Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

До захисту допущено
Кафедрою комп'ютерних систем та робототехніки
протокол № __ від __ червня 2026 р.

завідувач кафедри _____ Максим ХРУСЛОВ
(підпис)
«___»_____ 2026 р.

Кваліфікаційна робота

здобувача першого (бакалаврського) рівня вищої освіти

КОМП'ЮТЕРНА СИСТЕМА ВІДЕОСПОСТЕРЕЖЕННЯ ІЗ ЗАСТОСУВАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

Спеціальність ***151 – Автоматизація та комп'ютерно-інтегровані технології***

Освітня програма ***Автоматизація та комп'ютерно інтегровані технології***

Виконавець _____ **Дмитро МАКОГОН**
(підпис)

Науковий керівник _____ **Тетяна БИКОВА**
(підпис)

Харків – 2026

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел та п’яти додатків. Загальний обсяг роботи становить 62 сторінки, з яких 33 сторінки основної частини, що містить 9 рисунків, 3 таблиці, 23 найменування у списку використаних джерел.

Метою кваліфікаційної роботи є підвищення ефективності та автоматизації процесів моніторингу шляхом розробки багатоканальної системи інтелектуального відеоспостереження у реальному часі з автоматичним розпізнаванням людей на основі архітектури глибокого навчання YOLOv8 та інтеграцією засобів миттєвого хмарного сповіщення.

Об’єкт дослідження – процеси автоматизованого моніторингу простору, обробки багатоканальних відеопотоків та розпізнавання образів у реальному часі.

Предмет дослідження – архітектурні рішення програмного забезпечення, математичні моделі та нейромережеві алгоритми детекції об'єктів для виявлення цільових класів у відеопотоці з використанням апаратного прискорення обчислень.

Проблема, яка вирішується в кваліфікаційній роботі, полягає в тому, щоб за допомогою сучасних програмних інструментів та оптимізації алгоритмів обробки кадрів створити відмовостійку систему відеонагляду, яка мінімізує навантаження на обчислювальні ресурси при паралельній роботі кількох камер, автоматичне збереження доказів і хмарної дистрибуції тривожних сповіщень.

Область застосування – системи «розумний дім», інтегровані комплекси безпеки приватних та комерційних об'єктів, а також освітній процес за спеціальністю 151 «Автоматизація та комп’ютерно-інтегровані технології».

Ключові слова: Автоматизація та комп’ютерно – інтегровані технології, відеоспостереження, комп'ютерний зір, YOLOv8, python, openCV, customtkinter, багатопотоковість, апаратне прискорення, CUDA, telegram API.

ABSTRACT

The explanatory note to the qualification work consists of an introduction, three sections, conclusions, a list of sources used and five appendices. The total volume of the work is 62 pages, of which 33 pages are the main part containing 9 figures, 3 tables, and 23 items in the list of sources used.

The purpose of the qualification work is to increase the efficiency and automation of monitoring processes by developing a multi-channel real-time intelligent video surveillance system with automatic human recognition based on the YOLOv8 deep learning architecture and integrating instant cloud notification tools.

The object of research is the processes of automated space monitoring, multi-channel video stream processing, and real-time pattern recognition.

The subject of resea*rch* is software architectural solutions, mathematical models, and neural network object detection algorithms for identifying target classes in a video stream using hardware-accelerated computations.

The problem solved in the qualification work is to create a fault-tolerant video surveillance system using modern software tools and frame processing optimization that minimizes the load on computational resources (CPU/GPU) during parallel operation of multiple cameras, prevents UI freezing, and eliminates the omission of recorded security breaches through automated evidence retention and cloud alert distribution.

Area of application – "Smart Home" systems, integrated security complexes for private and commercial properties, access control automation systems at enterprises, as well as the educational process in specialty 151 "Automation and Computer-Integrated Technologies".

Keywords: Automation and Computer-Integrated Technologies, machine learning, video surveillance, computer vision, YOLOv8, python, openCV, customtkinter, multithreading, hardware acceleration, CUDA, telegram API.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ 5

ВСТУП 6

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ТА ПІДХОДІВ ДО ПОБУДОВИ СИСТЕМ ВІДЕОСПОСТЕРЕЖЕННЯ 9

1.1 Огляд класичних методів комп'ютерного зору та детекції руху ... 9

1.2 Аналіз сучасних нейромережевих архітектур для розпізнавання образів 12

1.3 Постановка задачі дослідження та обґрунтування обраного стеку технологій 14

Висновки за розділом 1 16

РОЗДІЛ 2. ПРОЕКТУВАННЯ, ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ІНТЕЛЕКТУАЛЬНОГО ВІДЕОСПОСТЕРЕЖЕННЯ 17

2.1 Вибір та обґрунтування програмних засобів та бібліотек 17

2.2 Модульна архітектура програмного комплексу 18

2.3 Інтеграція алгоритмів YOLOv8 та налаштування апаратного прискорення CUDA 20

2.4 Розробка підсистеми локального збереження доказів та хмарного сповіщення 22

Висновки за розділом 2 25

РОЗДІЛ 3. ЕКСПЛУАТАЦІЙНЕ ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ РОЗРОБЛЕНОГО ПРОГРАМНОГО КОМПЛЕКСУ ... 27

3.1 Методика проведення випробувань та конфігурація тестового стенду 27

3.2 Аналіз продуктивності, навантаження на обчислювальні ресурси системи 28

3.3 Верифікація роботи підсистем реагування за критеріями ТЗ 30

Висновки за розділом 3 32

ВИСНОВКИ 34

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 36

ДОДАТКИ 38

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ

ШІ	–	штучний інтелект;
ЕОМ	–	електронно-обчислювальна машина;
ПК	–	персональний комп'ютер;
ТЗ	–	технічне завдання;
ОС	–	операційна система;
API	–	(*Application Programming Interface*) – інтерфейс прикладного програмування;
CUDA	–	(*Compute Unified Device Architecture*) – архітектура паралельних обчислень на графічних процесорах від компанії NVIDIA;
YOLO	–	архітектура згорткових нейронних мереж для детекції об'єктів у режимі реального часу;
FPS	–	частота кадрування (кількість кадрів, що обробляються або відображаються системою за одну секунду);
GPU	–	(*Graphics Processing Unit*) – графічний процесор (відеокарта);
CPU	–	(*Central Processing Unit*) – центральний процесор;
GUI	–	(*Graphical User Interface*) – графічний інтерфейс користувача;
HTTP	–	(*HyperText Transfer Protocol*) – протокол передачі гіпертексту;
JPEG	–	(*Joint Photographic Experts Group*) – метод та формат стиснення цифрових зображень;

ВСТУП

Актуальність роботи: На сучасному етапі розвитку інформаційних технологій та комплексів автоматизації завдання забезпечення безпеки об'єктів різного призначення — від приватного житла до великих промислових підприємств — виходить на якісно новий інженерний рівень. Традиційні архітектури систем охоронного телебачення (CCTV) базуються на пасивному зборі відеоінформації та вимагають постійного візуального моніторингу со сторони людини-оператора. Проте практичний досвід експлуатації таких комплексів свідчить про низьку ефективність подібного підходу через дію так званого «людського фактора»: монотонність спостереження за статичними сценами призводить до швидкої втоми, критичного зниження концентрації уваги оператора вже після 20–30 хвилин безперервної роботи, що тягне за собою високу ймовірність пропуску правопорушень або аварійних ситуацій.

