МІНІСТЕРСТВО ВНУТРІШНІХ СПРАВ УКРАЇНИ
ЛЬВІВСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ ВНУТРІШНІХ СПРАВ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ УПРАВЛІННЯ, ПСИХОЛОГІЇ ТА БЕЗПЕКИ
Кафедра інформаційних технологій

Автоматизована система виявлення та аналізу рухомих об’єктів у відеопотоці камер спостереження

кваліфікаційна робота
здобувача вищої освіти
4 курсу денної форми навчання
Бориса ВОЗНЯКА

Науковий керівник:
Доктор філософії
Олег БАСИСТЮК

Рецензент:

_____

_____

Кваліфікаційна робота допущена до захисту
«___» _____ 2026 р., протокол № ____

Завідувач кафедри інформаційних технологій

_____ **Олег ЗАЧЕК**
(підпис)

Львів
2026

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота виконана студентом групи ІТ-42 Возняком Борисом Ігоровичом. Тема роботи: «Автоматизована система виявлення та аналізу рухомих об’єктів у відеопотоці камер спостереження». Робота подана на здобуття ступеня бакалавра за спеціальністю 126 «Інформаційні системи та технології» – Львівський державний університет внутрішніх справ, МВС України, Львів, 2026.

У даній роботі проведено аналіз сучасних систем відеоспостереження, методів визначення руху та алгоритмів комп’ютерного зору для автоматизованого моніторингу відеопотоку. Було досліджено особливості роботи систем детекції руху, методи аналізу кадрів у режимі реального часу, а також можливості використання нейронних мереж для визначення об’єктів у кадрі. У процесі розробки було виконано огляд сучасних бібліотек та засобів для створення desktop застосунків комп’ютерного зору, зокрема PyQt5, OpenCV [3], MSS[7] та Ultralytics YOLOv8. Також було досліджено принципи роботи алгоритмів порівняння кадрів, threshold обробки, contour detection та методів оптимізації навантаження під час аналізу відеопотоку.

Метою дипломної роботи є розробка автоматизованої системи виявлення та аналізу рухомих об’єктів у відеопотоці камер спостереження із використанням класичної детекції руху та моделі YOLOv8 для визначення об’єктів у зоні моніторингу. Об’єктом дослідження є процес автоматизованого моніторингу відеопотоку та виявлення активності у зоні спостереження. Предметом дослідження є методи детекції руху, алгоритми комп’ютерного зору, нейронні мережі YOLOv8 та програмні засоби для реалізації систем відеомоніторингу у режимі реального часу.

У результаті виконання дипломної роботи було розроблено програмний застосунок який дозволяє виконувати аналіз області екрана у режимі реального часу, визначати рух у зоні моніторингу, розпізнавати людей транспорт та інші об’єкти за допомогою моделі YOLOv8, а також автоматично створювати GIF-файли після виявлення активності.

Розроблена система підтримує overlay режим роботи поверх інших програм для перегляду камер, дозволяє створювати полігональні області детекції, змінювати розмір зони моніторингу та виконувати фільтрацію непотрібних спрацювань. Використання локальної обробки кадрів дозволяє уникнути передачі відеоданих на сторонні сервери та забезпечує стабільну роботу системи без необхідності використання окремої інфраструктури відеоспостереження.

Ключові слова: комп’ютерний зір, YOLOv8, детекція руху, відеоспостереження, OpenCV, PyQt5, MSS, overlay інтерфейс, нейронні мережі, моніторинг відеопотоку, object detection, автоматизована система.

ABSTRACT

This bachelor's thesis was completed by Boris Ihorovych Voznyak, a student in the IT-42 class. The topic of the thesis is: "An Automated System for Detecting and Analyzing Moving Objects in Surveillance Camera Video Streams." The thesis was submitted for the degree of Bachelor of Science in the specialty 126 "Information Systems and Technologies" – Lviv State University of Internal Affairs, Ministry of Internal Affairs of Ukraine, Lviv, 2026.

This thesis analyzes modern video surveillance systems, motion detection methods, and computer vision algorithms for automated video stream monitoring. It examines the characteristics of motion detection systems, methods for real-time frame analysis, and the potential use of neural networks to identify objects in a frame.

During the development process, a review was conducted of modern libraries and tools for creating desktop computer vision applications, specifically PyQt5, OpenCV, MSS, and Ultralytics YOLOv8. The principles of frame comparison algorithms, threshold processing, contour detection, and methods for optimizing computational load during video stream analysis were also investigated.

The goal of this thesis is to develop an automated system for detecting and analyzing moving objects in surveillance camera video streams using classical motion detection and the YOLOv8 model to identify objects within the monitored area.

The object of this study is the process of automated video stream monitoring and activity detection within a surveillance area.

The subject of this study includes motion detection methods, computer vision algorithms, YOLOv8 neural networks, and software tools for implementing real-time video monitoring systems.

As a result of this thesis, a software application was developed that allows for real-time analysis of a screen area, detection of motion within the monitoring zone, recognition of people, vehicles, and other objects using the YOLOv8 model, as well as the automatic creation of GIF files upon detecting activity.

The developed system supports overlay mode on top of other camera viewing applications, allows for the creation of polygonal detection areas, resizing of the monitoring zone, and filtering of false positives. The use of local frame processing avoids the transmission of video data to third-party servers and ensures stable system operation without the need for a separate video surveillance infrastructure.

ЗМІСТ

ВСТУП

Актуальність

В реаліях сучасності системи відеоспостереження генерують велику кількість трафіку. Це може бути як велике скупчення людей, потік машин на дорогах, так і багато інших зовнішніх джерел. Зазвичай буває важко спостерігати за великим обсягом руху на камерах, а у віддільних випадках є камери, на яких можна майже не акцентувати увагу, бо майже нічого не відбувається. І в такі моменти можуть статись казуси. Поки наглядач спостерігає за іншою камерою, не звертаючи уваги на ту, де зазвичай немає ніякого руху, він може пропустити потенційне порушення, будь то проникнення на особисту власність. Цьому можна запобігти за рахунок застосунку, який відправляє сповіщення про рух на камері. У випадках моніторингу великого об’єму камер потрібна автоматизація таких систем. Одним із варіантів може бути моніторинг камер лише в момент виявлення руху на них. Також за часту такий моніторинг може бути проблемним через так звану фальшиву детекцією руху, так як зміна кадрів на камері може бути спровокована як пробігаючим котиком, коливанням дерев, або ввімкненням і вимкненням світла, що змінюють кадри на потоці камер.

Аналіз існуючих рішень

Більшість програм для моніторингу камер за часту мають свою систему детекції руху. Вона використовується, в основному, у перегляді відеозаписів із затримкою, і лише деякі окремі застосунки можуть реалізувати це в реальному часі. Також за часту використовують AI систему, яка підключена через API віддільно до кожної камери. З комерційних застосунків можна зазначити Actuate та Analityck від 3dEYE. До недоліків можна віднести проблеми з самими камерами, тобто якість стандартного розширення картинки на дешевих камерах. Також проблемою можуть бути проблеми з потоком даних через сервера, умовно, якщо камера розташована далеко від сервера або має нестабільне інтернет з’єднання.

Мета розробки

Метою розробки є створення застосунку для моніторингу відео потоку з використанням нейронної мережі YOLOv8.

Завдання та дослідження

Провести аналіз сучасних систем відеоспостереження та методів детекції руху у відеопотоці.

Дослідити алгоритми визначення руху шляхом порівняння кадрів та оцінити їх ефективність при різних рівнях активності у кадрі.

Реалізувати систему виявлення та аналізу рухомих об’єктів із використанням моделі YOLOv8 для розпізнавання людей, транспорту та інших об’єктів.

Розробити графічний інтерфейс застосунку із підтримкою overlay режиму, запуску детекції, полігональних зон та допоміжних функцій моніторингу.

Реалізувати систему автоматичного запису GIF-файлів після виявлення руху та провести тестування застосунку в умовах різної інтенсивності руху для оцінки продуктивності та стабільності роботи системи.

Об’єкт дослідження

Об’єктом дослідження є процеси автоматизованого моніторингу та визначення активності в сегментах відео потоку.

Методи дослідження

У процесі виконання було використано методи комп’ютерного зору, алгоритм порівняння кадрів та технології детекції об’єктів за використання сформованої бази, яка включає в себе ці об’єкти. Реалізація виконувалась за використання мови програмування Python[1], бібліотек OpenCV, PyQt5, MSS, ImageIO та власне моделі YOLOv8.

Практична цінність роботи

Полягає в створенні програмного застосунку для автоматизації моніторингу відео потоку, який можна буде використовувати у системах охорони приватних територій, житлових приміщень. Розроблена система дозволяє зменшити обсяг роботи під час моніторингу камер, підвищити швидкість реагування на важливі події та знизити ризики людського фактору завдяки використанню алгоритму.

Предмет дослідження

Предметом дослідження є методи і засоби автоматизованого моніторингу та виявлення руху у відеопотоці.

РОЗДІЛ І АНАЛІЗ СИСТЕМ ВІДЕОСПОСТЕРЕЖЕННЯ ТА МЕТОДІВ ДЕТЕКЦІЇ РУХУ

Системи відеоспостереження на сьогоднішній день використовуються практично всюди. Найчастіше вони встановлюються на приватних територіях підприємствах парковках у магазинах складських приміщеннях житлових будинках або місцях великого скупчення людей. Основною задачею подібних систем є контроль території та можливість швидко помітити небезпечну або підозрілу активність у зоні спостереження.

У більшості випадків камери працюють у постійному режимі та безперервно записують відео потік через що накопичується велика кількість записів які у подальшому потрібно зберігати або переглядати вручну. При використанні невеликої кількості камер оператор ще може нормально контролювати ситуацію та слідкувати за подіями у реальному часі. Проте зі збільшенням кількості камер починає виникати проблема людського фактору оскільки людина фізично не може постійно концентрувати увагу одночасно на великій кількості відео потоків. У результаті частина важливих подій може залишатись непоміченою особливо у випадках коли на окремих камерах тривалий час майже немає активності.

Ще однією проблемою сучасних систем відеоспостереження є велика кількість непотрібної інформації яка накопичується під час постійного запису відео. У більшості випадків значна частина відеоматеріалу не містить важливих подій однак все одно займає місце на накопичувачах та потребує додаткового часу для перегляду. Саме через це у сучасних системах почали активно використовувати автоматичну детекцію руху яка дозволяє визначати момент появи активності у кадрі та реагувати лише тоді коли у зоні спостереження відбуваються певні зміни.

Найпростішим способом визначення руху є порівняння кадрів між собою. Якщо система бачить значну різницю між поточним та попереднім кадром це визначається як рух у зоні спостереження. Подібний підхід є досить простим у реалізації та не потребує великої кількості ресурсів однак у

реальних умовах він має ряд недоліків. На результат роботи можуть впливати погодні умови зміна освітлення тіні коливання дерев або цифрові шуми камери через що виникає велика кількість помилкових спрацювань.

Для покращення якості моніторингу у сучасних системах почали використовувати нейронні мережі та алгоритми розпізнавання об’єктів які дозволяють аналізувати не лише сам факт зміни кадру а й визначати які саме об’єкти знаходяться у зоні спостереження. Подібний підхід дозволяє значно зменшити кількість непотрібних спрацювань та зробити роботу системи більш стабільною.

У даному розділі буде проведено аналіз сучасних систем відеоспостереження методів детекції руху нейронних мереж для розпізнавання об’єктів а також бібліотек та засобів розробки які використовуються для створення програмного застосунку автоматизованого моніторингу відео потоку.

1.1 Аналіз сучасних систем відеоспостереження

Системи відеоспостереження на сьогодні використовуються майже у всіх сферах де потрібно контролювати територію або слідкувати за безпекою об’єкта. Найчастіше камери встановлюються на приватних територіях підприємствах парковках складах магазинах житлових будинках або у місцях великого скупчення людей. Основною задачею подібних систем є постійний моніторинг ситуації та збереження відео у випадку виникнення якихось подій. У більшості випадків камери працюють цілодобово та безперервно записують відео потік через що накопичується дуже велика кількість матеріалу який у подальшому потрібно або переглядати або зберігати на накопичувачах. При використанні невеликої кількості камер оператор ще може нормально контролювати ситуацію та слідкувати за всіма потоками одночасно. Але зі збільшенням кількості камер починає виникати проблема людського фактору. Людина фізично не може постійно концентрувати увагу на великій кількості екранів особливо у випадках коли на більшості камер тривалий час нічого не відбувається. Через це частина подій може бути пропущена або помічена із

запізненням. Особливо помітною дана проблема стає на великих об’єктах де одночасно використовується кілька десятків камер.

