame.pack(pady=10)
        tk.Button(button_frame, text="Почати запис",
command=self.start_recording, bg="green", fg="white").pack(side="left", padx=5)
        tk.Button(button_frame, text="Зупинити запис",
command=self.stop_recording, bg="red", fg="white").pack(side="left", padx=5)
        tk.Button(button_frame, text="Повернутися до режиму відеофайлів",
command=self.switch_to_video, bg="blue", fg="white").pack(side="left", padx=5)

        # Обробка закриття вікна
```

```
        self.root.protocol("WM_DELETE_WINDOW", self.on_closing)

        # Запуск обробки камери
        self.update_camera()

    def browse_directory(self):
        dir_path = filedialog.askdirectory()
        if dir_path:
            self.output_dir.set(dir_path)

    def start_recording(self):
        if not self.output_dir.get():
            messagebox.showerror("Помилка", "Будь ласка, оберіть папку для
збереження!")
            return
        if not self.is_recording:
            try:
                self.is_recording = True
                self.recording_label.config(text="Запис іде", fg="green")
                self.processor.start_recording(
                    output_dir=self.output_dir.get(),
                    save_detections_txt=self.save_detections_txt.get(),
                    save_detections_xlsx=self.save_detections_xlsx.get(),
                    save_plot=self.save_plot.get(),
                    save_types_plot=self.save_types_plot.get(),
                    save_heatmap=self.save_heatmap.get()
                )
            except Exception as e:
                messagebox.showerror("Помилка", f"Не вдалося почати запис:
{str(e)}")
                self.is_recording = False
                self.recording_label.config(text="Запис не іде", fg="red")

    def stop_recording(self):
        if self.is_recording:
            self.is_recording = False
            self.recording_label.config(text="Запис не іде", fg="red")
            self.processor.stop_recording()

    def update_camera(self):
        try:
            frame = self.processor.process_camera_stream()
            if frame is not None:
                # Конвертація кадру для tkinter
                frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                img = Image.fromarray(frame_rgb)
                img = img.resize((720, 540), Image.Resampling.LANCZOS)
                self.photo = ImageTk.PhotoImage(img)
                self.canvas.create_image(0, 0, image=self.photo, anchor=tk.NW)
            self.root.after(10, self.update_camera)
```

```
        except Exception as e:
            messagebox.showerror("Помилка", f"Сталася помилка: {str(e)}")
            self.on_closing()

    def switch_to_video(self):
        try:
            self.processor.stop_camera()
            self.switch_to_video_callback()
        except Exception as e:
            messagebox.showerror("Помилка", f"Не вдалося повернутися до режиму
відеофайлів: {str(e)}")
        finally:
            self.root.destroy()

    def on_closing(self):
        self.processor.stop_camera()
        self.root.destroy()
```

Додаток Г

Лістинг коду допоміжного модуля (video_processor.py)

```
import cv2
from ultralytics import YOLO
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from pathlib import Path
import os

class VideoProcessor:
    def __init__(self):
        self.model = YOLO("yolov8m.pt")
        self.transport_classes = ["car", "motorcycle", "bus", "truck"]
        self.class_names = {0: "person", 2: "car", 3: "motorcycle", 5: "bus", 7:
"truck"}
        self.cap = None
        self.out = None
        self.is_recording = False
        self.recording_index = 0
        self.is_running = False
        self.detections = []
        self.frame_count = 0
        self.heatmap = None

    def process_video(self, video_path, output_dir, save_detections_txt,
save_detections_xlsx,
                      save_plot, save_types_plot, save_heatmap,
progress_callback):
        # Ім'я вихідних файлів
        video_name = Path(video_path).stem
        output_video = os.path.join(output_dir, f"{video_name}_out.mp4")
        detections_txt = os.path.join(output_dir, f"{video_name}_detections.txt")
        detections_xlsx = os.path.join(output_dir,
f"{video_name}_detections.xlsx")
        plot_file = os.path.join(output_dir, f"{video_name}_objects_plot.png")
        types_plot_file = os.path.join(output_dir,
f"{video_name}_objects_by_type_plot.png")
        heatmap_file = os.path.join(output_dir, f"{video_name}_heatmap.png")

        # Відкриття відео
        self.cap = cv2.VideoCapture(video_path)
        if not self.cap.isOpened():
            raise Exception("Не вдалося відкрити відео файл")