У зв'язку з цим виникає гостра потреба у повній або частковій автоматизації процесів аналізу відеоінформації безпосередньо в момент її отримання. Інтеграція методів комп'ютерного зору (Computer Vision) та алгоритмів глибокого машинного навчання (Deep Learning) в інтегровані системи автоматизації дозволяє створювати інтелектуальні комплекси відеоспостереження. Сучасні згорткові нейронні мережі здатні з високою точністю детектувати та класифікувати об'єкти в кадрі (зокрема, ідентифікувати людей) у режимі реального часу, що зводить до мінімуму кількість хибних спрацьовувань від природних завад (зміна освітлення, рух тварин, коливання рослинності) та забезпечує надійне автоматичне виявлення несанкціонованого проникнення в охоронювані зони.

Особливої актуальності набуває розробка багатоканальних локальних систем штучного інтелекту, які здатні працювати з декількома джерелами відеосигналу одночасно на доступному обчислювальному обладнанні загального призначення (Edge Computing), забезпечуючи при цьому автономність, гнучкість налаштувань, формування юридично достовірної локальної бази доказів із

точним часовим маркуванням та миттєву хмарну дистрибуцію сповіщень на мобільні пристрої відповідальних осіб.

Метою дослідження: є підвищення ефективності, надійності та автономності систем технічної безпеки шляхом розробки, проектування та програмної реалізації багатоканального інтелектуального комплексу відеоспостереження на базі мови програмування Python з використанням високооптимізованих нейромережевих моделей сімейства YOLOv8, що функціонують у реальному часі з залученням апаратного прискорення обчислень та інтегрованими адаптивними інструментами хмарного і локального реагування.

Об'єкт дослідження: процеси автоматизованого моніторингу простору, високошвидкісної обробки багатоканальних цифрових відеопотоків та інтелектуального розпізнавання образів у реальному часі.

Методи дослідження: Для розв'язання поставлених у роботі інженерних та наукових завдань застосовано: методи цифрової обробки зображень та просторової фільтрації (OpenCV), математичні апарати штучних нейронних мереж для детекції об'єктів (архітектура YOLOv8), принципи об'єктно-орієнтованого проектування програмних систем (парадигма ООП та шаблони проектування), методи багатопотокового програмування (модуль threading) для паралельного розподілу задач, а також методи експериментального аналізу складних систем для оцінки часових та обчислювальних характеристик.

Предмет дослідження: модульні архітектурні рішення програмного забезпечення, математичні моделі згорткових нейронних мереж, методи паралельних обчислень та протоколи хмарної взаємодії для виявлення цільових класів об'єктів у реальному часі.

Завдання дослідження:

1. Виконати детальний аналіз існуючих науково-методичних підходів, класичних алгоритмів комп'ютерного зору та сучасних нейромережевих архітектур, визначивши межі їх придатності для багатоканальних систем реального часу.

2. Спроєктувати гнучку модульну архітектуру програмного комплексу, що реалізує багатопотоковий розподіл задач (захоплення відеосигналу, інференс нейромережі, оновлення GUI, мережева взаємодія) для усунення затримок інтерфейсу.

3. Реалізувати інтеграцію нейромережевої моделі YOLOv8 та розробити алгоритми оптимізації обчислювального навантаження за рахунок адаптивного проріджування кадрів та перенесення тензорних операцій на графічний процесор за допомогою технології NVIDIA CUDA.

4. Розробити сучасний адаптивний графічний інтерфейс користувача з підтримкою динамічного масштабування відеопотоків без порушення пропорцій кадру та інструментами прямого керування параметрами системи.

5. Створити підсистему реагування, що забезпечує автоматичне формування доказової бази шляхом накладання постійного часового штампу на кадри та реалізувати асинхронний модуль хмарного сповіщення через Telegram Bot API.

6. Провести комплексні експлуатаційні випробування створеної системи, оцінити показники швидкодії (FPS, час інференсу) та навантаження на компоненти ЕОМ за різної кількості каналів спостереження.

РОЗДІЛ 1
АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ТА ПІДХОДІВ ДО ПОБУДОВИ СИСТЕМ ВІДЕОСПОСТЕРЕЖЕННЯ

1.1 Огляд класичних методів комп'ютерного зору та детекції руху

Розвиток систем технічного зору пройшов тривалий еволюційний шлях від найпростіших аналогових комплексів фіксації сигналу до складних комп'ютеризованих систем автоматичного аналізу зображень. На початкових етапах інтеграції обчислювальної техніки в інженерні системи безпеки та автоматизації домінували класичні (детерміновані) методи обробки зображень [4]. Математичний апарат цих методів базується на аналізі просторово-часових змін яскравості окремих пікселів або локальних груп пікселів цифрового кадру без глибокого розуміння семантичного змісту сцени. Оскільки цифровий відеопотік представляє собою послідовність двовимірних матриць інтенсивностей світла, перші алгоритми автоматизації охоронних систем базувалися на виявленні локальних аномалій у цих матрицях.

Серед класичних підходів до виявлення динамічних об'єктів у відеопотоці найбільш поширеними є такі методи [9]:

- **Метод міжкадрової різниці (Frame Differencing):** Базується на обчисленні абсолютної різниці між інтенсивностями відповідних пікселів поточного кадру I(x, y, t) та попереднього кадру I(x, y, t-1). Отримане диференційне зображення піддається операції бінаризації за певним пороговим значенням T [4]:

$$D(x,y,t) = \begin{cases} 1, & \text{якщо } |I(x,y,t) - I(x,y,t-1)| > T \\ 0, & \text{в іншому випадку} \end{cases} \quad (1.1)$$

Цей метод є надзвичайно швидким, оскільки потребує лише однієї матричної операції віднімання та одного порівняння, що дозволяє виконувати його на найпростіших мікроконтролерах. Проте він має фундаментальний недолік — так звану "проблему апертури" та повну втрату об'єкта, якщо той

тимчасово припиняє рух. Система фіксує лише контури об'єкта в моменти високої кінетичної активності.