У старих системах відеоспостереження найчастіше використовувались CCTV камери які працювали через аналогову передачу сигналу. Відео передавалось напряму через кабель до монітора або пристрою запису. Основною перевагою подібних систем була простота роботи та відсутність необхідності налаштовувати мережеве підключення. Також такі системи могли працювати повністю локально без доступу до інтернету. Але з часом почали з’являтись проблеми пов’язані із якістю зображення та складністю масштабування подібних рішень особливо якщо потрібно було підключати велику кількість камер одночасно. З розвитком мережевих технологій почали активно використовуватись IP камери які передають відео потік через локальну мережу або інтернет. Подібний підхід дозволив значно спростити підключення великої кількості камер та зробив можливим віддалений моніторинг системи практично з будь якого місця. Також сучасні IP камери підтримують значно кращу якість відео у порівнянні зі старими аналоговими системами. Більшість сучасних моделей підтримують високу роздільну здатність нічний режим запису виявлення руху та інші допоміжні функції. Частина камер вже має вбудовані AI функції для базового аналізу кадру та розпізнавання об’єктів.

Для роботи із великою кількістю камер використовуються спеціальні системи моніторингу які називаються VMS. Основною задачею подібних систем є централізоване керування всіма камерами перегляд відео потоку збереження архівів записів та налаштування роботи всієї системи відеоспостереження. Також через подібні системи оператор може швидко перемикатись між камерами переглядати архіви або налаштовувати сценарії роботи системи. У сучасних VMS системах також часто використовуються функції автоматичної детекції руху або інтеграція із нейронними мережами. Для збереження відео у більшості випадків використовуються DVR або NVR системи. DVR переважно використовується у старих аналогових системах де

відео записується напряму із камер на локальний накопичувач. NVR використовується уже разом із IP камерами та працює із цифровими потоками відео. Основною задачею подібних систем є постійний запис відео та можливість подальшого перегляду архівів. Частина сучасних рішень також підтримує функції віддаленого доступу через інтернет автоматичну детекцію руху або базовий аналіз кадру.

Попри розвиток сучасних систем відеоспостереження основною проблемою все одно залишається постійний моніторинг великої кількості відео потоків. У більшості випадків система записує відео безперервно навіть тоді коли у кадрі фактично нічого не відбувається. Через це накопичується велика кількість непотрібного матеріалу який у подальшому потрібно переглядати вручну. Також оператор змушений постійно слідкувати за екранами навіть у моменти коли активності практично немає. Через тривалу концентрацію уваги людина починає швидше втомлюватись що збільшує ризик пропуску важливих подій. Для вирішення подібних проблем у сучасних системах почали активно використовувати автоматичну детекцію руху. Основна ідея такого підходу полягає у тому щоб система реагувала лише у момент появи активності в кадрі. Це дозволяє значно зменшити кількість часу який витрачається на перегляд архівів та спрощує моніторинг камер у реальному часі. Найпростішим методом визначення руху є порівняння кадрів між собою. Якщо система бачить значну різницю між поточним та попереднім кадром це визначається як рух у зоні спостереження.

Навіть попри простоту такого підходу на практиці виникає багато проблем пов’язаних із помилковими спрацюваннями. На роботу системи можуть впливати погодні умови зміна освітлення рух дерев тіні або цифрові шуми камери. Особливо сильно дана проблема проявляється у нічний час або при використанні дешевих камер із низькою якістю зображення. Через це система може реагувати навіть у тих випадках коли реального руху у зоні спостереження фактично немає. Саме через це у сучасних системах почали використовувати нейронні мережі які дозволяють не лише визначати факт

зміни кадру а й аналізувати який саме об’єкт знаходиться у зоні спостереження. Подібний підхід дозволяє значно зменшити кількість фальшивих спрацювань та зробити систему більш стабільною. Однією із найпоширеніших моделей для подібних задач є YOLOv8 яка використовується для пошуку та розпізнавання об’єктів у режимі реального часу. Основною перевагою даної моделі є висока швидкість роботи та можливість запуску навіть на звичайному пк без необхідності використання окремих серверів.

Під час аналізу існуючих рішень було встановлено що більшість сучасних систем відеоспостереження є комерційними продуктами які часто потребують додаткових ліцензій окремих AI серверів або стабільного інтернет з’єднання для роботи. Частина таких систем виконує аналіз відео не локально а на сторонніх серверах через що виникає додаткове навантаження на мережу та можуть з’являтись затримки під час передачі відео потоку. Також багато подібних рішень мають обмеження щодо підтримки різних типів камер або VMS систем. Саме тому у даній роботі було вирішено створити окремий локальний застосунок який може працювати поверх будь якої програми для перегляду камер без необхідності втручання у її внутрішню структуру. Основною задачею системи є визначення руху у вибраній області екрана та подальший аналіз об’єктів за допомогою моделі YOLOv8. Також окрему увагу було приділено полігональним зонам детекції які дозволяють обмежувати область аналізу та ігнорувати частини кадру із постійним фоновим рухом.

1.2 Методи виявлення руху

У системах відеоспостереження однією з основних задач є визначення моменту коли у кадрі починається якась активність. Якщо система просто безперервно записує відео то у результаті накопичується дуже велика кількість матеріалу який потім доводиться переглядати вручну. Особливо помітною дана проблема стає при використанні великої кількості камер коли оператор фізично не може постійно контролювати всі потоки одночасно. Саме через це у сучасних системах почали використовувати автоматичну детекцію руху яка дозволяє реагувати лише у момент появи активності в зоні спостереження.

Найпростішим методом визначення руху є порівняння кадрів між собою. Під час роботи система бере поточний кадр та порівнює його із попереднім. Якщо між ними є достатньо велика різниця то це визначається як рух. Фактично система просто перевіряє чи змінилось щось у зоні спостереження за короткий проміжок часу. Подібний метод є досить простим у реалізації та не потребує великої кількості ресурсів саме через це його часто використовують у дешевих системах відеоспостереження або старих програмах для моніторингу камер.

У більшості випадків для спрощення обробки кадри спочатку переводяться у чорно білий формат після чого між ними виконується порівняння пікселів. Якщо кількість змінених пікселів перевищує встановлене значення система починає вважати що у кадрі з'явився рух. Також часто додатково використовуються різні фільтри згладжування для того щоб зменшити вплив дрібних шумів на зображенні. Попри простоту подібного підходу у реальних умовах починає виникати велика кількість проблем. Основною проблемою є помилкові спрацювання які виникають через фактори що не пов'язані із реальною активністю у зоні спостереження. Наприклад система може реагувати на рух дерев через вітер тіні від транспорту зміну освітлення або різкі спалахи світла. У деяких випадках достатньо щоб сонце зайшло за хмару або ввімкнувся ліхтар щоб система почала визначати це як повноцінний рух.

Особливо сильно дана проблема проявляється у нічний час або при використанні дешевих камер із низькою якістю зображення. Через слабке освітлення на кадрі починає з'являтись цифровий шум через що система постійно бачить незначні зміни картинки навіть якщо у кадрі фактично нічого не відбувається. У результаті оператору доводиться переглядати велику кількість непотрібних спрацювань що ускладнює роботу із системою відеоспостереження. Ще однією проблемою звичайного порівняння кадрів є ситуації коли об'єкт рухається дуже повільно або знаходиться далеко від камери. У таких випадках зміни між кадрами можуть бути недостатньо великими через що система просто не побачить рух. Також проблеми можуть

виникати якщо об’єкт частково перекривається іншими елементами кадру або зливається із фоном сцени.

Для покращення стабільності роботи подібних систем почали використовувати інші методи аналізу зображення. Одним із таких методів є віднімання фону. Основна ідея даного підходу полягає у створенні окремого фону сцени із яким у подальшому порівнюється поточний кадр. Якщо певна частина зображення починає сильно відрізнятись від основного фону то система визначає це як рухомий об’єкт. Подібний метод працює стабільніше ніж звичайне порівняння кадрів однак все одно має проблеми при різкій зміні освітлення або погодних умовах. Через велику кількість недоліків класичних методів детекції руху у сучасних системах почали використовувати нейронні мережі. На відміну від звичайного порівняння кадрів подібні алгоритми можуть аналізувати не лише сам факт зміни зображення а й визначати який саме об’єкт знаходиться у кадрі. Це дозволяє значно зменшити кількість непотрібних спрацювань та зробити систему більш стабільною у реальних умовах.

Наприклад нейронна мережа може ігнорувати рух дерев тіней або дрібних тварин але при цьому стабільно визначати появу людини або транспорту у зоні спостереження. Саме це дозволяє значно зменшити навантаження на оператора оскільки система починає реагувати лише на ті об’єкти які дійсно можуть бути важливими для моніторингу. Однією з найбільш популярних моделей для подібних задач є YOLOv8 яка використовується для пошуку та розпізнавання об’єктів у режимі реального часу. Основною перевагою даної моделі є висока швидкість роботи та можливість запуску навіть на звичайному пк без необхідності використання окремих серверів для обробки відео. Модель підтримує готові класи для визначення людей транспорту тварин та інших об’єктів що значно спрощує інтеграцію у системи відеоспостереження.

Попри переваги нейронних мереж постійний запуск моделі на кожному кадрі створює достатньо велике навантаження на систему особливо при

використанні декількох камер одночасно. Через це у даній роботі було вирішено поєднати класичну детекцію руху із використанням нейронної мережі. Спочатку система визначає появу активності за допомогою звичайного аналізу кадру а вже після цього запускається модель YOLOv8 для аналізу об’єктів у зоні спостереження. Подібний підхід дозволяє значно зменшити навантаження на систему та зробити роботу застосунку стабільнішою при тривалому використанні. Також додатково були реалізовані полігональні області детекції які дозволяють обмежувати частину кадру у межах якої буде виконуватись аналіз руху. Це особливо важливо у випадках коли частина сцени містить постійний фоновий рух який не потребує аналізу наприклад дорогу дерева або територію за межами самого об’єкта спостереження.

Таким чином використання декількох методів визначення руху одночасно дозволяє зробити систему більш стабільною та значно зменшити кількість помилкових спрацювань. Поєднання звичайного аналізу кадрів із використанням нейронної мережі дозволяє отримати достатньо швидку роботу системи без необхідності постійно навантажувати обладнання аналізом кожного кадру відео потоку.

1.3 Нейронні мережі та використання моделі YOLOv8

Через велику кількість проблем які виникають у звичайних системах детекції руху з часом почали використовувати більш сучасні методи аналізу відео потоку. Основною проблемою класичної детекції є те що система реагує практично на будь яку зміну кадру незалежно від того що саме знаходиться у зоні спостереження. У реальних умовах це створює велику кількість помилкових спрацювань особливо якщо камери встановлені на відкритій території де постійно присутній рух дерев зміна освітлення погодні умови або тіні від різних об’єктів. Саме через це виникла потреба у використанні алгоритмів які можуть не просто бачити зміну картинки а визначати що саме знаходиться у кадрі. Для подібних задач почали використовувати нейронні мережі які навчаються на великій кількості зображень та відео після чого

можуть визначати різні типи об’єктів. Якщо пояснювати простіше то під час навчання модель бачить дуже багато прикладів людей машин тварин та інших об’єктів через що у подальшому вже може приблизно розуміти як вони виглядають навіть якщо зображення має погану якість або сам об’єкт частково перекритий іншими елементами кадру. Саме через це нейронні мережі почали активно використовуватись у сучасних системах відеоспостереження.