        # Параметри відео
        width = int(self.cap.get(cv2.CAP_PROP_FRAME_WIDTH))
```

```
        height = int(self.cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
        fps = int(self.cap.get(cv2.CAP_PROP_FPS))
        total_frames = int(self.cap.get(cv2.CAP_PROP_FRAME_COUNT))
        fourcc = cv2.VideoWriter_fourcc(*"mp4v")
        self.out = cv2.VideoWriter(output_video, fourcc, fps, (width, height))

        # Ініціалізація для аналізу
        self.detections = []
        self.frame_count = 0
        self.heatmap = np.zeros((height, width), dtype=np.float32)

        # Очищення detections.txt, якщо обрано
        if save_detections_txt:
            with open(detections_txt, "w", encoding="utf-8") as f:
                f.write("Кадр,Тип об’єкта,TrackID,Координати\n")

        while self.cap.isOpened():
            ret, frame = self.cap.read()
            if not ret:
                break

            # Детекція об’єктів із трекінгом
            results = self.model.track(
                frame,
                persist=True,
                conf=0.3,
                classes=[0, 2, 3, 5, 7],
                max_det=1000,
                imgsz=1280,
                tracker="bytetrack.yaml"
            )

            # Обробка результатів
            for det in results[0].boxes:
                if det.id is not None:
                    x1, y1, x2, y2 = map(int, det.xyxy[0])
                    cls = int(det.cls[0])
                    label = self.class_names.get(cls, "unknown")
                    track_id = int(det.id[0])
                    conf = float(det.conf[0])

                    # Зменшення рамки на 5%
                    shrink_factor = 0.05
                    width_box = x2 - x1
                    height_box = y2 - y1
                    x1 += int(width_box * shrink_factor)
                    y1 += int(height_box * shrink_factor)
                    x2 -= int(width_box * shrink_factor)
                    y2 -= int(height_box * shrink_factor)
```

```
                    # Вибір кольору
                    color = (0, 255, 0) if label == "person" else (255, 0, 0) if
label in self.transport_classes else (0, 255, 0)

                    # Малювання рамки
                    cv2.rectangle(frame, (x1, y1), (x2, y2), color, 1)
                    # Відображення ID та класу
                    cv2.putText(frame, f"{label} ID:{track_id}", (x1, y1 - 20),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 1)
                    # Відображення коефіцієнта впевненості
                    cv2.putText(frame, f"{conf:.2f}", (x1, y1 - 10),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 1)

                    # Збереження координат
                    if save_detections_txt or save_detections_xlsx:
                        coords = f"[{x1:d};{y1:d};{x2:d};{y2:d}]"
                        self.detections.append([self.frame_count, label,
track_id, coords])

                    # Оновлення теплової карти
                    if save_heatmap:
                        self.heatmap[y1:y2, x1:x2] += 1

            # Запис кадру
            self.out.write(frame)
            self.frame_count += 1

            # Оновлення прогрес-бару
            progress = (self.frame_count / total_frames) * 100
            progress_callback(progress)

        # Збереження координат
        if (save_detections_txt or save_detections_xlsx) and self.detections:
            df = pd.DataFrame(self.detections, columns=["Кадр", "Тип об’єкта",
"TrackID", "Координати"])
            if save_detections_txt:
                df.to_csv(detections_txt, index=False, encoding="utf-8")
            if save_detections_xlsx:
                df.to_excel(detections_xlsx, index=False, engine="openpyxl")

        # Графік кількості об’єктів
        if save_plot and self.detections:
            df = pd.DataFrame(self.detections, columns=["Кадр", "Тип об’єкта",
"TrackID", "Координати"])
            df_grouped = df.groupby("Кадр")["Тип об’єкта"].count()
            plt.figure(figsize=(10, 5))
            plt.plot(df_grouped.index, df_grouped.values)
            plt.xlabel("Кадр")
            plt.ylabel("Кількість об’єктів")
            plt.title("Кількість об’єктів за часом")
```

```
            plt.savefig(plot_file)
            plt.close()

        # Графік за типами об’єктів
        if save_types_plot and self.detections:
            df = pd.DataFrame(self.detections, columns=["Кадр", "Тип об’єкта",
"TrackID", "Координати"])
            counts_by_type = df.groupby(["Кадр", "Тип
об’єкта"]).size().unstack(fill_value=0)
            plt.figure(figsize=(10, 5))
            counts_by_type.plot(kind="line")
            plt.xlabel("Кадр")
            plt.ylabel("Кількість об’єктів")
            plt.title("Кількість об’єктів за типами")
            plt.savefig(types_plot_file)
            plt.close()