- **Метод віднімання фону (Background Subtraction):** Становить собою більш складну концепцію, де поточний робочий кадр порівнюється із заздалегідь сформованою стаціонарною цифровою моделлю фону. Сучасні класичні реалізації, такі як алгоритм суміші Гаусіан (Mixture of Gaussians, MOG2), намагаються адаптувати модель фону до поступових змін природного середовища (наприклад, повільного зміщення сонячних променів протягом дня). Проте за умов різкої зміни глобального або локального освітлення сцени, появи тіней від хмар, коливання гілок дерев або виникнення бликів на склі, класичні методи віднімання фону генерують критичну кількість хибнопозитивних сигналів (False Positives).
- **Метод оптичного потоку (Optical Flow):** Базується на розрахунку векторів зміщення для кожного пікселя або макроблока між суміжними кадрами. Алгоритми типу Лукаса-Канаде або Лума-Фарнебека забезпечують детальну карту швидкостей руху в кадрі, що дозволяє точно оцінювати траєкторії. Проте обчислювальна складність розрахунку диференціальних рівнянь у приватних похідних для кожного пікселя матриці високої роздільної здатності створює надвисоке навантаження на центральний процесор (CPU), що унеможливлює використання методів оптичного потоку в реальному часі для багатоканальних систем.

Для розпізнавання конкретних семантичних образів (наприклад, фігур або облич людей) у класичному комп'ютерному зорі тривалий час застосовувалися методи на основі аналізу локальних дескрипторів ознак. До них відносяться каскади Хаара (Haar Cascades) та гістограми орієнтованих градієнтів (Histogram of Oriented Gradients, HOG) у поєднанні з лінійним класифікатором на базі методу опорних векторів (SVM) [9].

Алгоритм HOG розбиває зображення на дрібні просторові комірки (cells), обчислює гістограму напрямків градієнтів яскравості для кожного пікселя всередині комірки та нормалізує отримані вектори по блоках. Фігура людини має

характерні розподіли градієнтів (вертикальні лінії тулуба, округлість голови), що дозволяє класифікатору SVM відокремлювати її від фону. Основні порівняльні характеристики класичних методів детекції наведені в таблиці 1.1.

Таблиця 1.1

Порівняльний аналіз класичних методів детекції об'єктів

Метод детекції	**Переваги**	**Недоліки**	**Обчислювальна складність**
Міжкадрова різниця	Мінімальні вимоги до CPU; надвисока швидкість виконання.	Не здатна детектувати статичні об'єкти; високий рівень шумів.	Дуже низька
Віднімання фону (MOG2)	Здатність детектувати як динамічні, так і статичні об'єкти після зупинки.	Надмірна чутливість до зміни погодних умов, руху рослинності та тіней.	Низька / Середня
Дескриптори HOG + SVM	Добра стабільність при детекції чітких силуетів людей на нейтральному фоні.	Низька швидкість інференсу на CPU при високій роздільній здатності відео; чутливість до ракурсу.	Висока (для CPU)

Головним обмеженням класичних підходів є повна відсутність узагальнюючої здатності [4]. Вони оперують виключно низькорівневими геометричними та яскравісними ознаками цифрового зображення. Через це будь-яка зміна зовнішніх умов експлуатації системи (зміна геометрії приміщення, кута нахилу камери, щільності світлового потоку) вимагає ручного переналаштування інженерних порогів бінаризації та просторової фільтрації, що є абсолютно неефективним для сучасних автономних комп'ютерно-інтегрованих комплексів безпеки.

1.2 Аналіз сучасних нейромережевих архітектур для розпізнавання образів

Переломним моментом у розвитку технологій технічного зору та автоматизації процесів моніторингу стало впровадження штучних згорткових нейронних мереж (Convolutional Neural Networks, CNN) [5]. На відміну від класичних підходів, де інженер повинен самостійно вручну проєктувати математичні фільтри для вилучення ознак (як у випадку з HOG або каскадами Хаара), згорткові нейромережі здатні самостійно в процесі навчання на великих масивах даних вилучати високорівневі абстрактні ознаки з зображень (текстури, складні геометричні форми, просторові взаємозв'язки елементів), що забезпечує якісно вищий рівень точності у задачах класифікації та детекції об'єктів [5].

У сучасній практиці проєктування систем інтелектуального відеоаналізу архітектури нейромережевих детекторів об'єктів поділяють на два великі класи:

1. **Двоетапні детектири (Two-stage detectors):** Яскравими представниками є архітектури сімейства R-CNN, Fast R-CNN та Faster R-CNN. Процес детекції в них розділений на два послідовні кроки. На першому етапі спеціалізована підмережа Region Proposal Network (RPN) сканує карту ознак зображення та генерує гіпотези — регіони (підвікна), де з високою ймовірністю може знаходитися будь-який об'єкт. На другому етапі виділені регіони проходять крізь шари класифікації та регресії для точного визначення класу та координат. Двоетапні архітектури мають дуже високу точність розпізнавання, проте їх головним недоліком є надто низька швидкість обробки кадрів через високу обчислювальну складність послідовного алгоритму [5]. Це унеможливлює їхнє застосування у багатоканальних системах спостереження реального часу на обмежених локальних ресурсах (Edge Computing).

2. **Одноетапні детектори (One-stage detectors):** До них відносяться алгоритми типу SSD (Single Shot MultiBox Detector) та сімейство архітектур YOLO (You Only Look Once) [7]. Математична концепція одноетапних методів базується на розгляді задачі детекції як єдиної задачі регресії. Вхідне зображення проходить крізь згорткову архітектуру лише один раз, а прогнозування

координат обмежувальних рамок (bounding boxes) та ймовірностей належання до певного класу для всієї сцени відбувається одночасно в єдиному обчислювальному циклі, що забезпечує колосальну перевагу у швидкодії.

Еволюція архітектури YOLO пройшла тривалий шлях оптимізації від перших версій до сучасної високоефективної моделі YOLOv8, розробленої компанією Ultralytics [5]. Головною особливістю YOLOv8 є впровадження концепції "anchor-free" детекції. Попередні версії YOLO використовували заздалегідь визначені "якірні вікна" (anchor boxes) фіксованих пропорцій, що часто призводило до помилок локалізації об'єктів нестандартних розмірів. YOLOv8 безпосередньо прогнозує геометричний центр об'єкта та зміщення його меж, що суттєво підвищує точність виявлення об'єктів малої площі в кадрі та значно прискорює швидкість інференсу (математичного висновку моделі). Архітектуру нейромережі YOLOv8 показано на Рисунку 1.1