У більшості сучасних систем моніторингу нейронні мережі використовуються для визначення людей транспорту або інших важливих об’єктів у зоні спостереження. Подібний підхід дозволяє значно зменшити кількість непотрібних спрацювань оскільки система може ігнорувати рух дерев тіней або дрібних тварин але при цьому стабільно реагувати на появу людини у кадрі. У результаті оператору вже не потрібно переглядати велику кількість зайвих спрацювань як це відбувається у звичайних системах детекції руху. Однією із найпоширеніших моделей для подібних задач є YOLO яка використовується для пошуку та розпізнавання об’єктів у режимі реального часу. Назва YOLO розшифровується як You Only Look Once. Основна особливість даної моделі полягає у тому що аналіз кадру виконується за один прохід через нейронну мережу завдяки чому вдається отримати достатньо високу швидкість роботи. Саме через це модель часто використовується у системах відеоспостереження де важлива швидка реакція на події та можливість працювати у реальному часі без великих затримок.

У даній роботі використовується модель YOLOv8 яка є однією із новіших версій даної серії моделей. Основною перевагою даної версії є хороша швидкість роботи при достатньо нормальній точності визначення об’єктів. Також важливою перевагою є підтримка готових класів для визначення людей транспорту тварин та інших об’єктів через що немає необхідності самостійно навчати модель з нуля. Це значно спрощує процес інтеграції нейронної мережі у програмний застосунок та дозволяє швидше реалізувати основний функціонал системи. Під час роботи застосунку модель отримує поточний кадр після чого виконує його аналіз та визначає об’єкти які

знаходяться у зоні спостереження. Після завершення аналізу система створює спеціальні рамки навколо знайдених об’єктів та передає інформацію про їх тип. Таким чином система може визначати чи знаходиться у кадрі людина транспорт або інший об’єкт який є важливим для моніторингу. Попри переваги нейронних мереж подібні моделі також мають свої недоліки. Основною проблемою є достатньо велике навантаження на систему під час аналізу відео потоку. Якщо запускати модель постійно на кожному кадрі можуть виникати просадки fps затримки під час обробки відео або збільшення навантаження на процесор та відеокарту. Особливо помітною дана проблема стає при використанні декількох камер одночасно або при роботі із високою роздільною здатністю відео.

Також швидкість роботи напряму залежить від самого обладнання. Якщо використовується слабкий процесор або відсутня відеокарта то швидкість аналізу може бути недостатньою для стабільної роботи у режимі реального часу. У деяких випадках можуть виникати проблеми і з самим визначенням об’єктів. Наприклад при поганому освітленні або якщо об’єкт знаходиться далеко від камери модель може неправильно визначити його тип або взагалі пропустити появу об’єкта у кадрі. Саме через це у даному застосунку було вирішено не запускати модель постійно а використовувати її лише після того як система вже помітила рух у зоні детекції. Спочатку виконується звичайне визначення активності у кадрі за допомогою аналізу змін між кадрами а вже після цього запускається модель YOLOv8 для аналізу об’єктів. Подібний підхід дозволяє значно зменшити навантаження на систему та зробити роботу застосунку стабільнішою при тривалому використанні.

Також додатково були реалізовані полігональні області детекції які дозволяють обмежувати частину кадру у межах якої буде виконуватись аналіз. Подібний підхід є важливим через те що у багатьох випадках частина сцени може містити постійний фоновий рух який не є важливим для моніторингу наприклад дорогу дерева або територію за межами самого об’єкта

спостереження. Завдяки цьому система може аналізувати лише потрібну область та не реагувати на зайві зміни кадру.

Таким чином використання моделі YOLOv8 у поєднанні із класичною детекцією руху дозволяє зробити систему більш стабільною та значно зменшити кількість помилкових спрацювань. Подібний підхід дозволяє отримати достатньо швидку роботу застосунку без необхідності постійно навантажувати систему аналізом кожного кадру відео потоку.

1.4 Аналіз бібліотек та засобів розробки

Для реалізації програмного застосунку було вирішено використовувати мову програмування Python оскільки вона має велику кількість готових бібліотек для роботи із відео обробкою кадрів створенням графічного інтерфейсу та інтеграцією нейронних мереж. Також важливою перевагою Python є простота самої мови через що можна значно швидше реалізувати потрібний функціонал без необхідності писати велику кількість складного коду. Ще однією причиною вибору Python стала хороша сумісність із бібліотеками які використовуються для роботи із моделями YOLO та аналізом відео потоку у режимі реального часу.

Однією з основних бібліотек у даному застосунку є OpenCV яка використовується для роботи із зображенням та обробкою кадрів. Саме через OpenCV виконується аналіз відео потоку порівняння кадрів визначення руху обробка зображення та частина допоміжних функцій пов’язаних із детекцією. Також бібліотека дозволяє працювати із контурами фільтрами та різними алгоритмами обробки кадру які використовуються під час аналізу активності у зоні спостереження. Основною перевагою OpenCV є висока швидкість роботи та велика кількість готових інструментів через що не потрібно реалізовувати подібні алгоритми повністю вручну.

Для створення графічного інтерфейсу застосунку використовується бібліотека PyQt5. Дана бібліотека дозволяє створювати повноцінні графічні програми із кнопками меню панелями налаштувань та іншими елементами керування. Через PyQt5[16] було реалізовано головне вікно застосунку

налаштування області моніторингу кнопки запуску та зупинки детекції а також взаємодію користувача із полігональними зонами. Однією із важливих переваг даної бібліотеки є можливість створення прозорих вікон які можуть працювати поверх інших застосунків. Саме ця функція використовується у даному проєкті оскільки система накладається поверх програми для перегляду камер та виконує аналіз вибраної області екрана.

Для отримання зображення із вибраної області екрана використовується бібліотека MSS. Основною задачею даної бібліотеки є швидке зчитування кадрів із екрана без створення великого навантаження на систему. Подібний підхід був вибраний через те що застосунок не підключається напряму до самих камер а працює поверх уже відкритого відео потоку. Фактично система просто аналізує частину екрана яку вибирає користувач після чого виконує детекцію руху та аналіз об’єктів на основі отриманого зображення. Це дозволяє використовувати застосунок практично із будь якою програмою для відеоспостереження без необхідності інтеграції у конкретні VMS системи або прямої роботи із потоками камер.

Для інтеграції нейронної мережі використовується бібліотека Ultralytics[13] через яку виконується робота із моделлю YOLOv8. Саме через дану бібліотеку виконується завантаження моделі аналіз кадрів та визначення об’єктів у зоні спостереження. Основною перевагою Ultralytics є те що бібліотека вже має готову реалізацію роботи із моделлю через що не потрібно вручну налаштовувати всі процеси пов’язані із запуском нейронної мережі. Також бібліотека дозволяє достатньо просто працювати із результатами детекції та отримувати інформацію про знайдені об’єкти їх координати та тип.

Для створення коротких анімованих записів використовується бібліотека imageio[9]. Після того як система помічає рух кадри починають тимчасово зберігатись а далі із них автоматично формується гіфка яку можна переглянути пізніше. Подібний підхід дозволяє не записувати постійно великі відео файли а зберігати лише короткі моменти активності у зоні

спостереження. Це значно зменшує обсяг зайнятого місця на накопичувачі та спрощує подальший перегляд подій.

Також у застосунку використовуються стандартні бібліотеки Python для роботи із файлами потоками системними функціями та багатопоточністю. Частина із них використовується для створення директорій збереження роботи із часом або оптимізації окремих процесів під час виконання програми. Для забезпечення стабільної роботи інтерфейсу окремі задачі були винесені у додаткові потоки що дозволяє уникнути зависання програми під час аналізу відео потоку або роботи нейронної мережі.

Під час вибору засобів розробки основна увага приділялась швидкості роботи простоті інтеграції та можливості локального запуску застосунку без використання сторонніх серверів. Використання готових бібліотек дозволило значно спростити розробку системи та реалізувати основний функціонал без необхідності створювати всі алгоритми повністю з нуля. Таким чином поєднання Python OpenCV PyQt5 MSS та Ultralytics дозволило створити локальний застосунок для моніторингу відео потоку який підтримує визначення руху роботу із полігональними зонами інтеграцію нейронної мережі та автоматичне створення гіфок після виявлення активності у зоні спостереження.

1.5 Постановка задачі

Після аналізу сучасних систем відеоспостереження та методів виявлення руху було встановлено що основною проблемою подібних рішень є велика кількість непотрібних спрацювань та складність постійного моніторингу великої кількості камер одночасно. У більшості випадків оператор змушений постійно переглядати відео потік навіть у ті моменти коли у кадрі фактично нічого не відбувається. Також значною проблемою є звичайні системи детекції руху які реагують практично на будь яку зміну кадру включаючи погодні умови тіні зміну освітлення або цифрові шуми камери. Через це виникає велика кількість помилкових спрацювань які ускладнюють роботу із системою та створюють додаткове навантаження на оператора. На основі проведеного

аналізу було поставлено задачу створити локальний програмний застосунок для моніторингу відео потоку який зможе автоматично визначати рух у вибраній області екрана та додатково аналізувати об’єкти за допомогою моделі YOLOv8. Основною задачею системи є не просто визначення зміни кадру а зменшення кількості фальшивих спрацювань за рахунок аналізу того що саме знаходиться у зоні детекції.

Застосунок повинен працювати у режимі реального часу та накладатись поверх інших програм для перегляду камер без необхідності інтеграції у їх внутрішню структуру. Користувач повинен мати можливість самостійно змінювати положення області моніторингу її розмір та межі детекції залежно від того яка саме частина відео потоку потребує аналізу. Також система повинна підтримувати швидкий запуск та зупинку моніторингу без необхідності повторного налаштування всієї області спостереження. Однією із основних задач стало створення полігональних зон детекції які дозволяють обмежувати область у межах якої буде виконуватись аналіз руху. Подібний підхід є важливим через те що у багатьох випадках частина кадру може містити постійний рух який не потребує аналізу наприклад дорогу дерева або територію за межами самого об’єкта спостереження. Саме тому користувач повинен мати можливість самостійно створювати потрібну форму зони детекції та змінювати її відповідно до власних потреб.

Також важливою задачею є оптимізація роботи системи оскільки постійний запуск нейронної мережі на кожному кадрі створює достатньо велике навантаження на систему особливо при тривалому використанні або роботі із декількома камерами одночасно. Через це у даному застосунку було вирішено спочатку використовувати звичайне визначення активності у кадрі а вже після появи руху запускати модель YOLOv8 для аналізу об’єктів. Подібний підхід дозволяє значно зменшити навантаження на систему та зробити роботу застосунку стабільнішою. Додатково була поставлена задача реалізувати автоматичне створення гіфок після виявлення руху у зоні спостереження. Система повинна зберігати короткі моменти активності для

подальшого перегляду без необхідності постійного запису великої кількості відео. Це дозволяє значно спростити пошук потрібних моментів та зменшити обсяг даних які потрібно зберігати на комп’ютері.

Під час розробки також необхідно врахувати стабільність роботи інтерфейсу швидкість обробки кадрів та можливість тривалої роботи застосунку без критичного навантаження на систему. Окрему увагу потрібно приділити коректній роботі прозорого вікна та взаємодії користувача із полігональними зонами оскільки саме через інтерфейс виконується основна частина налаштування системи. Таким чином у даній роботі необхідно реалізувати програмний застосунок який буде виконувати аналіз вибраної області екрана визначати появу руху запускати модель YOLOv8 для аналізу об’єктів підтримувати полігональні зони детекції та автоматично створювати гіфки після появи активності. У результаті система повинна зменшити кількість непотрібних спрацювань покращити ефективність моніторингу та спростити роботу із великою кількістю камер відеоспостереження

РОЗДІЛ ІІ ВИБІР ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ

У даному розділі розглядаються основні програмні засоби та бібліотеки які були використані для реалізації системи моніторингу відеопотоку із функціями детекції руху та аналізу об’єктів. Під час вибору технологій основна увага приділялась швидкодії застосунку можливості роботи у режимі реального часу простоті інтеграції різних компонентів системи та мінімальному навантаженню на комп’ютер під час тривалого моніторингу.

Для реалізації застосунку було використано мову програмування Python бібліотеку PyQt5[5] для створення графічного інтерфейсу OpenCV для обробки кадрів MSS для захоплення області екрана а також модель YOLOv8 для визначення об’єктів у зоні спостереження. Окремо розглядається вибір засобів для запису та збереження зафіксованих подій у форматі GIF.

У розділі виконано порівняння основних технологій та обґрунтовано причини вибору саме тих інструментів які були використані під час розробки системи.