        # Теплова карта
        if save_heatmap:
            plt.figure(figsize=(10, 5))
            plt.imshow(self.heatmap, cmap="hot", interpolation="nearest")
            plt.colorbar(label="Щільність")
            plt.title("Теплова карта щільності об’єктів")
            plt.savefig(heatmap_file)
            plt.close()

        # Очищення ресурсів
        self.cap.release()
        self.out.release()
        self.detections = []
        self.frame_count = 0
        self.heatmap = None

    def process_camera_stream(self):
        if not self.is_running:
            self.cap = cv2.VideoCapture(0)
            if not self.cap.isOpened():
                raise Exception("Не вдалося відкрити камеру")
            self.is_running = True
            width = int(self.cap.get(cv2.CAP_PROP_FRAME_WIDTH))
            height = int(self.cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
            self.heatmap = np.zeros((height, width), dtype=np.float32)
            self.detections = []
            self.frame_count = 0

        if self.is_running:
            ret, frame = self.cap.read()
            if not ret:
                self.stop_camera()
                return None
```

```
            # Детекція об’єктів із трекінгом
            results = self.model.track(
                frame,
                persist=True,
                conf=0.3,
                classes=[0, 2, 3, 5, 7],
                max_det=1000,
                imgsz=1280,
                tracker="bytetrack.yaml"
            )

            # Обробка результатів
            for det in results[0].boxes:
                if det.id is not None:
                    x1, y1, x2, y2 = map(int, det.xyxy[0])
                    cls = int(det.cls[0])
                    label = self.class_names.get(cls, "unknown")
                    track_id = int(det.id[0])
                    conf = float(det.conf[0])

                    # Зменшення рамки на 5%
                    shrink_factor = 0.05
                    width_box = x2 - x1
                    height_box = y2 - y1
                    x1 += int(width_box * shrink_factor)
                    y1 += int(height_box * shrink_factor)
                    x2 -= int(width_box * shrink_factor)
                    y2 -= int(height_box * shrink_factor)

                    # Вибір кольору
                    color = (0, 255, 0) if label == "person" else (255, 0, 0) if
label in self.transport_classes else (0, 255, 0)

                    # Малювання рамки
                    cv2.rectangle(frame, (x1, y1), (x2, y2), color, 1)
                    # Відображення ID та класу
                    cv2.putText(frame, f"{label} ID:{track_id}", (x1, y1 - 20),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 1)
                    # Відображення коефіцієнта впевненості
                    cv2.putText(frame, f"{conf:.2f}", (x1, y1 - 10),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 1)

                    # Збереження координат, якщо запис увімкнено
                    if self.is_recording and (self.save_detections_txt or
self.save_detections_xlsx):
                        coords = f"[{x1:d};{y1:d};{x2:d};{y2:d}]"
                        self.detections.append([self.frame_count, label,
track_id, coords])
```

```
                    # Оновлення теплової карти, якщо запис увімкнено
                    if self.is_recording and self.save_heatmap:
                        self.heatmap[y1:y2, x1:x2] += 1

            # Запис кадру, якщо увімкнено
            if self.is_recording and self.out is not None:
                self.out.write(frame)

            self.frame_count += 1
            return frame
        return None

    def start_recording(self, output_dir, save_detections_txt,
save_detections_xlsx, save_plot, save_types_plot, save_heatmap):
        if not self.cap or not self.cap.isOpened():
            raise Exception("Камера не ініціалізована")

        self.save_detections_txt = save_detections_txt
        self.save_detections_xlsx = save_detections_xlsx
        self.save_plot = save_plot
        self.save_types_plot = save_types_plot
        self.save_heatmap = save_heatmap

        # Знайти унікальний індекс для імені файлу
        while True:
            output_path = os.path.join(output_dir,
f"camera_out_{self.recording_index}.mp4")
            if not os.path.exists(output_path):
                break
            self.recording_index += 1

        # Імена файлів для додаткових даних
        self.detections_txt = os.path.join(output_dir,
f"camera_detections_{self.recording_index}.txt")
        self.detections_xlsx = os.path.join(output_dir,
f"camera_detections_{self.recording_index}.xlsx")
        self.plot_file = os.path.join(output_dir,
f"camera_objects_plot_{self.recording_index}.png")
        self.types_plot_file = os.path.join(output_dir,
f"camera_objects_by_type_plot_{self.recording_index}.png")
        self.heatmap_file = os.path.join(output_dir,
f"camera_heatmap_{self.recording_index}.png")