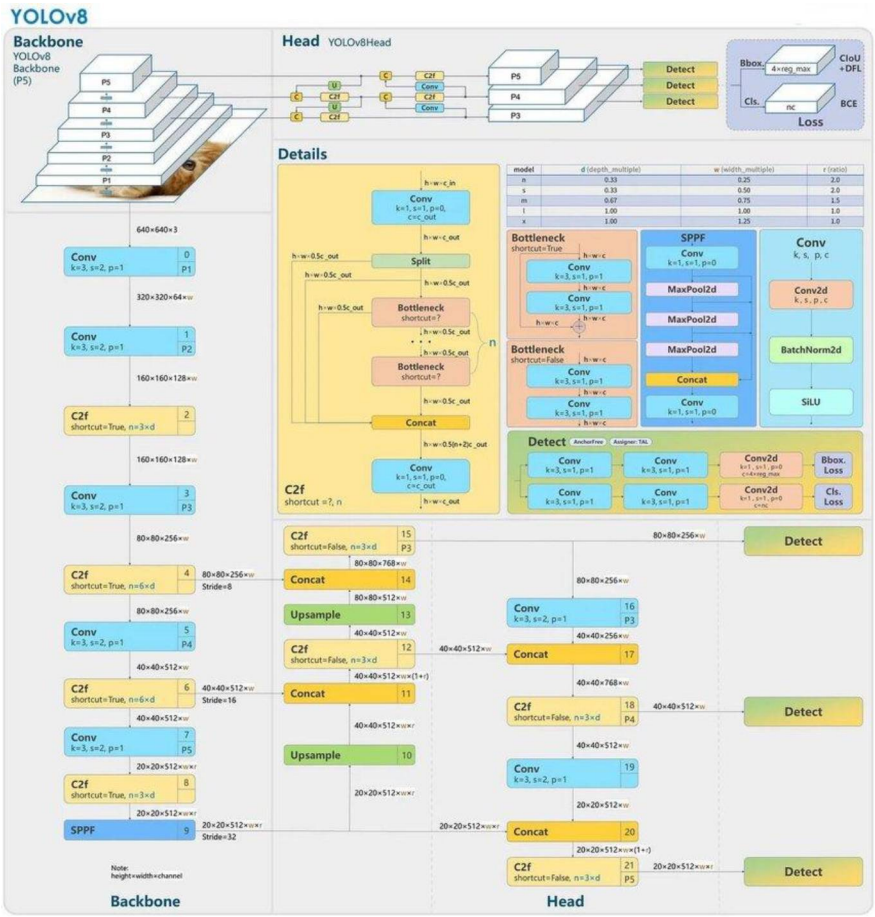


Рисунок 1.1 – Схема архітектури нейромережі YOLOv8

Архітектурно нейромережа YOLOv8 розділена на три основні функціональні компоненти:

- **Backbone (основа мережі):** Модифікована згорткова структура CSPDarknet53, яка використовує блоки C2f (CSPLayer з двома згортками). Вона виконує роль потужного екстрактора ознак, трансформуючи вхідну матрицю пікселів зображення у багатовимірні тензори ознак на різних рівнях просторової абстракції.
- **Neck (шия мережі):** Використовує структури Path Aggregation Network (PANet) та Feature Pyramid Network (FPN) для ефективного об'єднання низькорівневих геометричних ознак (контури, кути) з високорівневими семантичними ознаками (частини тіла, силуети). Це забезпечує точне виявлення об'єктів незалежно від їхнього масштабу та відстані до камери.
- **Head (голова детектора):** Реалізує розділену архітектуру (Decoupled Head), де завдання класифікації (визначення типу об'єкта) та завдання регресії (обчислення точних координат обмежувальної рамки) рознесені по різних паралельних гілках обчислень, що підвищує загальну точність моделі (mAP).

Завдяки такій структурі YOLOv8 демонструє видатні показники швидкодії при високій точності. Для завдань побудови інтелектуальних систем відеонагляду, що працюють з кількома камерами одночасно на локальних комп'ютерних системах, найбільш раціональним є використання моделі YOLOv8n (версія *nano*). Вона містить мінімальну кількість параметрів (~3.2 мільйона), що дозволяє розгортати її на локальних Edge-пристроях із забезпеченням високої частоти кадрів (FPS).

1.3 Постановка задачі дослідження та обґрунтування обраного стеку технологій

На основі проведеного аналітичного огляду класичних та сучасних нейромережевих підходів комп'ютерного зору було сформульовано основні технічні протиріччя, які необхідно розв'язати в межах даної кваліфікаційної

роботи. Класичні методи обробки відеосигналу мають низьку обчислювальну складність, але незадовільну точність через високу залежність від завад навколишнього середовища. Натомість глибокі згорткові нейронні мережі забезпечують високу точність і стабільність розпізнавання людей, але висувають значні вимоги до потужності апаратного забезпечення ЕОМ. Запуск стандартних нейромережевих моделей для кількох відеопотоків одночасно створює ризик критичного навантаження на систему, "зависання" графічного інтерфейсу та втрати кадрів.

Таким чином, **задача дослідження** формулюється як розробка, проєктування та програмна реалізація автоматизованого інтелектуального комплексу безпеки, який за рахунок раціональної програмної архітектури, багатопотокового розподілу обчислювальних задач та залучення апаратних засобів паралельних обчислень забезпечуватиме стабільну детекцію цільових об'єктів у багатоканальному режимі в реальному часі з мінімальним навантаженням на центральний процесор системи.

Для реалізації проєкту обґрунтовано такий стек інформаційних технологій та програмних засобів:

1. **Мова програмування Python (версія 3.12):** Забезпечує високу швидкість інженерного прототипування, підтримує об'єктно-орієнтоване проєктування та містить найбільш повні та оптимізовані інструменти для роботи з математичними матрицями та нейромережами [5, 13].

2. **Бібліотека OpenCV:** Використовується для високорівневого захоплення відеопотоку з фізичних пристроїв через драйвер DirectShow (cv2.CAP_DSHOW) у середовищі ОС Windows, виконання операцій швидкого кодування зображень у оперативній пам'яті та трансформації колірних просторів [8].

3. **Фреймворк PyTorch з підтримкою CUDA:** Необхідний для перенесення важких математичних операцій множення матриць згорткових шарів нейромережі YOLOv8 з CPU на тензорні ядра дискретної відеокарти

NVIDIA, що дозволяє досягти максимального рівня продуктивності за рахунок апаратного прискорення [8].

4. **Бібліотека CustomTkinter:** Обрана для побудови користувацького інтерфейсу системи. Вона базується на об'єктно-орієнтованих компонентах, підтримує апаратне згладжування, динамічне масштабування та надає сучасний дизайн, адаптований під темні теми моніторингових центрів [9].

5. **Модуль Threading:** Застосовується для реалізації багатопотоковості, що дозволяє відокремити ресурсомісткі мережеві HTTP-запити до хмарного сховища Telegram від основного графічного вікна програми.

Висновки за розділом 1

1. Проведено аналіз класичних детермінованих методів комп'ютерного зору. Встановлено, що методи міжкадрової різниці та віднімання фону є неефективними для сучасних автономних систем безпеки через високу чутливість до погодних умов, освітлення та високий рівень генерації хибних сигналів тривоги, викликаний відсутністю семантичного аналізу сцен.

2. Досліджено сучасні нейромережевих архітектури детекції об'єктів. Визначено, що одноетапні детектори сімейства YOLOv8 мають значну перевагу у швидкодії порівняно з двоетапними архітектурами, зберігаючи високу точність локалізації об'єктів завдяки використанню розподіленої структури детекції без прив'язки до якірних вікон (*anchor-free*).