2.1 Вибір середовища розробки та засобів створення інтерфейсу

Під час розробки системи моніторингу відеопотоку основна увага приділялась швидкості створення застосунку простоті інтеграції бібліотек комп’ютерного зору та можливості роботи у режимі реального часу. Саме через це для реалізації програмної частини було обрано мову програмування Python яка є однією з найбільш популярних мов у сфері обробки зображень машинного навчання та автоматизації. Однією з основних переваг Python є наявність великої кількості готових бібліотек для роботи із графічним інтерфейсом аналізом відео та нейронними мережами. Також Python дозволяє значно швидше реалізовувати та тестувати різні компоненти системи у порівнянні із мовами нижчого рівня такими як C++.Для оцінки доцільності використання Python було виконано порівняння із іншими мовами програмування які також можуть використовуватись для створення систем відеоспостереження.

Таблиця Порівняння мов програмування

Мова	Переваги	Недоліки
Python	Велика кількість AI та CV бібліотек, швидка розробка	Нижча продуктивність
C++	Висока швидкодія	Складніша розробка
Java	Кросплатформеність	Менша кількість бібліотек для CV
C#	Зручна робота із Windows	Складніша інтеграція AI

Після вибору мови програмування наступним етапом став вибір засобу для створення графічного інтерфейсу. Основною вимогою до інтерфейсу була підтримка прозорих overlay вікон які можуть працювати поверх інших програм для перегляду камер. Також важливою вимогою була підтримка обробки подій миші малювання полігонів та можливість динамічної зміни розміру області моніторингу. Для реалізації графічного інтерфейсу було використано бібліотеку PyQt5. Даний фреймворк дозволяє створювати desktop застосунки із підтримкою прозорості роботи поверх інших вікон та великою кількістю інструментів для побудови GUI.

Створення overlay вікна у застосунку виконується за допомогою наступних параметрів[6].

```
self.setWindowFlags(
   Qt.FramelessWindowHint |
   Qt.WindowStaysOnTopHint … )
self.setAttribute(
   Qt.WA_TranslucentBackground … )
```

Дані параметри дозволяють прибрати стандартну рамку системного вікна та забезпечують прозорий фон overlay області що використовується для

моніторингу відеопотоку.

Для оцінки доцільності використання PyQt5 було виконано порівняння із іншими популярними GUI фреймворками.

Таблиця Порівняння GUI фреймворків

Фреймворк	Переваги	Недоліки
PyQt5	Overlay вікна, прозорість, велика кількість GUI компонентів	Більший розмір бібліотеки
Tkinter	Простота використання	Обмежений інтерфейс
Kivy	Підтримка touch інтерфейсів	Складніша desktop інтеграція
Electron	Сучасний дизайн	Високе навантаження на систему

2.2 Вибір бібліотек та інструментів розробки

Для реалізації системи моніторингу та детекції руху було обрано набір бібліотек і програмних засобів, які забезпечують роботу графічного інтерфейсу, аналіз відеопотоку, обробку зображень та інтеграцію нейронної мережі YOLOv8. Основним критерієм під час вибору була можливість створення локального застосунку, здатного працювати у режимі реального часу без необхідності використання сторонніх серверів або хмарних сервісів. Основною мовою розробки було обрано Python. Дана мова має велику кількість бібліотек для роботи із комп’ютерним зором, машинним навчанням та графічними інтерфейсами. Також Python дозволяє швидко реалізовувати та тестувати функціонал без значних витрат часу на низькорівневу оптимізацію. Для створення графічного інтерфейсу застосунку використовується бібліотека

PyQt5. Вона дозволяє створювати desktop застосунки із підтримкою прозорих overlay вікон, обробки подій миші та клавіатури, а також різних елементів керування. Саме PyQt5 використовується для створення області моніторингу, toolbar, sidebar та системи полігональних зон. Для роботи із відеокадрами та обробки зображень використовується бібліотека OpenCV. Вона забезпечує виконання основних операцій аналізу кадру включаючи переведення зображення у grayscale формат, розмиття Gaussian Blur, threshold обробку та пошук контурів руху. Також OpenCV використовується для роботи із координатами боксів та попередньої підготовки кадрів перед передачею у модель YOLOv8. Захоплення області екрана реалізоване через бібліотеку MSS. Її основною перевагою є достатньо висока швидкість роботи та низьке навантаження на систему у порівнянні із деякими іншими методами screen capture. Саме MSS дозволяє отримувати кадри із overlay області у режимі реального часу. Для реалізації системи детекції об’єктів використовується модель YOLOv8 із бібліотеки Ultralytics[15]. Дана модель забезпечує швидке визначення людей, транспорту та інших об’єктів у кадрі. У проєкті використовується версія YOLOv8n, яка є найбільш легкою серед моделей серії YOLOv8 та створює менше навантаження на систему. Для створення GIF-анімацій використовується бібліотека imageio. Вона дозволяє формувати короткі анімовані фрагменти після виявлення руху без використання додаткових відеокодеків.

Бібліотека	Призначення
PyQt5	Створення графічного інтерфейсу
OpenCV	Обробка зображень та аналіз руху
MSS	Захоплення області екрана
Ultralytics YOLOv8	Детекція об’єктів
NumPy [2]	Робота із масивами даних
imageio	Створення GIF-анімацій

Також під час розробки використовувалось віртуальне середовище Python .venv, яке дозволяє ізолювати бібліотеки проєкту та уникати конфліктів між різними версіями залежностей.

Для встановлення всіх необхідних компонентів використовується файл requirements.txt, у якому зберігається список бібліотек та їх версій. Це дозволяє швидко переносити проєкт на інші комп'ютери та спрощує повторне налаштування середовища розробки.

Інструмент	Основні переваги
Python	Простота розробки та велика кількість бібліотек
PyQt5	Підтримка overlay інтерфейсів
OpenCV	Висока швидкість обробки кадрів
MSS	Швидке захоплення екрана
YOLOv8n	Баланс між швидкістю та точністю
imageio	Просте створення GIF-файлів

Таким чином обраний набір бібліотек та інструментів дозволив реалізувати локальний застосунок для моніторингу відеопотоку із підтримкою детекції руху, аналізу об'єктів та автоматичного створення GIF-анімацій у режимі реального часу.

2.3 Вибір моделі детекції об'єктів та алгоритму виявлення руху

Однією з основних задач під час розробки системи було забезпечення достатньо швидкого та стабільного визначення активності у відеопотоці. Для цього необхідно було обрати алгоритм виявлення руху та модель детекції об'єктів, які могли б працювати у режимі реального часу без надмірного навантаження на систему. Для визначення руху було вирішено використовувати метод порівняння сусідніх кадрів. Основною перевагою такого підходу є простота реалізації та висока швидкість роботи. Алгоритм не

потребує складного навчання або використання великих обчислювальних ресурсів, тому добре підходить для локального застосунку. Принцип роботи методу полягає у порівнянні поточного кадру із попереднім. Якщо між кадрами з'являються значні зміни, система визначає це як рух у зоні спостереження. Для зменшення кількості шумів використовується попередня обробка кадру через grayscale та Gaussian Blur.

Перевага	Опис
Висока швидкість	Аналіз виконується у режимі реального часу
Простота реалізації	Не потребує складного налаштування
Низьке навантаження	Працює навіть на слабких системах
Швидка реакція	Майже миттєве визначення руху

Попри простоту алгоритму, звичайне визначення руху має ряд недоліків. Система може реагувати на зміну освітлення, погодні умови, цифрові шуми камери або рух неважливих об'єктів. Саме через це було вирішено додатково інтегрувати систему детекції об'єктів.

Для аналізу об'єктів було обрано модель YOLOv8. Дана модель належить до сучасних систем комп'ютерного зору та дозволяє швидко визначати об'єкти у кадрі із достатньо високою точністю. Основною причиною вибору YOLOv8 стала її швидкодія, що є важливим фактором для систем моніторингу реального часу.

У проєкті використовується версія YOLOv8n. Дана модель є найменшою серед моделей серії YOLOv8 та створює значно менше навантаження на систему у порівнянні із більшими версіями.

Модель	Швидкодія	Точність	Навантаження
YOLOv8n	Висока	Середня	Низьке
YOLOv8s	Середня	Низька	Середнє

YOLOv8m	Нижча	Висока	Високе

Під час тестування було встановлено, що використання більших моделей може покращити точність визначення об’єктів, проте одночасно збільшується навантаження на процесор та відеокарту. Через це для локального застосунку було обрано саме YOLOv8n як компроміс між продуктивністю та якістю детекції.

Також важливою особливістю реалізації є те, що модель YOLOv8 запускається не постійно, а лише після виявлення руху. Подібний підхід дозволяє значно зменшити кількість запусків нейронної мережі та покращити загальну продуктивність системи.

Етап	Опис
Захоплення кадру	Отримання області екрана через MSS
Аналіз руху	Порівняння поточного та попереднього кадру
Фільтрація шумів	Blur та threshold обробка
Детекція об’єктів	Запуск YOLOv8 після виявлення руху
Перевірка полігону	Аналіз положення об’єкта у зоні
Реакція системи	Відображення бокса та запис GIF

Таким чином поєднання алгоритму визначення руху та моделі YOLOv8 дозволило створити систему, яка може швидко реагувати на появу активності у кадрі та одночасно зменшувати кількість фальшивих спрацювань завдяки аналізу типів об’єктів у зоні спостереження.

РОЗДІЛ ІІІ РЕАЛІЗАЦІЯ СИСТЕМИ

Після проведення аналізу існуючих систем відеоспостереження та методів детекції руху було прийнято рішення створити власний локальний застосунок для моніторингу відео потоку який поєднує звичайне визначення руху із використанням моделі YOLOv8. Основною задачею під час розробки було створення системи яка могла б працювати поверх інших програм для перегляду камер без необхідності окремого підключення до самих потоків відео або використання сторонніх серверів для аналізу зображення. Також важливим завданням була мінімізація навантаження на систему оскільки застосунок повинен стабільно працювати у режимі реального часу протягом тривалого часу без значних просадок продуктивності.

Під час проєктування системи основна увага приділялась простоті роботи самого застосунку та можливості швидко взаємодіяти із зоною моніторингу. Через це було вирішено використовувати прозоре overlay вікно яке накладається поверх інших програм та дозволяє користувачу самостійно вибирати область у межах якої буде виконуватись аналіз руху. Подібний підхід дозволяє використовувати систему практично із будь якими програмами для відеоспостереження незалежно від типу камер або способу передачі відео потоку.

Під час роботи застосунок отримує кадри із вибраної області екрана після чого виконується аналіз зміни між кадрами для визначення активності у зоні спостереження. Якщо система помічає рух запускається додатковий аналіз об’єктів за допомогою моделі YOLOv8 яка визначає чи знаходяться у кадрі люди транспорт або інші важливі об’єкти. Після підтвердження активності система може автоматично створювати гіфки із моментами руху для подальшого перегляду.

Також під час реалізації було важливо забезпечити можливість обмеження області детекції оскільки у реальних умовах частина кадру може містити постійний рух який не є важливим для моніторингу. Саме через це у застосунку були реалізовані полігональні області які дозволяють користувачу

самостійно вибирати зону у межах якої буде виконуватись аналіз кадру. Завдяки цьому кількість фальшивих спрацювань суттєво зменшується.

У даному розділі буде розглянута архітектура застосунку структура проєкту реалізація графічного інтерфейсу алгоритм визначення руху інтеграція моделі YOLOv8 та система автоматичного створення гіфок після виявлення активності у зоні спостереження.

3.1 Архітектура застосунку

Під час розробки застосунку основною задачею було створення системи яка могла б працювати поверх інших програм для перегляду камер та не потребувала прямого підключення до самих відео потоків. Через це було вирішено використовувати overlay підхід при якому застосунок створює прозоре вікно поверх екрана та виконує аналіз лише вибраної області. Подібний підхід забезпечує універсальність застосунку та можливість роботи з різними програмами відеоспостереження.