        # Параметри відео
        width = int(self.cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        height = int(self.cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
        fps = int(self.cap.get(cv2.CAP_PROP_FPS)) or 30
        fourcc = cv2.VideoWriter_fourcc(*"mp4v")
        self.out = cv2.VideoWriter(output_path, fourcc, fps, (width, height))
```

```
        # Ініціалізація detections.txt
        if save_detections_txt:
            with open(self.detections_txt, "w", encoding="utf-8") as f:
                f.write("Кадр,Тип об’єкта,TrackID,Координати\n")

        self.is_recording = True

    def stop_recording(self):
        if self.is_recording:
            # Збереження координат
            if (self.save_detections_txt or self.save_detections_xlsx) and
self.detections:
                df = pd.DataFrame(self.detections, columns=["Кадр", "Тип
об’єкта", "TrackID", "Координати"])
                if self.save_detections_txt:
                    df.to_csv(self.detections_txt, index=False, encoding="utf-8")
                if self.save_detections_xlsx:
                    df.to_excel(self.detections_xlsx, index=False,
engine="openpyxl")

            # Графік кількості об’єктів
            if self.save_plot and self.detections:
                df = pd.DataFrame(self.detections, columns=["Кадр", "Тип
об’єкта", "TrackID", "Координати"])
                df_grouped = df.groupby("Кадр")["Тип об’єкта"].count()
                plt.figure(figsize=(10, 5))
                plt.plot(df_grouped.index, df_grouped.values)
                plt.xlabel("Кадр")
                plt.ylabel("Кількість об’єктів")
                plt.title("Кількість об’єктів за часом")
                plt.savefig(self.plot_file)
                plt.close()

            # Графік за типами об’єктів
            if self.save_types_plot and self.detections:
                df = pd.DataFrame(self.detections, columns=["Кадр", "Тип
об’єкта", "TrackID", "Координати"])
                counts_by_type = df.groupby(["Кадр", "Тип
об’єкта"]).size().unstack(fill_value=0)
                plt.figure(figsize=(10, 5))
                counts_by_type.plot(kind="line")
                plt.xlabel("Кадр")
                plt.ylabel("Кількість об’єктів")
                plt.title("Кількість об’єктів за типами")
                plt.savefig(self.types_plot_file)
                plt.close()

            # Теплова карта
            if self.save_heatmap and self.heatmap is not None:
                plt.figure(figsize=(10, 5))
```

```
                plt.imshow(self.heatmap, cmap="hot", interpolation="nearest")
                plt.colorbar(label="Щільність")
                plt.title("Теплова карта щільності об’єктів")
                plt.savefig(self.heatmap_file)
                plt.close()

        if self.out is not None:
            self.out.release()
            self.out = None
        self.is_recording = False
        self.recording_index += 1
        self.detections = []
        self.frame_count = 0

    def stop_camera(self):
        self.is_running = False
        if self.cap is not None:
            self.cap.release()
            self.cap = None
        if self.out is not None:
            self.out.release()
            self.out = None
        self.detections = []
        self.frame_count = 0
        self.heatmap = None
```

Додаток Д

Лістинг коду головного модуля програми (main.py)

```
import tkinter as tk
from gui import YOLOApp, CameraWindow

class App:
    def __init__(self):
        self.root = None
        self.app = None
        self.is_running = True
        self.show_video_window()

    def show_video_window(self):
        if not self.is_running:
            return
        if self.root is not None:
            try:
                self.root.destroy()
            except tk.TclError:
                pass
        self.root = tk.Tk()
        self.app = YOLOApp(self.root, self.show_camera_window)

    def show_camera_window(self):
        if not self.is_running:
            return
        if self.root is not None:
            try:
                self.root.destroy()
            except tk.TclError:
                pass
        self.root = tk.Tk()
        self.app = CameraWindow(self.root, self.show_video_window)

    def run(self):
        try:
            self.root.mainloop()
        except tk.TclError:
            pass
        finally:
            self.is_running = False
            if self.root is not None:
                try:
                    self.root.destroy()
                except tk.TclError:
                    pass
```