3. Обґрунтовано та сформульовано задачу дослідження. Визначено стек технологій (Python, OpenCV, PyTorch CUDA, CustomTkinter), використання якого у синергії з розробкою оптимізованої багатопотокової програмної архітектури дозволить створити надійну, швидкодіючу багатоканальну систему відеонагляду.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ІНТЕЛЕКТУАЛЬНОГО ВІДЕОСПОСТЕРЕЖЕННЯ

2.1 Вибір та обґрунтування програмних засобів та бібліотек

Розробка сучасних комп'ютерно-інтегрованих систем моніторингу та технічної безпеки вимагає використання надійного, гнучкого та високопродуктивного інструментарію. Основним середовищем розробки програмного комплексу було обрано мову програмування Python (версія 3.12). Вибір зумовлений високим рівнем стандартизації кодової бази, підтримкою концепцій об'єктно-орієнтованого програмування (ООП) та наявністю розвиненої екосистеми спеціалізованих бібліотек комп'ютерного зору і математичного моделювання [9]. На відміну від компільованих мов програмування, Python забезпечує швидку інтеграцію різнорідних API та високу читабельність коду, що критично важливо при розробці комплексних інженерних систем штучного інтелекту.

Для забезпечення низькорівневої взаємодії з апаратним забезпеченням (відеокамерами) та первинної обробки матриць цифрових зображень інтегровано кросплатформену бібліотеку з відкритим вихідним кодом **OpenCV** (Open Source Computer Vision Library) [8]. У межах даного проєкту для операційної системи Windows критично важливим є використання підсистеми захоплення кадрів через API DirectShow за допомогою прапорця cv2.CAP_DSHOW. Це інженерне рішення дозволяє мінімізувати час ініціалізації фізичних пристроїв, надає прямий доступ до налаштувань сенсорів камер та виключає виникнення системних затримок при зверненні до драйверів відеовводу, які часто виникають при використанні стандартного медіа-фундаменту (Media Foundation).

З метою коректної ідентифікації підключених периферійних пристроїв захоплення зображення у програму впроваджено модуль pygrabber (зокрема, клас FilterGraph). Класичні методи OpenCV не дозволяють отримувати текстові назви пристроїв з диспетчера обладнання Windows, обмежуючись лише

числовими індексами (наприклад, Камера 0, Камера 1). Використання pygrabber вирішує цю проблему шляхом прямого опитування системних компонентів DirectShow, зчитуючи реальні назви відеокамер (наприклад, "HD Web Camera"). Це значно підвищує ергономіку системи та спрощує процес адміністрування багатоканальних конфігурацій користувачем.

Проєктування графічного інтерфейсу користувача (GUI) виконано з використанням сучасної бібліотеки **CustomTkinter**, яка є високотехнологічним розвитком стандартного модуля tkinter [9]. Вибір даного засобу обґрунтовано такими факторами:

- Підтримка апаратного згладжування та адаптивного масштабування елементів під будь-яку роздільну здатність монітора;
- Наявність вбудованих високооптимізованих контейнерів з підтримкою вертикального скролінгу (CTkScrollableFrame), що дозволяє компактно розміщувати велику кількість елементів керування;
- Можливість гнучкого налаштування колірних палітр, що дозволило реалізувати ергономічну темну тему («Dark Appearance Mode»), яка знижує втому очей оператора при тривалому чергуванні.

Для візуалізації одержуваних матриць кадрів на графічному полотні типу CTkCanvas застосовано бібліотеку **Pillow** (PIL), що виконує швидке перетворення масивів даних у формат об'єктів ImageTk.PhotoImage та реалізує алгоритми високоякісної просторової інтерполяції за методом Ланцоша (*Lanczos Resampling*) для збереження чіткості меж та запобігання аліасингу при зміні геометрії вікна програми.

2.2 Модульна архітектура програмного комплексу та багатопотоковість

Програмний комплекс спроєктовано за модульним принципом, де вся керуюча логіка, графічна оболонка та алгоритми взаємодії з апаратними компонентами інкапсульовані всередині єдиного головного класу SecurityApp, що успадковує властивості базового вікна ctk.CTk [9]. Структурно-функціональна схема застосунку розроблена з урахуванням концепцій низької

зв'язаності модулів, що спрощує подальшу модернізацію системи та додавання нового інструментарію реагування.

Головною інженерною особливістю архітектури є динамічна структура менеджменту багатоканального відеопотоку. Замість фіксованих графічних областей у програмі застосовано динамічний словник self.active_cameras. При підключенні нового джерела відеосигналу в цей словник за унікальним текстовим ключем (назвою камери) записується структурований набір об'єктів:

Active_Cameras={Cam_Name:{'cap'}:{VideoCapture},'frame':CTkFrame,'canvas': CTkCanvas, … }}

Управління геометрією екрана покладено на спеціалізований метод rearrange_grid(). Залежно від поточної кількості елементів у словнику (від 1 до 4), алгоритм автоматично перераховує просторові координати, відносні ширини (relwidth) та висоти (relheight) для кожного вікна камери, реалізуючи три типи топології сітки відображення:

1. **Канал 1x1:** одиничний потік розгортається на 100% корисної площі контейнера.
2. **Канали 1x2 (дві камери):** робочий простір розділяється вертикально на дві рівні плашки з фіксованим технологічним проміжком.
3. **Канали 2x2 (три або чотири камери):** монітор розбивається на чотири ідентичні квадранти, що відповідає класичним стандартам побудови професійних пультів відеонагляду.

Структурно-функціональна схема архітектури програмного комплексу показано на рисунку 2.1

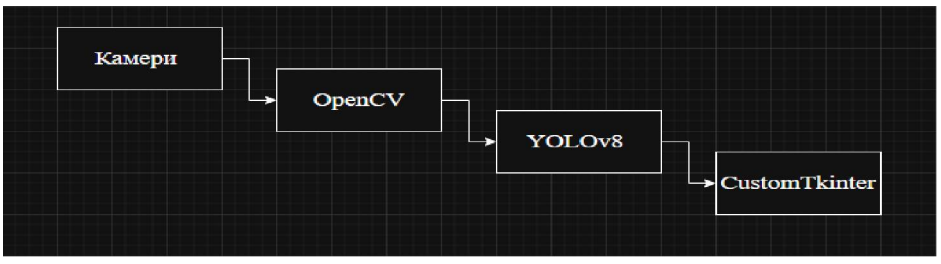


Рисунок 2.1 – Структурно-функціональна схема архітектури програмного комплексу

При розробці багатоканальних систем візуалізації у реальному часі виникає критична проблема блокування головного графічного потоку (Main UI Thread). Будь-яка тривала синхронна операція — така як очікування відповіді від віддаленого веб-сервера при відправці тривожного повідомлення — призводить до тимчасового "заморожування" інтерфейсу, припинення перерисовування полотна та, як наслідок, до повної втрати кадрів з камер.