Після запуску застосунок створює головне прозоре вікно яке розташовується поверх інших програм. Для цього використовуються параметри Qt які прибирають стандартну рамку вікна та дозволяють залишати його поверх інших застосунків.

```
self.setWindowFlags(Qt.FramelessWindowHint
Qt.WindowStaysOnTopHint)
self.setAttribute(Qt.WA_TranslucentBackground)
```

Даний фрагмент відповідає за створення overlay вікна без стандартної рамки системи. Також застосунок отримує прозорий фон що дозволяє користувачу бачити відеопотік який знаходиться під самим застосунком.

Основою архітектури є клас SmartVMS у якому знаходиться вся основна логіка роботи системи. Саме тут виконується ініціалізація інтерфейсу запуск моделі YOLO створення таймера для аналізу кадрів та збереження всіх параметрів які використовуються під час роботи програми.

```
class SmartVMS(QWidget):
    def __init__(self):
```

```
super().__init__()
```

Під час запуску також створюється окрема папка для запису гіфок після виявлення руху.

```
os.makedirs("gifs", exist_ok=True)
```

Після ініціалізації інтерфейсу застосунок запускає систему отримання кадрів із екрана через бібліотеку MSS[8] а також створює таймер який відповідає за постійний запуск аналізу кадрів.

```
self.sct = mss.mss()
self.timer = QTimer()
self.timer.timeout.connect(self.process_frame)
```

Сам аналіз виконується у функції process_frame() яка запускається через певний проміжок часу та постійно перевіряє область моніторингу на наявність руху. Для запуску системи використовується окрема кнопка Start після натискання якої запускається таймер обробки кадрів.

```
def start_detection(self):
   self.timer.start(50)
```

Під час роботи система отримує координати області моніторингу та формує область екрана яку потрібно аналізувати.

```
geom = self.interaction_area.geometry()
gp = self.interaction_area.mapToGlobal(QPoint(0,0))
bbox = { "left": gp.x(),
         "top": gp.y(),
         "width": geom.width(),
         "height": geom.height() …}
```

Після цього через MSS виконується захоплення поточного кадру із екрана.

```
shot = self.sct.grab(bbox)
frame = cv2.cvtColor(np.array(shot), cv2.COLOR_BGRA2BGR)
```

Далі система переводить кадр у чорно білий формат та виконує розмиття зображення для зменшення шумів під час аналізу.

```
gray = cv2.GaussianBlur(
   cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY),
   (15, 15),  0 … )
```

Після попередньої обробки виконується порівняння поточного кадру із попереднім для визначення активності у зоні спостереження.

```
delta = cv2.absdiff(self.prev_frame, gray)
thresh = cv2.threshold(delta, 30, 255, cv2.THRESH_BINARY)[1]
```

Якщо система бачить достатню кількість змін у кадрі запускається додатковий аналіз через модель YOLOv8.

```
res = self.model.predict(
   frame, conf=0.45, verbose=False,
   stream=True … )
```

Під час запуску модель аналізує кадр та повертає знайдені обєкти після чого система перевіряє чи входять вони у список дозволених класів.

```
CLASS_MAPPING = { … }
```

Подібний підхід дозволяє відфільтровувати непотрібні класи та залишати лише ті обєкти які потрібні для моніторингу. Після цього система формує координати боксів навколо знайдених обєктів та перевіряє чи знаходиться центр обєкта у межах полігональної області.

```
if not self.track_zones or any(
   cv2.pointPolygonTest(
      np.array([[p.x(), p.y()] for p in z]),
      (cx, cy),  False
   ) >= 0 for z in self.track_zones … ):
```

Саме через це система може ігнорувати рух за межами основної області моніторингу та реагувати лише на події всередині полігону.

Окремою частиною архітектури є система створення гіфок. Після виявлення руху кадри тимчасово зберігаються у буфер після чого автоматично формується коротка анімація.

```
if len(self.gif_frames) < 60:
```

self.gif_frames.append(frame.copy())

Після завершення руху система автоматично створює файл у папці gifs.

imageio.mimsave(

path, [cv2.cvtColor(f, cv2.COLOR_BGR2RGB)

for f in self.gif_frames], fps=12 …)

Для відображення боксів зон детекції та інших елементів інтерфейсу використовується функція paintEvent() у якій через QPainter малюються рамки полігони та результати детекції поверх прозорого вікна.

p.drawRect(x1 + ox, y1 + oy, x2 - x1, y2 - y1)

Таким чином архітектура застосунку побудована таким чином щоб поєднати роботу графічного інтерфейсу аналіз кадрів запуск нейронної мережі систему полігональних зон та автоматичне створення гіфок у межах одного локального застосунку який може працювати у режимі реального часу поверх інших програм для відеоспостереження.

3.2 Формування структури проєкту

Під час розробки застосунку була сформована проста структура проєкту яка містить основні файли необхідні для роботи системи детекції руху інтеграції моделі YOLOv8 та створення графічного інтерфейсу. Основна частина логіки реалізована у одному головному файлі що дозволяє спростити процес тестування запуску та внесення змін під час розробки. Подібний підхід був вибраний через те що сам застосунок не є дуже великим проєктом із великою кількістю окремих модулів а більшість функцій тісно повязані між собою та постійно взаємодіють у процесі роботи системи.

У структурі проєкту присутні файли моделі бібліотек папка для збереження гіфок а також окреме віртуальне середовище яке використовується для запуску програми та встановлення необхідних залежностей. Загальний вигляд структури проєкту представлений на рисунку.

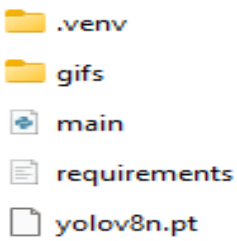

Рис 1. вміст папки

Основним файлом застосунку є main.py у якому знаходиться практично вся логіка роботи системи. Саме у цьому файлі виконується створення графічного інтерфейсу запуск overlay вікна обробка кадрів аналіз руху інтеграція YOLOv8 робота із полігональними зонами та запис гіфок після виявлення активності у зоні спостереження. Також у даному файлі реалізовані всі основні функції для взаємодії користувача із застосунком включаючи зміну розміру області моніторингу переміщення вікна запуск та зупинку детекції.

Основна логіка роботи застосунку починається зі створення головного класу який відповідає за роботу всіх компонентів системи.

class SmartVMS(QWidget):

Саме у цьому класі виконуються ініціалізація інтерфейсу запуск таймерів завантаження моделі YOLO та створення всіх основних елементів програми.

Окремо у структурі проєкту знаходиться папка gifs яка використовується для автоматичного збереження коротких гіфок після виявлення руху. Під час роботи застосунок формує тимчасовий буфер кадрів після чого автоматично створює анімований файл який записується у дану папку. Подібний підхід дозволяє зберігати лише потрібні моменти активності без необхідності постійного запису великої кількості відео.

Створення папки виконується автоматично під час запуску програми.

os.makedirs("gifs", exist_ok=True)

Також у структурі проєкту присутній файл yolov8n.pt який містить ваги нейронної мережі YOLOv8. Саме через даний файл виконується завантаження

моделі для подальшого аналізу кадрів та визначення обєктів у зоні спостереження. Було використано версію yolov8n оскільки вона має достатньо хорошу швидкість роботи та створює менше навантаження на систему у порівнянні із більшими моделями.

Завантаження моделі виконується під час запуску застосунку.

self.model = YOLO("yolov8n.pt")

Для встановлення необхідних бібліотек у проєкті використовується файл requirements.txt. У ньому знаходиться список основних залежностей які потрібні для запуску програми включаючи бібліотеки для роботи із графічним інтерфейсом обробкою відео та інтеграцією нейронної мережі. Подібний підхід дозволяє швидко встановити всі необхідні бібліотеки на іншому компютері без ручного налаштування кожного компонента окремо.

У файлі знаходяться основні бібліотеки які використовуються у застосунку.

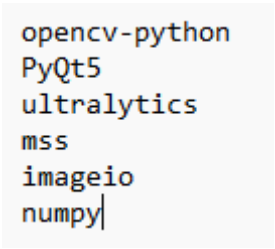

Рис.2 наповнення requirements.txt

Для запуску застосунку також використовується папка .venv яка містить окреме віртуальне середовище Python. Основною задачею такого середовища є ізоляція бібліотек проєкту від інших програм які можуть бути встановлені у системі. Це дозволяє уникнути проблем із несумісністю версій бібліотек та спрощує перенесення проєкту на інші пристрої.

Таким чином сформована структура проєкту дозволяє організувати всі основні компоненти застосунку у межах однієї системи та забезпечує простий запуск програми швидке внесення змін під час розробки та стабільну роботу

всіх основних модулів повязаних із детекцією руху аналізом обєктів та роботою графічного інтерфейсу.

3.3 Реалізація графічного інтерфейсу

Однією з основних частин застосунку є графічний інтерфейс оскільки саме через нього користувач взаємодіє із системою моніторингу змінює область детекції запускає аналіз відео потоку та керує допоміжними функціями. Під час розробки інтерфейсу основна увага приділялась простоті використання та можливості швидко взаємодіяти із зоною моніторингу без необхідності відкривати велику кількість додаткових вікон або меню. Також важливим завданням було створення прозорого overlay інтерфейсу який міг би працювати поверх інших програм для перегляду камер та не перекривати сам відеопотік.

Для створення графічного інтерфейсу використовується бібліотека PyQt5 яка дозволяє створювати повноцінні desktop застосунки із підтримкою прозорих вікон кнопок меню та різних елементів керування. Основна частина інтерфейсу реалізована у межах головного класу SmartVMS де створюються всі основні елементи системи та обробляються дії користувача.

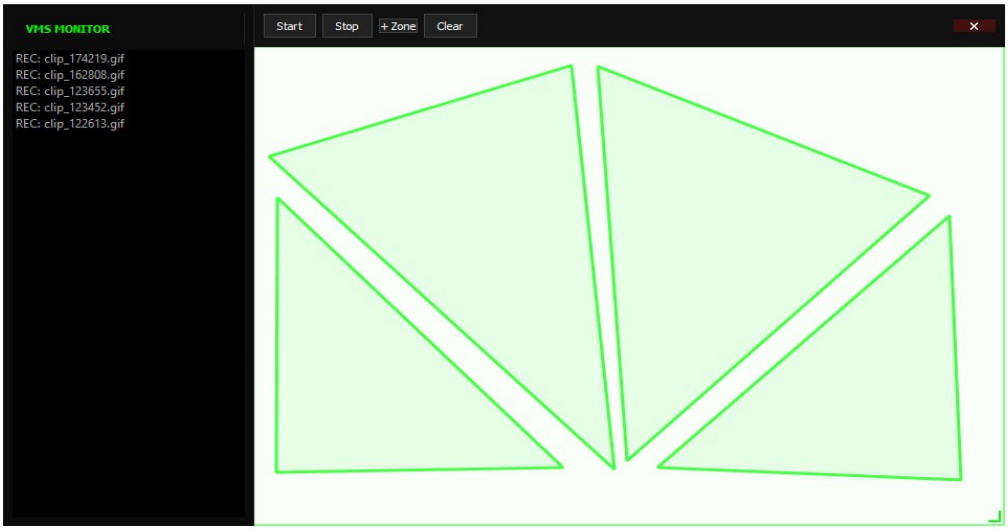

Рис. 3 GUI

Під час запуску застосунок створює прозоре вікно поверх інших програм.

self.setWindowFlags(

Qt.FramelessWindowHint

Qt.WindowStaysOnTopHint …)

Даний фрагмент дозволяє прибрати стандартну рамку системного вікна та залишати застосунок поверх інших програм. Це потрібно для того щоб область моніторингу могла працювати безпосередньо поверх відкритого відео потоку.

Для підтримки прозорого фону використовується окремий параметр PyQt5.

self.setAttribute(Qt.WA_TranslucentBackground)

Завдяки цьому користувач бачить сам відеопотік під overlay вікном а система при цьому може малювати бокси полігони та інші елементи поверх кадру.

Інтерфейс застосунку складається із декількох основних частин серед яких toolbar sidebar область моніторингу та система редагування полігонів. Кожен із цих елементів виконує окрему задачу та відповідає за певну частину взаємодії користувача із системою.

3.3.1 Toolbar

Toolbar використовується для швидкого доступу до основних функцій застосунку. Саме тут знаходяться кнопки запуску та зупинки детекції а також елементи керування допоміжними режимами роботи системи. Основною задачею toolbar є забезпечення швидкого доступу до функцій без необхідності відкривати окремі меню або додаткові вікна.