```
if __name__ == "__main__":
    app = App()
    app.run()
```

Додаток Е

Оброблене відео з детекцією об’єктів

Додаток Ж

файли-результати, що містять координати об’єктів

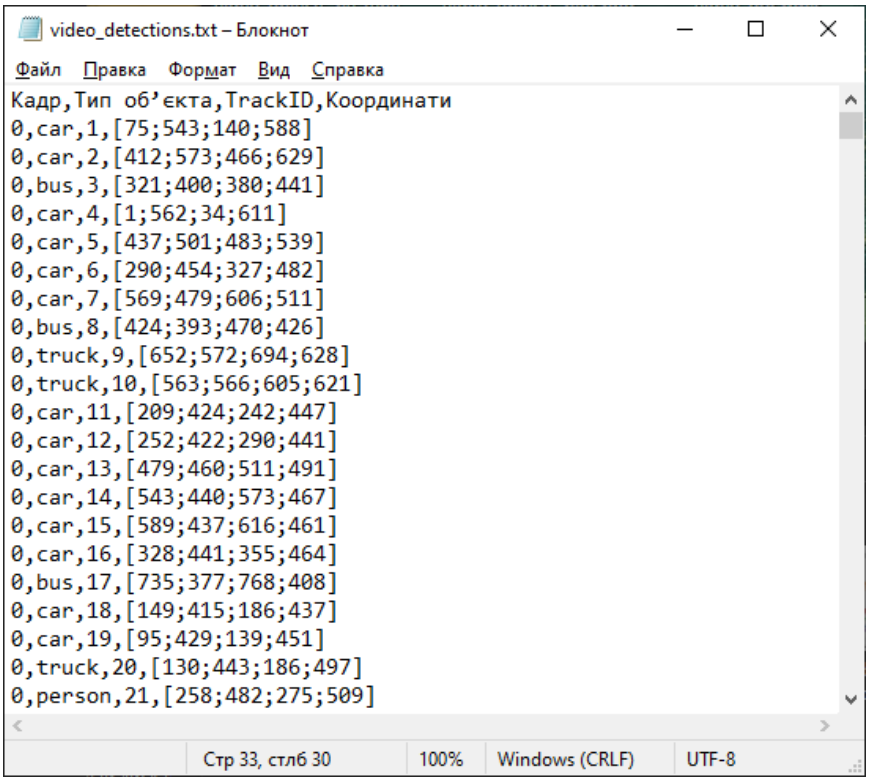


Рисунок – тектсовий документ (.txt) з координатами об’єктів

	A	B	C	D	E	F	G
1	Кадр	Тип об’єкта	TrackID	Координати			
2	0	car	1	[75;543;140;588]			
3	0	car	2	[412;573;466;629]			
4	0	bus	3	[321;400;380;441]			
5	0	car	4	[1;562;34;611]			
6	0	car	5	[437;501;483;539]			
7	0	car	6	[290;454;327;482]			
8	0	car	7	[569;479;606;511]			
9	0	bus	8	[424;393;470;426]			
10	0	truck	9	[652;572;694;628]			
11	0	truck	10	[563;566;605;621]			
12	0	car	11	[209;424;242;447]			