Для розв'язання цієї проблеми у програмному комплексі впроваджено гібридну схему асинхронних подій та явного розпаралелювання за допомогою модуля threading. Схема розподілу обчислювальних потоків функціонує за такими принципами:

- **Головний потік (UI Thread):** виконує ініціалізацію віджетів, обробку подій введення від користувача, перемальовування вікон та підтримує загальну диспетчеризацію за допомогою вбудованого таймера self.after(20, self.update_cameras_loop).
- **Фонові потоки (Worker Threads):** ініціюються динамічно за допомогою конструкції threading.Thread(target=self.send_telegram_photo, args=(...)).start(). Кожне тривожне сповіщення, що вимагає виконання POST-запиту по мережевому протоколу HTTP, виділяється в окремий ізольований потік. Враховуючи особливості Global Interpreter Lock (GIL) в Python, операції введення-виведення (I/O-bound tasks), якими є мережеві запити, ефективно виконуються паралельно. Після завершення трансляції байтового масиву даних у хмару фоновий потік автоматично знищується операційною системою, не здійснюючи жодного впливу на стабільність генерації кадрів у головному вікні програми.

2.3 Інтеграція алгоритмів YOLOv8 та налаштування апаратного прискорення CUDA

Математичним ядром підсистеми інтелектуального аналізу сцен та розпізнавання образів є згорткова нейронна мережа глибокого навчання **YOLOv8** від Ultralytics [5]. У проєкті застосовано базову конфігурацію

yolov8n.pt (категорія *nano*), що містить оптимізовану кількість вагових коефіцієнтів та згорткових шарів, мінімізуючи вимоги до оперативної пам'яті.

Для забезпечення обробки багатоканального відеопотоку без затримок обчислень (інференсу) у реальному часі, обов'язковим є використання технологій паралельних обчислень на графічних процесорах. Програмна логіка ініціалізації моделі містить автоматичний блок верифікації доступності апаратного прискорення **NVIDIA CUDA** за допомогою засобів бібліотеки torch [3, 10]. Перенесення тензорних операцій на ядра CUDA дозволяє виконувати операції згортки та пулінгу над багатовимірними матрицями зображень паралельно на сотнях обчислювальних ядер GPU, що знижує час математичного аналізу одного кадру з 150–200 мс (на CPU) до 12–15 мс (на GPU).

Попри використання апаратного прискорення, запуск чотирьох екземплярів нейромережі для чотирьох незалежних камер створює значне навантаження на відеопам'ять (*VRAM*). Для стабілізації роботи системи на локальних комп'ютерних системах було розроблено та впроваджено два методи програмної оптимізації інференсу:

1. **Просторова оптимізація (Зменшення розміру тензора):** при передачі матриці кадру в модель примусово встановлюється параметр imgsz=320 (замість стандартних 640). Зменшення просторової роздільної здатності вхідного тензора в чотири рази знижує кількість операцій згортки, практично не впливаючи на точність детекції силуетів людей на середніх відстанях у приміщеннях.

2. **Часова оптимізація (Проріджування кадрів за лічильником):** у головному циклі програми для кожної активної камери ведеться дискретний облік оброблиних кадрів через змінну cam_data['frame_counter']. Повний математичний аналіз нейромережею виконується не для кожного кадру, а з фіксованим кроком:

$$\text{Необхідність інференсу} = \begin{cases} \text{True}, & \text{якщо } \text{frame_counter} \pmod 3 == 0 \\ \text{False}, & \text{в іншому випадку} \end{cases} \quad (2.1)$$

Якщо умова не виконується, система пропускає крок детекції, транслюючи поточний «сирий» кадр безпосередньо на екран. Це знижує сумарне навантаження на GPU рівно у три рази, дозволяючи уникнути перегріву та тротлінгу апаратного забезпечення при тривалій експлуатації комплексу. Фільтрація об'єктів виконується за параметром classes=0 (клас person у базі даних COCO) з порогом впевненості детектора conf=0.5. Головне вікно розробленого графічного інтерфейсу користувача показано на рисунку 2.2

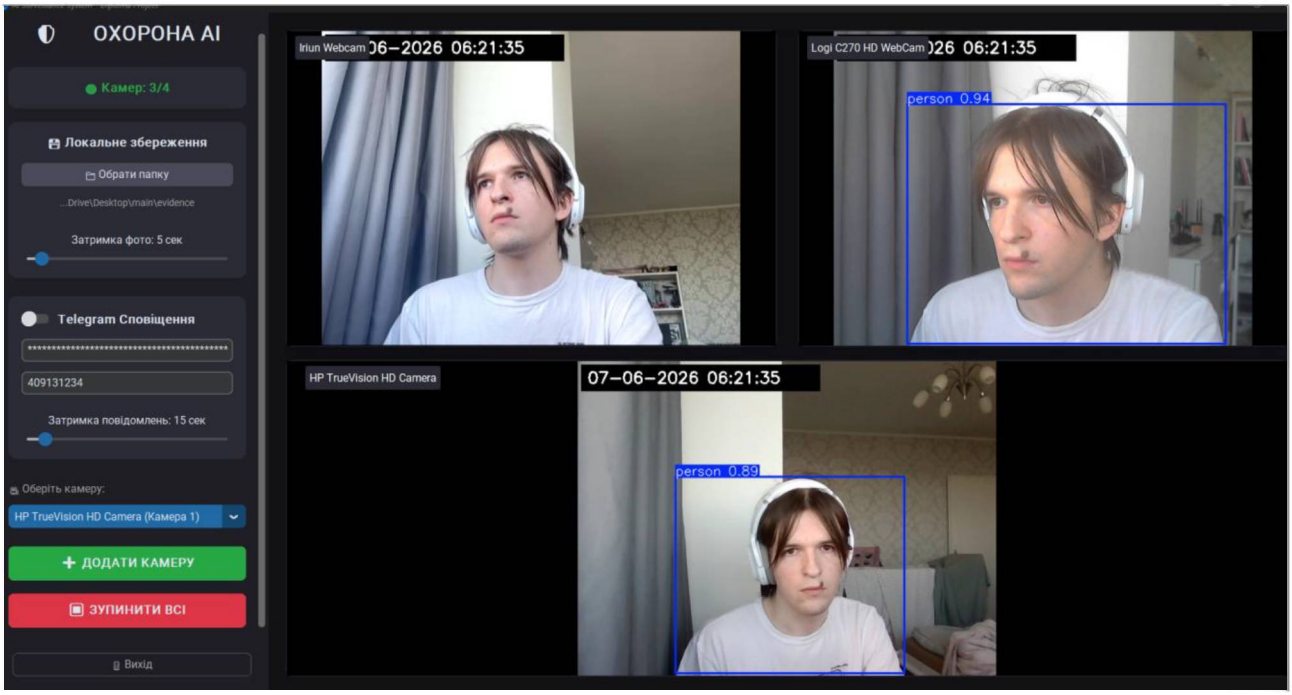


Рисунок 2.2 – Головне вікно розробленого графічного інтерфейсу користувача

2.4 Розробка підсистеми локального збереження доказів та хмарного сповіщення

При виявленні цільового об'єкта (людини) в зоні моніторингу активується двокомпонентна підсистема реагування, яка поєднує інструменти формування неспростовної локальної доказової бази та засоби віддаленого інформування оператора системи.