Для створення кнопок використовується стандартний функціонал PyQt5.

self.start_btn = QPushButton("Start")

self.stop_btn = QPushButton("Stop")

Після натискання кнопок викликаються відповідні функції запуску або зупинки системи аналізу кадрів.

self.start_btn.clicked.connect(self.start_detection)

self.stop_btn.clicked.connect(self.stop_detection)

Подібний підхід дозволяє швидко керувати системою детекції без перезапуску самого застосунку.

3.3.2 Transparent monitoring area

Основною частиною інтерфейсу є прозора область моніторингу у межах якої виконується аналіз відеопотоку. Саме ця область накладається поверх програми із камерами та використовується для захоплення кадрів через MSS.

Під час роботи користувач може змінювати положення області моніторингу а також її розмір залежно від того яка саме частина відеопотоку потребує аналізу.

Для захоплення координат області використовується геометрія вікна.

geom = self.interaction_area.geometry()

Після цього координати передаються у систему захоплення екрана через MSS для отримання поточного кадру.

Однією з важливих особливостей є те що overlay область не блокує перегляд самого відео оскільки основна частина вікна залишається прозорою а система відображає лише службові елементи інтерфейсу.

3.3.3 Resize system

Під час розробки було реалізовано систему зміни розміру overlay вікна оскільки користувачу необхідно швидко підлаштовувати область моніторингу під різні формати відео або кількість камер на екрані. Також була реалізована можливість розгортання області моніторингу практично на весь екран.

Для зміни розміру використовуються події руху миші та координати курсора.

def mouseMoveEvent(self, event):

Під час руху курсора система перевіряє чи знаходиться користувач у зоні ресайзу після чого змінює розмір overlay області.

Подібний підхід дозволяє зробити взаємодію із системою більш зручною та швидко змінювати область детекції без необхідності відкривати окремі налаштування.

3.3.4 Polygon editor

Однією з основних функцій інтерфейсу є система створення полігональних зон детекції. Дана функція використовується для обмеження області у межах якої система буде реагувати на рух та запускати аналіз через YOLOv8.

Під час створення полігону користувач вручну додає точки після чого система формує замкнену область.

self.current_polygon.append(event.pos())

Після завершення створення полігону область зберігається у список активних зон детекції.

self.track_zones.append(self.current_polygon)

Для відображення полігонів та інших елементів поверх overlay вікна використовується QPainter.

p.drawPolygon(QPolygon(self.current_polygon))

Подібний підхід дозволяє гнучко налаштовувати область моніторингу та виключати із перевірки частини кадру де постійно присутній непотрібний рух наприклад дерева дорога або інші зовнішні обєкти.

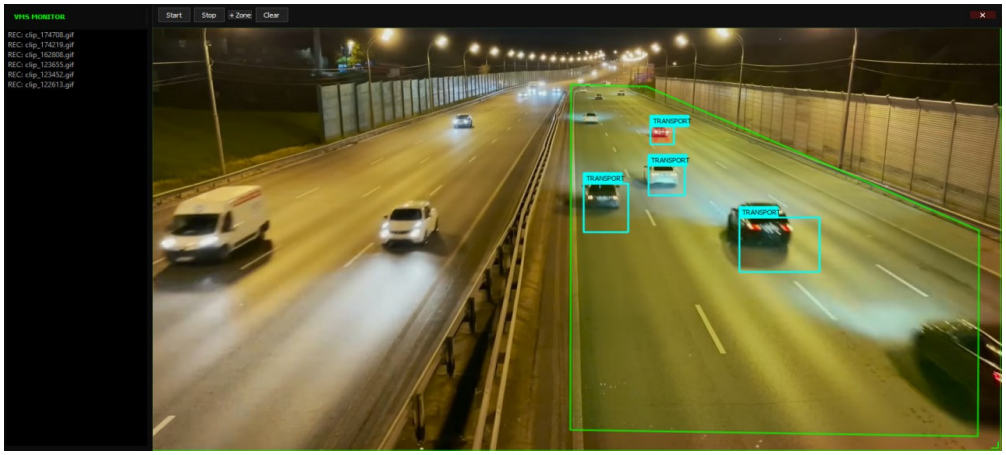

Рис. 4 Запущений тест детекції

Таким чином графічний інтерфейс застосунку забезпечує повноцінну взаємодію користувача із системою детекції руху та дозволяє швидко налаштовувати область моніторингу запускати аналіз відеопотоку змінювати параметри роботи системи та створювати полігональні області без необхідності використання додаткових програм або окремих модулів налаштування.

3.4 Реалізація алгоритму виявлення руху

Однією з основних частин застосунку є система визначення руху у вибраній області моніторингу оскільки саме вона відповідає за запуск подальшого аналізу через модель YOLOv8. Під час розробки було важливо реалізувати достатньо простий та швидкий алгоритм який міг би працювати у режимі реального часу без сильного навантаження на систему. Саме через це було вирішено використовувати метод порівняння кадрів який дозволяє визначати зміну зображення між поточним та попереднім кадром.

Після запуску системи застосунок починає постійно отримувати кадри із вибраної області екрана через бібліотеку MSS. Отримане зображення передається у функцію аналізу де виконується попередня обробка кадру та перевірка на наявність руху.

Отримання поточного кадру виконується наступним чином.

```
shot = self.sct.grab(bbox)

frame = cv2.cvtColor( np.array(shot),

   cv2.COLOR_BGRA2BGR … )
```

Після отримання кадру система переводить його у чорно білий формат. Подібний підхід використовується для того щоб зменшити кількість інформації яку потрібно аналізувати під час порівняння кадрів. Для визначення руху кольорова інформація фактично не потрібна тому використання grayscale підвищити продуктивність та прискорити сам процес аналізу.

Переведення кадру у grayscale виконується через OpenCV[4].

gray = cv2.cvtColor(

frame,

cv2.COLOR_BGR2GRAY …)

Після цього до кадру застосовується Gaussian Blur який використовується для зменшення шумів та дрібних змін зображення. Це потрібно для того щоб система менше реагувала на незначні зміни кадру які можуть виникати через цифрові шуми камери або дрібні коливання зображення.

gray = cv2.GaussianBlur(

gray, (15, 15), 0 …)

Після попередньої обробки система виконує порівняння поточного кадру із попереднім. Для цього використовується функція absdiff() яка визначає абсолютну різницю між двома кадрами.

delta = cv2.absdiff(

self.prev_frame, gray …)

Фактично алгоритм порівнює яскравість кожного пікселя у двох кадрах та визначає наскільки сильно змінилось зображення. Математично різницю кадрів можна представити у вигляді формули:

D(x,y)=|Ft(x,y)−Ft−1(x,y)| де :

Ft(x,y) – поточний кадр

Ft−1(x,y) – попередній кадр

D(x,y) – різниця між пікселями

Після цього система виконує threshold обробку яка переводить зображення у бінарний формат. Якщо різниця між пікселями перевищує встановлений поріг піксель стає білим в іншому випадку чорним.

thresh = cv2.threshold(

delta, 30, 255, cv2.THRESH_BINARY)[1]

Роботу threshold можна представити наступною формулою:

$$T(x,y)=\{255,\ D(x,y)>\theta\ \ 0,\ D(x,y)\leq\theta$$

Після threshold система отримує чорно білу маску на якій білі області відповідають зонам де було помічено зміну кадру. Далі виконується пошук контурів які використовуються для визначення областей руху.

Захоплення кадру	MSS
Grayscale	Зменшення інформації
Blur	Фільтрація шумів
absdiff	Пошук різниці
threshold	Бінаризація
contours	Пошук руху

contours, _ = cv2.findContours(

thresh,

cv2.RETR_EXTERNAL,

cv2.CHAIN_APPROX_SIMPLE …)

Після отримання контурів система виконує додаткову фільтрацію оскільки дрібні контури можуть виникати через шуми камери зміну освітлення або інші незначні зміни кадру. Для цього використовується перевірка площі контуру.

if cv2.contourArea(c) < 500 continue

Подібний підхід дозволяє ігнорувати дрібні зміни кадру та реагувати лише на достатньо великі області руху. Саме це допомагає зменшити кількість фальшивих спрацювань під час роботи системи.

Після визначення області руху система перевіряє чи знаходиться активність у межах полігональної зони яку створив користувач. Це потрібно для того щоб не реагувати на рух за межами основної області моніторингу.

cv2.pointPolygonTest(

polygon,

(cx, cy),

False …)

Якщо рух знаходиться всередині дозволеної області система запускає подальший аналіз через модель YOLOv8. Подібний підхід дозволяє не запускати нейронну мережу постійно на кожному кадрі а використовувати її лише після виявлення активності. Саме через це вдається значно зменшити навантаження на систему та покращити продуктивність застосунку під час тривалого моніторингу.

Однією з основних проблем подібного підходу є можливість помилкових спрацювань через погодні умови зміну освітлення або цифрові шуми камери. Особливо помітною дана проблема стає у нічний час або при використанні дешевих камер із низькою якістю зображення. Саме через це у системі використовуються blur та фільтрація контурів які дозволяють зменшити вплив подібних факторів на роботу алгоритму.

Таким чином реалізований алгоритм визначення руху дозволяє достатньо швидко аналізувати відеопотік у режимі реального часу визначати появу активності у зоні моніторингу та запускати подальший аналіз через модель YOLOv8 лише у потрібний момент що значно зменшує навантаження на систему під час роботи застосунку.

3.5 Інтеграція YOLOv8

Після реалізації системи визначення руху наступним етапом стала інтеграція моделі YOLOv8 для аналізу обєктів у зоні спостереження. Основною задачею моделі є не просто визначення факту руху а розуміння того

який саме обєкт знаходиться у кадрі. Подібний підхід дозволяє значно зменшити кількість фальшивих спрацювань оскільки система може реагувати лише на певні типи обєктів та ігнорувати зайві зміни кадру які не є важливими для моніторингу.

У даному застосунку було вирішено використовувати модель YOLOv8n оскільки дана версія є однією з найбільш легких серед моделей серії YOLOv8 та має хорошу швидкість роботи навіть на звичайному пк без потужної відеокарти. Під час тестування було помітно що більші моделі хоча і можуть давати трохи кращу точність але створюють значно більше навантаження на систему через що починають виникати просадки fps та затримки під час аналізу відеопотоку. Саме через це для даного застосунку було вибрано баланс між швидкістю роботи та точністю визначення обєктів.

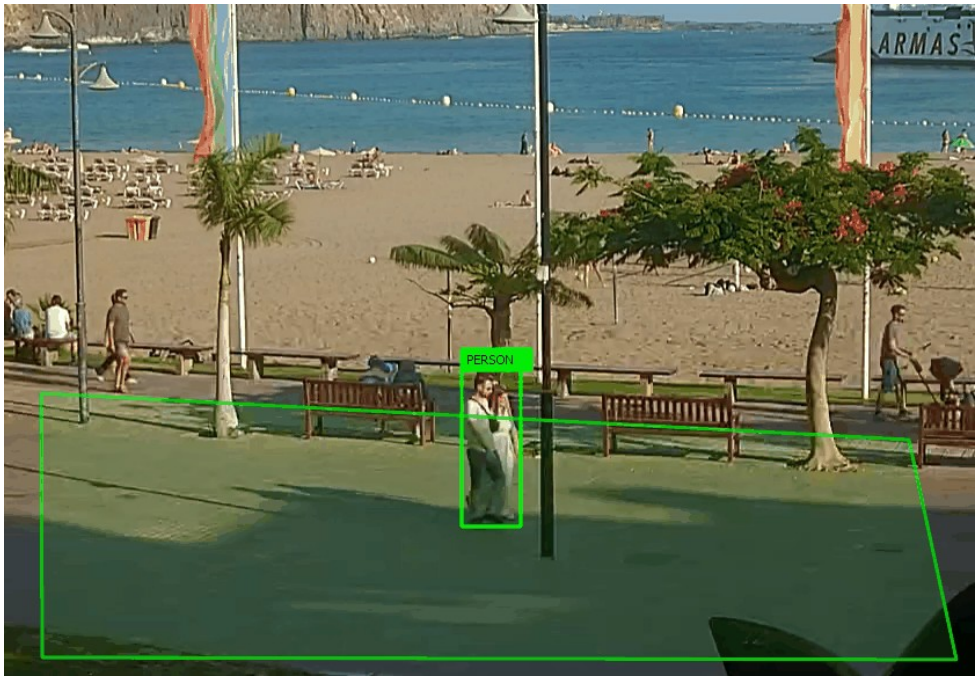

Рис 5. приклад детекції боксу людини

Файл моделі знаходиться безпосередньо у папці проєкту та завантажується під час запуску застосунку.

self.model = YOLO("yolov8n.pt")

Після завантаження моделі система може використовувати її для аналізу кадрів та визначення обєктів у зоні моніторингу. Важливою особливістю реалізації є те що модель не запускається постійно на кожному кадрі а

використовується лише після того як система вже помітила рух. Подібний підхід дозволяє значно зменшити навантаження на систему та зробити роботу застосунку більш стабільною при тривалому моніторингу.