13	0	car	12	[252;422;290;441]			
14	0	car	13	[479;460;511;491]			
15	0	car	14	[543;440;573;467]			
16	0	car	15	[589;437;616;461]			
17	0	car	16	[328;441;355;464]			
18	0	bus	17	[735;377;768;408]			
19	0	car	18	[149;415;186;437]			
20	0	car	19	[95;429;139;451]			
21	0	truck	20	[130;443;186;497]			
22	0	person	21	[258;482;275;509]			

Sheet1

Рисунок – таблиця MS Excel (.xlsx) з координатами об’єктів

Додаток З

файли-результати, що містять графіки

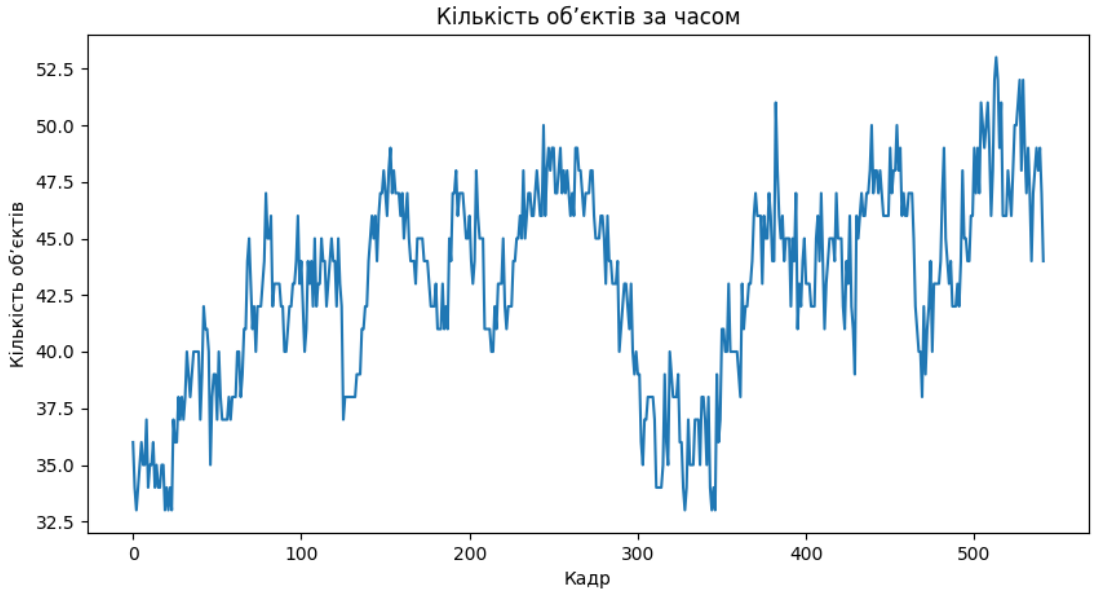


Рисунок – графік кількості об’єктів за часом

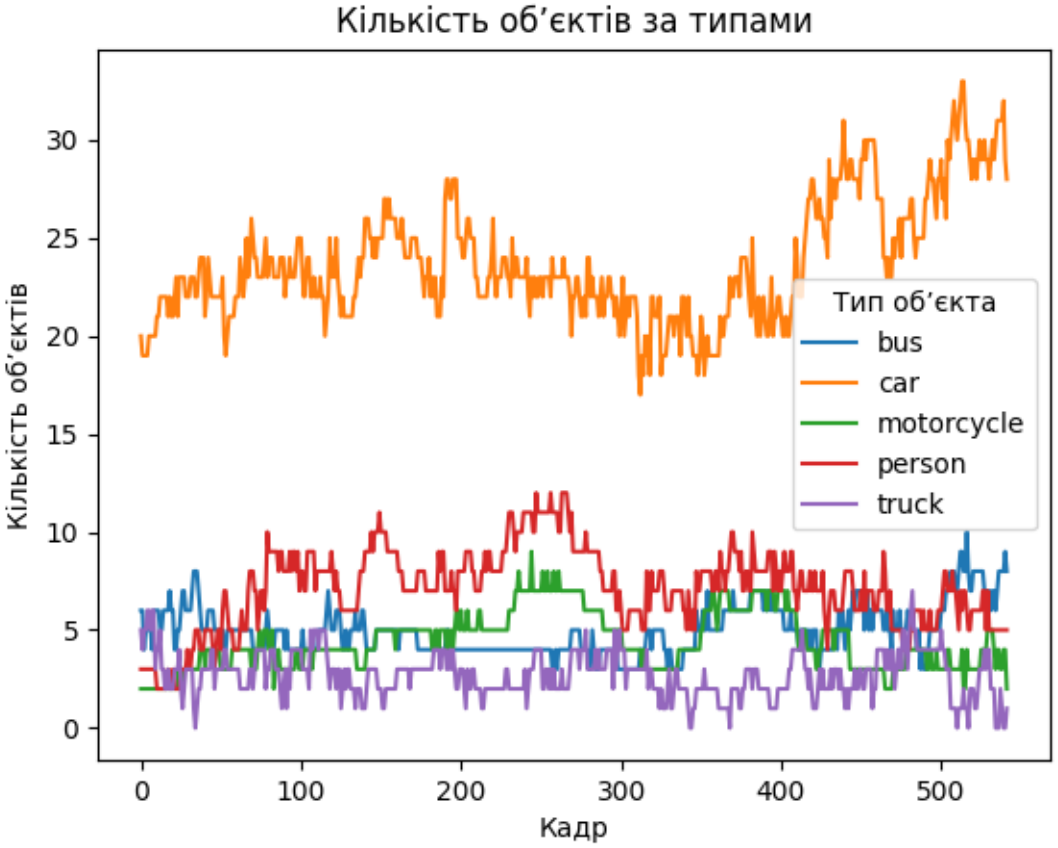


Рисунок – графік кількості об’єктів за типами

Додаток И

файл-результат, що містить теплову карту активності