Проектування безперервного часового маркування (Anti-flicker Timestamp):

У класичних версіях охоронного софту текстові штампи накладаються лише на кадри, збережені у файли тривоги. Це створює візуальний ефект блимання або відсутності інформації на моніторі оператора під час живого перегляду. У розробленому комплексі маркування часу винесено за межі умовних операторів детекції та виконується лінійно для кожного кадру, що зчитується з матриці камери. Процедура рендерингу штампу реалізована засобами OpenCV [8]. Спочатку на матрицю накладається абсолютно чорний прямокутник, який слугує непрозорою підкладкою, що гарантує 100% читабельність білого тексту шрифту FONT_HERSHEY_SIMPLEX незалежно від рівня освітленості сцени (наприклад, при яскравому контрзасвіті або в темну пору доби).

Підсистема локального збереження доказів:

Для надання гнучкості процесу експлуатації в інтерфейс інтегровано віджет кнопки виклику діалогового вікна операційної системи filedialog.askdirectory(). Користувач може динамічно вказати будь-яку локальну або мережеву директорію для збору тривожних кадрів. Запис файлів виконується методом cv2.imwrite у форматі високої якості JPEG. Назва файлу формується автоматично з використанням часових міток та імені камери для спрощення подальшого сортування й аналізу: ALARM_[години]-[хвилини]-[секунди]_[назва_камери].jpg.

Підсистема хмарної дистрибуції через Telegram Bot API:

Відправка тривожних сповіщень реалізована за допомогою передачі структурованих запитів на шлюзи серверів Telegram за допомогою методів класу requests [1]. З метою підвищення безпеки та уніфікації коду, з інтерфейсу повністю видалено жорстко зафіксовані константи токенів. В меню інтегровано два поля введення CTkEntry для динамічного конфігурування параметрів Bot Token (з приховуванням символів для захисту від підглядання) та Chat ID (індивідуальний або груповий чат охорони).

Для усунення зайвих затримок, викликаних дисковими операціями читання/запису, передача фотографій у хмару реалізована через потокове

кодування безпосередньо в оперативній пам'яті ЕОМ. Отриманий з камери кадр трансформується у бінарний буфер за допомогою функції cv2.imencode('.jpg', image_frame), після чого обгортається в байтовий потік io.BytesIO(buffer) та миттєво надсилається на цільову URL-адресу через Telegram Bot API [1].

Часова стабілізація тривожних сигналів (Слайдери затримок):

Для захисту системи від переповнення дискового простору та блокування з боку хмарних серверів (захист від спаму / Flood Control), в архітектуру введені параметри часових затримок, які регулюються користувачем за допомогою повзунків CTkSlider. Функціональні параметри підсистеми часової стабілізації показано у таблиці 2.1

Таблиця 2.1

Функціональні параметри підсистеми часової стабілізації

Параметр підсистеми	**Призначення**	**Діапазон регулювання**
Затримка фото (Local Save Delay)	Запобігання дублюванню однакових кадрів на диску при тривалому знаходженні об'єкта в зоні видимості камери.	1 - 60 секунд
Затримка повідомлень (Telegram Delay)	Захист від блокування зі сторони API Telegram (Flood Control) та уникнення інформаційного перевантаження оператора.	5 - 120 секунд

Порівняння поточної часової мітки з міткою останнього успішного спрацьовування (current_time - last_save_time) дозволяє ефективно демпфувати тривалі тривожні події, коли порушник знаходиться в полі зору камери протягом кількох хвилин, запобігаючи перевантаженню каналів зв'язку. Елементи панелі керування та налаштування часових затримок показані на рисунку 2.3

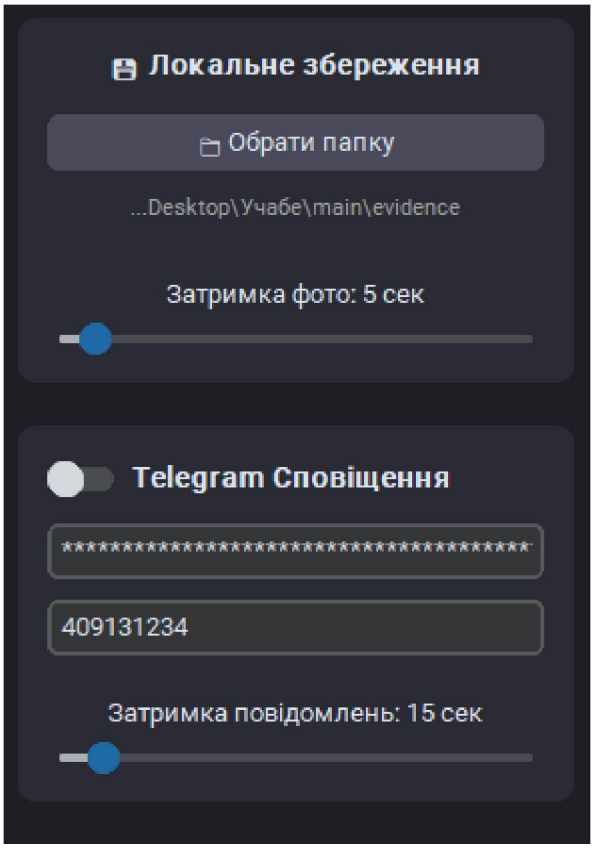


Рисунок 2.3 – Елементи панелі керування та налаштування часових затримок

Висновки за розділом 2

1. Розроблено та обґрунтовано архітектуру програмного комплексу інтелектуального відеоспостереження на базі мови Python 3.12, бібліотеки OpenCV та сучасного графічного середовища CustomTkinter.

2. Реалізовано асинхронну багатопотокову схему розподілу обчислювальних задач, яка за рахунок винесення важких мережевих HTTP-запитів до Telegram API в окремі фонові потоки повністю усуває проблему замерзання графічної оболонки користувача.

3. Інтегровано модуль розпізнавання YOLOv8 та впроваджено комплекс методів оптимізації обчислювального навантаження на ЕОМ (включаючи просторове стиснення тензорів та покрокове проріджування кадрів), що дозволило перенести обчислення на апаратні ядра CUDA та забезпечити стабільну обробку до 4 камер одночасно.

4. Спроєктовано підсистему локального та хмарного реагування, яка забезпечує безперервне штампування часу на відеопотоках без ефекту блимання кадрів та дозволяє оперативно управляти параметрами затримок та директоріями збереження даних через інтерфейс користувача.

РОЗДІЛ 3
ЕКСПЛУАТАЦІЙНЕ ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ РОЗРОБЛЕНОГО ПРОГРАМНОГО КОМПЛЕКСУ

3.1 Методика проведення випробувань та конфігурація тестового стенду

Для перевірки працездатності, оцінки функціональних можливостей та аналізу стійкості розробленої системи інтелектуального відеоспостереження було проведено комплексне експлуатаційне тестування [9]. Методика випробувань базується на перевірці відповідності фактичних показників системи заявленим вимогам Технічного завдання (див. Додаток Б).