Після виявлення руху поточний кадр передається у модель YOLOv8 для подальшого аналізу.

res = self.model.predict(

frame conf=0.45, verbose=False, stream=True …)

Під час виконання predict() модель аналізує кадр та повертає список знайдених обєктів разом із координатами боксів та рівнем впевненості. Параметр conf використовується для визначення мінімального рівня впевненості моделі при якому результат буде вважатись валідним. У даному випадку використовується значення 0.45 що дозволяє відфільтрувати частину помилкових визначень.

Після завершення аналізу система починає обробляти знайдені обєкти та отримує координати кожного бокса.

for r in res:

for b in r.boxes:

Для кожного знайденого обєкта система отримує координати області детекції.

x1, y1, x2, y2 = map(int, b.xyxy[0] …)

Також окремо визначається клас обєкта який був знайдений моделлю.

cls = int(b.cls[0])

У застосунку використовується власна система групування класів оскільки для моніторингу не потрібні всі типи обєктів які підтримує YOLOv8. Через це частина класів обєднується у окремі категорії.

CLASS_MAPPING = {

0: "person",

5: "transport",

7: "transport",

15: "animal",

16: "animal" … }

Подібний підхід дозволяє спростити подальшу обробку результатів та не працювати окремо із великою кількістю класів які фактично не потрібні для системи моніторингу. Наприклад різні типи транспорту можуть оброблятись як одна категорія без додаткового розділення.

Після визначення класу система формує бокс навколо знайденого обєкта та відображає його поверх overlay вікна.

p.drawRect(

x1 + ox,

y1 + oy,

x2 - x1,

y2 - y1 …)

Також окремо виводиться назва знайденого класу для зручності моніторингу.

p.drawText(

x1 + ox,

y1 + oy - 5,

label …)

Однією з важливих частин реалізації є перевірка того чи знаходиться обєкт у межах полігональної області яку створив користувач. Для цього система визначає центр бокса та перевіряє його положення відносно полігону.

Подібний підхід дозволяє реагувати лише на обєкти які знаходяться у межах дозволеної області моніторингу та ігнорувати рух за межами полігону.

Під час тестування було помітно що продуктивність системи сильно залежить від розміру області моніторингу та кількості обєктів у кадрі. Якщо одночасно присутня велика кількість руху навантаження на систему починає збільшуватись через постійний запуск моделі. Також швидкість роботи залежить від самого обладнання та потужності відеокарти. На слабких системах можуть виникати просадки fps особливо при роботі із високою якістю відео або декількома камерами одночасно.

Попри це використання YOLOv8 дозволило значно покращити якість визначення обєктів у порівнянні зі звичайною детекцією руху. Система почала краще відфільтровувати непотрібні спрацювання та реагувати саме на ті обєкти які є важливими для моніторингу. Таким чином інтеграція YOLOv8 дозволила поєднати просту систему визначення руху із більш точним аналізом обєктів та зробити роботу застосунку значно ефективнішою у реальних умовах використання.

3.6 Реалізація системи запису GIF

Однією з додаткових функцій застосунку стала система автоматичного запису гіфок після виявлення руху у зоні моніторингу. Основною задачею даної функції є збереження коротких фрагментів активності без необхідності постійного запису великого обєму відео. Подібний підхід дозволяє значно економити місце на диску та спрощує подальший перегляд моментів у яких система зафіксувала рух або появу обєктів у кадрі.

Під час роботи застосунок постійно отримує кадри із області моніторингу та після виявлення руху починає тимчасово зберігати їх у буфер. Для цього використовується окремий список у який додаються копії кадрів.

```
self.gif_frames.append(
    frame.copy() … )
```

Використання буфера дозволяє накопичувати невелику кількість кадрів перед створенням гіфки. Подібний підхід працює значно швидше ніж

постійний запис повноцінного відео файлу та створює менше навантаження на систему під час роботи застосунку.

Для того щоб буфер не займав занадто багато памяті використовується обмеження максимальної кількості кадрів.

if len(self.gif_frames) < 60:

У даному випадку система зберігає приблизно 60 кадрів після чого формується готова гіфка. Тривалість анімації залежить від fps який використовується під час створення файлу. Для запису гіфок у застосунку використовується бібліотека imageio яка дозволяє формувати анімовані gif файли без необхідності використання додаткових відеокодеків.

Створення гіфки виконується наступним чином[10].

```
imageio.mimsave(
   path,
   [ cv2.cvtColor(  f, cv2.COLOR_BGR2RGB  )
      for f in self.gif_frames  ], fps=12 )
```

Перед записом кадри переводяться із формату BGR у RGB оскільки OpenCV використовує інший формат кольору ніж imageio. Після цього всі кадри обєднуються у одну анімацію та зберігаються у папку gifs.

Для збереження гіфок використовується окрема директорія яка створюється автоматично під час запуску застосунку.

os.makedirs("gifs", exist_ok=True …)

Після створення нової гіфки система автоматично додає її у список записів який відображається у sidebar інтерфейсу. Це дозволяє швидко переглядати всі збережені фрагменти активності без необхідності відкривати папку вручну.

Під час тестування було помітно що gif формат добре підходить для коротких записів моментів активності оскільки файли займають значно менше місця ніж звичайне відео. Особливо це помітно при великій кількості камер де

постійний запис може дуже швидко заповнювати накопичувач. Також gif формат дозволяє швидко переглядати події без використання окремого відеоплеєра.

Однією з проблем такого підходу є те що gif не дуже добре підходить для довгих записів або великої кількості кадрів оскільки розмір файлу починає швидко збільшуватись. Через це у застосунку використовується коротка тривалість запису та обмеження кількості кадрів у буфері.

Також важливою особливістю є те що запис гіфки запускається лише після виявлення руху та проходження перевірки через систему детекції. Завдяки цьому у папку не записуються порожні або непотрібні фрагменти де фактично не було активності.

Таким чином реалізована система запису гіфок дозволяє автоматично зберігати короткі фрагменти руху у зоні моніторингу забезпечує швидкий перегляд зафіксованих подій та зменшує використання дискового простору у порівнянні із постійним записом повного відеопотоку.

ВИСНОВКИ

У результаті виконання даної роботи було розроблено локальний застосунок для моніторингу відеопотоку із системою визначення руху та інтеграцією моделі YOLOv8 для аналізу обєктів у кадрі. Основною задачею під час розробки було створення системи яка могла б працювати у режимі реального часу поверх інших програм для перегляду камер без необхідності прямого підключення до самих відеопотоків або використання зовнішніх серверів для обробки зображення. Під час виконання роботи було проведено аналіз існуючих систем відеоспостереження та методів детекції руху. У ході аналізу було визначено що більшість сучасних систем або потребують складної інтеграції із камерами та VMS платформами або створюють значне навантаження на систему через постійний аналіз відеопотоку. Також було виявлено що звичайні алгоритми визначення руху часто створюють велику кількість фальшивих спрацювань через зміну освітлення погодні умови або

цифрові шуми камери. Саме через це було прийнято рішення поєднати класичну систему визначення руху із використанням нейронної мережі YOLOv8 для більш точного аналізу обєктів у кадрі. У роботі було виконано вибір основних засобів реалізації застосунку. Для розробки системи була використана мова програмування Python а також бібліотеки PyQt5 OpenCV MSS imageio та Ultralytics YOLOv8. Використання даних технологій дозволило реалізувати повноцінний desktop застосунок із підтримкою прозорого overlay інтерфейсу захоплення кадрів із екрана аналізу руху та роботи нейронної мережі у режимі реального часу. У процесі реалізації було створено графічний інтерфейс який дозволяє користувачу швидко взаємодіяти із системою моніторингу змінювати область детекції запускати або зупиняти аналіз кадрів а також створювати полігональні області для обмеження зони спостереження. Використання overlay підходу дозволило зробити систему універсальною оскільки вона може працювати практично із будь якими програмами для перегляду камер незалежно від способу передачі відео. Також у роботі було реалізовано алгоритм визначення руху на основі порівняння поточного та попереднього кадру. Для зменшення кількості фальшивих спрацювань використовувались попередня обробка кадрів Gaussian Blur threshold обробка та фільтрація контурів за площею. Подібний підхід дозволив достатньо швидко визначати появу активності у зоні моніторингу та запускати подальший аналіз через YOLOv8 лише у потрібний момент що значно зменшило навантаження на систему. Окремою частиною роботи стала інтеграція моделі YOLOv8n для визначення обєктів у кадрі. Під час реалізації було створено систему групування класів яка дозволяє реагувати лише на важливі обєкти такі як люди транспорт або тварини. Це дозволило значно зменшити кількість помилкових спрацювань у порівнянні зі звичайною детекцією руху та підвищити ефективність роботи системи у реальних умовах використання. Також у застосунку була реалізована система автоматичного створення GIF файлів після виявлення активності у зоні спостереження. Дана функція дозволяє зберігати короткі фрагменти руху без необхідності

постійного запису великого обєму відео. Використання GIF формату дозволило зменшити використання дискового простору та спростити подальший перегляд зафіксованих подій. Під час тестування було виявлено що продуктивність системи напряму залежить від розміру області моніторингу кількості руху у кадрі та потужності самого обладнання. На слабких системах при великій кількості обєктів можуть виникати просадки fps або затримки під час запуску моделі YOLOv8. Попри це застосунок показав достатньо стабільну роботу у режимі реального часу та продемонстрував можливість використання подібного підходу для локальних систем моніторингу. Таким чином поставлена мета роботи була досягнута. У результаті було створено локальний застосунок для моніторингу відеопотоку який поєднує класичне визначення руху із сучасними методами аналізу обєктів на основі нейронних мереж та може використовуватись для спостереження за активністю у вибраній області екрана у режимі реального часу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Python Official Documentation : офіційна документація мови програмування Python. URL: https://docs.python.org/3/ (дата звернення: 01.05.2026).

2. NumPy Official Documentation : офіційна документація бібліотеки NumPy. URL: https://numpy.org/doc/ (дата звернення: 01.05.2026).

3. OpenCV Official Documentation : офіційна документація бібліотеки OpenCV для компютерного зору. URL: https://docs.opencv.org/4.x/ (дата звернення: 03.05.2026).

4. OpenCV GitHub Repository : офіційний GitHub репозиторій OpenCV. URL: https://github.com/opencv/opencv (дата звернення: 03.05.2026).

5. PyQt5 Documentation : офіційна документація бібліотеки PyQt5 для створення графічного інтерфейсу. URL: https://www.riverbankcomputing.com/static/Docs/PyQt5/ (дата звернення: 05.05.2026).

6. Qt Documentation : документація фреймворку Qt. URL: https://doc.qt.io/ (дата звернення: 05.05.2026).

7. MSS Python Library : документація бібліотеки MSS для захоплення екрана. URL: https://python-mss.readthedocs.io/ (дата звернення: 07.05.2026).

8. MSS GitHub Repository : GitHub репозиторій бібліотеки MSS. URL: https://github.com/BoboTiG/python-mss (дата звернення: 08.05.2026).

9. ImageIO Documentation : офіційна документація бібліотеки imageio. URL: https://imageio.readthedocs.io/en/stable/ (дата звернення: 09.05.2026).

10. ImageIO GitHub Repository : GitHub репозиторій бібліотеки imageio. URL: https://github.com/imageio/imageio (дата звернення: 09.05.2026).

11. PyTorch Official Documentation : офіційна документація бібліотеки PyTorch. URL: https://pytorch.org/docs/stable/index.html (дата звернення: 11.05.2026).

12. PyTorch GitHub Repository : офіційний GitHub репозиторій PyTorch. URL: https://github.com/pytorch/pytorch (дата звернення: 11.05.2026).

13. Ultralytics YOLOv8 Documentation : офіційна документація моделі YOLOv8. URL: https://docs.ultralytics.com/models/yolov8/ (дата звернення: 13.05.2026).

14. Ultralytics GitHub Repository : GitHub репозиторій бібліотеки Ultralytics YOLO. URL: https://github.com/ultralytics/ultralytics (дата звернення: 14.05.2026).

15. Introducing Ultralytics YOLOv8 : офіційна стаття про YOLOv8 від Ultralytics. URL: https://www.ultralytics.com/blog/introducing-ultralytics-yolov8 (дата звернення: 15.05.2026).

16. Qt for Python Documentation : документація Qt for Python для роботи із PyQt/PySide. URL: https://doc.qt.io/qtforpython/ (дата звернення: 16.05.2026).

Додатки

Додаток А

```
import sys
import os
import time
try:
    import numpy as np
except ImportError:
    np = None
import cv2
import mss
from datetime import datetime
from PyQt5.QtWidgets import (QApplication, QWidget, QPushButton, QHBoxLayout,QVBoxLayout, QListWidget, QListWidgetItem, QLabel, QMessageBox)
from PyQt5.QtCore import Qt, QTimer, QPoint, QRect
from PyQt5.QtGui import QPainter, QPen, QColor, QPolygonF
import imageio
from ultralytics import YOLO[11,12,14]

COLORS = {
    "transport": QColor(0, 255, 255),
    "animal": QColor(255, 255, 0),
    "person": QColor(0, 255, 0),
    "zone_fill": QColor(0, 255, 0, 20),
    "zone_line": QColor(0, 255, 0, 180),
    "monitor_border": QColor(0, 255, 0, 60),
    "window_outer": QColor(0, 255, 0, 100)
}
```

```
CLASS_MAPPING = {0: "person", 1: "transport", 2: "transport", 3: "transport", 5: "transport", 7: "transport", 15: "animal", 16: "animal"}

class SmartVMS(QWidget):
    def __init__(self):
        super().__init__()
        os.makedirs("gifs", exist_ok=True)
                        self.setWindowFlags(Qt.FramelessWindowHint | Qt.WindowStaysOnTopHint)
        self.setAttribute(Qt.WA_TranslucentBackground)
        self.setMinimumSize(800, 500)
        self.setGeometry(100, 100, 1100, 650)

        self._resizing = False
        self._drag_pos = None
        self.margin = 20

        self.is_drawing_mode = False
        self.current_polygon_points = []
        self.track_zones = []
        self.tracked_objects = []
        self.prev_frame = None
        self.recording_gif = False
        self.gif_frames = []
        self.last_motion_time = 0

        self.init_ui()
        self.sct = mss.mss()
        self.timer = QTimer()
        self.timer.timeout.connect(self.process_frame)
```

```
        self.model = YOLO("yolov8n.pt")

    def init_ui(self):
        layout = QHBoxLayout(self)
        layout.setContentsMargins(0, 0, 0, 0)
        layout.setSpacing(0)

        self.sidebar = QWidget()
        self.sidebar.setFixedWidth(240)
        self.sidebar.setStyleSheet("background-color: #0c0c0c; border-right: 1px solid #222;")
        side_layout = QVBoxLayout(self.sidebar)
        side_layout.addWidget(QLabel("VMS MONITOR", styleSheet="color: #0f0; font-weight: bold; padding: 10px;"))
        self.alert_list = QListWidget(styleSheet="background: #000; border: none; color: #aaa;")
        self.alert_list.itemDoubleClicked.connect(self.open_gif)
        side_layout.addWidget(self.alert_list)

        self.right_container = QWidget()
        right_vbox = QVBoxLayout(self.right_container)
        right_vbox.setContentsMargins(0, 0, 0, 0)
        right_vbox.setSpacing(0)

        self.toolbar = QWidget()
        self.toolbar.setFixedHeight(45)
        self.toolbar.setStyleSheet("background-color: #111; border-bottom: 1px solid #222;")
        t_layout = QHBoxLayout(self.toolbar)
```

```
            self.btn_start = QPushButton("Start")
            self.btn_stop = QPushButton("Stop")
            self.btn_zone = QPushButton("+ Zone")
            self.btn_clear = QPushButton("Clear")

            for btn in [self.btn_start, self.btn_stop, self.btn_zone, self.btn_clear]:
                btn.setStyleSheet("background: #222; color: #fff; border: 1px solid #444; padding: 5px 12px;")
                t_layout.addWidget(btn)

            self.btn_start.clicked.connect(self.start_detection)
            self.btn_stop.clicked.connect(self.stop_detection)
            self.btn_zone.clicked.connect(self.toggle_drawing_mode)
            self.btn_clear.clicked.connect(self.clear_polygons)

            t_layout.addStretch()
            close = QPushButton("✕", styleSheet="background: #411; color: white; border: none; width: 40px;")
            close.clicked.connect(self.close)
            t_layout.addWidget(close)

            right_vbox.addWidget(self.toolbar)

            self.interaction_area = QWidget()
            self.interaction_area.setStyleSheet("background-color: rgba(0, 255, 0, 5);")
            right_vbox.addWidget(self.interaction_area, 1)

            layout.addWidget(self.sidebar)
            layout.addWidget(self.right_container)
```

```
            self.refresh_alert_list()

        def mousePressEvent(self, event):
            pos = event.pos()
            ox, oy = self.sidebar.width(), self.toolbar.height()

             if pos.x() >= self.width() - self.margin or pos.y() >= self.height() -
self.margin:
                self._resizing = True

            elif self.toolbar.geometry().translated(ox, 0).contains(pos):
                self._drag_pos = event.globalPos() - self.pos()№
            elif self.is_drawing_mode and pos.x() > ox and pos.y() > oy:
              self.current_polygon_points.append(QPoint(pos.x() - ox, pos.y() - oy))
                self.update()

        def mouseMoveEvent(self, event):
            if self._resizing:
                self.resize(max(800, event.pos().x()), max(500, event.pos().y()))
            elif self._drag_pos:
                self.move(event.globalPos() - self._drag_pos)

             if event.pos().x() >= self.width() - self.margin and event.pos().y() >=
self.height() - self.margin:
                self.setCursor(Qt.SizeFDiagCursor)
            else:
                self.setCursor(Qt.ArrowCursor)

        def mouseReleaseEvent(self, event):
            self._resizing = False
```

```
            self._drag_pos = None

        def process_frame(self):
            try:
                geom = self.interaction_area.geometry()
                gp = self.interaction_area.mapToGlobal(QPoint(0,0))
                bbox = {"left": gp.x(), "top": gp.y(), "width": geom.width(), "height":
geom.height()}

                shot = self.sct.grab(bbox)
                frame = cv2.cvtColor(np.array(shot), cv2.COLOR_BGRA2BGR)
                                gray = cv2.GaussianBlur(cv2.cvtColor(frame,
cv2.COLOR_BGR2GRAY), (15, 15), 0)

                if self.prev_frame is None:
                    self.prev_frame = gray
                    return

                delta = cv2.absdiff(self.prev_frame, gray)
                thresh = cv2.threshold(delta, 30, 255, cv2.THRESH_BINARY)[1]
                self.prev_frame = gray

                if np.sum(thresh) / 255 > 1000:
                    self.last_motion_time = time.time()
                        res = self.model.predict(frame, conf=0.45, verbose=False,
stream=True)
                    new_objs = []
                    for r in res:
                        for d in r.boxes:
                            cat = CLASS_MAPPING.get(int(d.cls[0]))
```

```
                    if cat:
                        x1, y1, x2, y2 = map(int, d.xyxy[0])
                        if np.sum(thresh[y1:y2, x1:x2]) / 255 > 25:
                            cx, cy = (x1+x2)//2, (y1+y2)//2
                                                    if not self.track_zones or any(cv2.pointPolygonTest(np.array([[p.x(), p.y()] for p in z]), (cx, cy), False) >= 0 for z in self.track_zones):
                                        new_objs.append({"box": (x1, y1, x2, y2), "cat": cat})

                self.tracked_objects = new_objs
                if not self.recording_gif:
                    self.recording_gif = True
                    self.gif_frames = []
                if len(self.gif_frames) < 60: self.gif_frames.append(frame.copy())
            else:
                if time.time() - self.last_motion_time > 2.0:
                    if self.recording_gif: self.save_gif()
                    self.tracked_objects = []
            self.update()
        except: pass

    def save_gif(self):
        if len(self.gif_frames) > 15:
            path = f"gifs/clip_{datetime.now().strftime('%H%M%S')}.gif"
            imageio.mimsave(path, [cv2.cvtColor(f, cv2.COLOR_BGR2RGB) for f in self.gif_frames], fps=12)
            self.refresh_alert_list()
        self.recording_gif = False
        self.gif_frames = []
```

```
        def refresh_alert_list(self):
            self.alert_list.clear()
            if os.path.exists("gifs"):
                                for f in sorted(os.listdir("gifs"), reverse=True):
self.alert_list.addItem(f"REC: {f}")

        def open_gif(self, item):
            os.startfile(os.path.abspath(f"gifs/{item.text().replace('REC: ', '')}"))

        def paintEvent(self, event):
            p = QPainter(self)
            p.setRenderHint(QPainter.Antialiasing)
            ox, oy = self.sidebar.width(), self.toolbar.height()
            ww, wh = self.width(), self.height()

            p.setPen(QPen(COLORS["window_outer"], 2))
            p.drawRect(1, 1, ww-2, wh-2)

            p.setPen(QPen(COLORS["monitor_border"], 1))
            p.drawRect(ox, oy, ww - ox - 1, wh - oy - 1)

            p.setPen(QPen(COLORS["person"], 2))
            p.drawLine(ww-15, wh-5, ww-5, wh-5)
            p.drawLine(ww-5, wh-15, ww-5, wh-5)

            p.setPen(QPen(COLORS["zone_line"], 3))
            for z in self.track_zones:
                p.setBrush(COLORS["zone_fill"])
               p.drawPolygon(QPolygonF([QPoint(pt.x()+ox, pt.y()+oy) for pt in z]))
```

```
            if self.is_drawing_mode and self.current_polygon_points:
                p.setPen(QPen(Qt.white, 2, Qt.DashLine))
                pts = [QPoint(pt.x()+ox, pt.y()+oy) for pt in self.current_polygon_points]
                if len(pts) > 1: p.drawPolyline(QPolygonF(pts))

            for o in self.tracked_objects:
                col = COLORS.get(o["cat"], Qt.white)
                x1, y1, x2, y2 = o["box"]
                p.setPen(QPen(col, 3))
                p.setBrush(Qt.NoBrush)
                p.drawRect(x1 + ox, y1 + oy, x2 - x1, y2 - y1)
                p.setBrush(col)
                p.drawRect(x1 + ox, y1 + oy - 18, 60, 18)
                p.setPen(Qt.black)
                p.drawText(x1 + ox + 5, y1 + oy - 5, o["cat"].upper())

        def start_detection(self): self.timer.start(50)
        def stop_detection(self): self.timer.stop(); self.save_gif(); self.tracked_objects = []; self.update()

        def toggle_drawing_mode(self):
            if not self.is_drawing_mode:
                self.is_drawing_mode = True
                self.btn_zone.setStyleSheet("background: #060; color: #0f0; border: 1px solid #0f0;")
            else:
                if len(self.current_polygon_points) > 2:
                    self.track_zones.append(self.current_polygon_points.copy())
                self.current_polygon_points.clear()
```

```
                self.is_drawing_mode = False
                 self.btn_zone.setStyleSheet("background: #222; color: #fff; border:
1px solid #444;")
            self.update()

        def clear_polygons(self):
            self.track_zones.clear()
            self.update()

    if __name__ == "__main__":
        app = QApplication(sys.argv)
        ex = SmartVMS()
        ex.show()
        sys.exit(app.exec_())
```