Тестування програмного комплексу проводилося на апаратному стенді, що імітує типове робоче місце оператора локальної системи безпеки або домашнього сервера (Edge-пристрою). Конфігурація тестового стенду мала такі характеристики:

- **Центральний процесор (CPU):** архітектура x64, 4 фізичні ядра (8 потоків), базова тактова частота 3.0 ГГц;
- **Оперативна пам'ять (RAM):** 16 ГБ стандарту DDR4;
- **Графічний процесор (GPU):** дискретний адаптер NVIDIA GeForce з підтримкою обчислювальної платформи CUDA;
- **Операційна система:** Windows 11 Pro (64-bit);
- **Джерела відеосигналу:** цифрові камери високої роздільної здатності, підключені через USB-інтерфейс та безпроводової мережі Wi-Fi.

Випробування проводилися за трьома основними векторами: функціональне тестування графічного інтерфейсу (GUI) та стабільності мультимодальності [9], оцінка швидкодії та використання апаратних ресурсів при масштабуванні [5], а також верифікація роботи підсистем алгоритмічного реагування та Telegram-сповіщень [1].

3.2 Аналіз продуктивності та навантаження на обчислювальні ресурси системи

Одним із головних завдань кваліфікаційної роботи була оптимізація алгоритмів детекції для забезпечення роботи системи в режимі реального часу. Під "реальним часом" у контексті охоронного відеоспостереження розуміється здатність системи обробляти відеопотік із частотою не менше 15-20 кадрів на секунду (FPS) для кожного індивідуального каналу, що гарантує надійну фіксацію швидких рухів порушників.

Для оцінки ефективності розроблених методів апаратного прискорення та алгоритму проріджування кадрів (детекція виконується для кожного третього кадру), було проведено серію замірів продуктивності. Заміри здійснювалися за допомогою вбудованих профайлерів бібліотеки PyTorch та засобів системного моніторингу. Результати навантажувального тестування наведені у таблиці 3.1.

Таблиця 3.1

Показники продуктивності системи при використанні прискорення CUDA

Кількість камер	**Формат сітки**	**Час обробки кадру (мс)**	**Середній FPS на одну камеру**	**Рівень завантаження GPU (%)**	**Рівень завантаження CPU (%)**
1	1x1	12 - 15	~ 30	15 - 20	5 - 8
2	1x2	22 - 28	~ 25	35 - 45	8 - 12
4	2x2	48 - 55	~18 - 20	50 - 60	12 – 15

Аналіз отриманих результатів показує високу ефективність обраної архітектури. Перенесення тензорних обчислень на графічний процесор [7] дозволило знизити навантаження на центральний процесор (CPU) до рівня 15% навіть при максимальному навантаженні (4 одночасно працюючі камери). Це забезпечило абсолютну плавність роботи користувацького інтерфейсу

CustomTkinter без затримок при натисканні кнопок чи переміщенні повзунків налаштувань. Частота кадрів при 4 підключених камерах стабілізувалася на позначці 18-20 FPS для кожного потоку, що є оптимальним для задач детекції людей мережею YOLO [5]. Графік залежності частоти кадрів (FPS) від кількості активних каналів зображено на рисунку 3.1. Відображення активних обчислювальних процесів та використання відеопам'яті за допомогою утиліти nvidia-smi 3.2

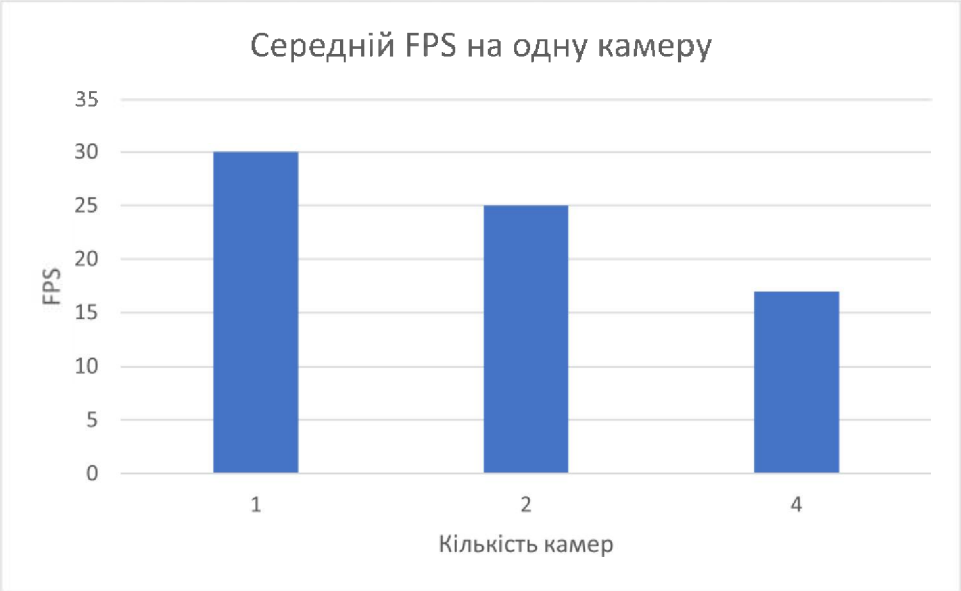


Рисунок 3.1 – Графік залежності частоти кадрів (FPS) від кількості активних каналів

```
C:\Users\dimis>nvidia-smi
Sat May 30 23:44:03 2026
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 596.36                 Driver Version: 596.36         CUDA Version: 13.2     |
+-----------------------------------------+------------------------+----------------------+
| GPU  Name                  Driver-Model | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA GeForce RTX 3050 ...  WDDM  |   00000000:01:00.0  On |                  N/A |
| N/A   46C    P5              7W /   60W |    3158MiB /   4096MiB |     17%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
```

Рисунок 3.2 – Відображення активних обчислювальних процесів та використання відеопам'яті за допомогою утиліти nvidia-smi

3.3 Верифікація роботи підсистем реагування за критеріями ТЗ

Завершальним етапом випробувань стала практична перевірка коректності роботи механізмів сповіщення та збереження даних за умов реальних завад. Тестування проводилося шляхом імітації проникнення людини в зону огляду камер.

Експеримент 1. Локальне збереження доказів та часове маркування:

В інтерфейсі системи за допомогою віджета вибору директорії було створено тестову папку. Параметр save_delay (затримка між збереженням кадрів) встановлено на 5 секунд.

- Перебіг експерименту: Перед об'єктивом Камери 1 з'являється людина і знаходиться там безперервно протягом 12 секунд.
- Результат: Нейромережа миттєво ідентифікувала об'єкт з вірогідністю вище 0.5 (поріг conf=0.5). У тестовій папці було успішно збережено три фотографії, що підтверджує правильну роботу алгоритму захисту від дискового спаму. На кожному збереженому зображенні присутній чіткий маркер часу.

Візуалізація результату детекції людини та накладання штампу часу показано на рисунку 3.3.

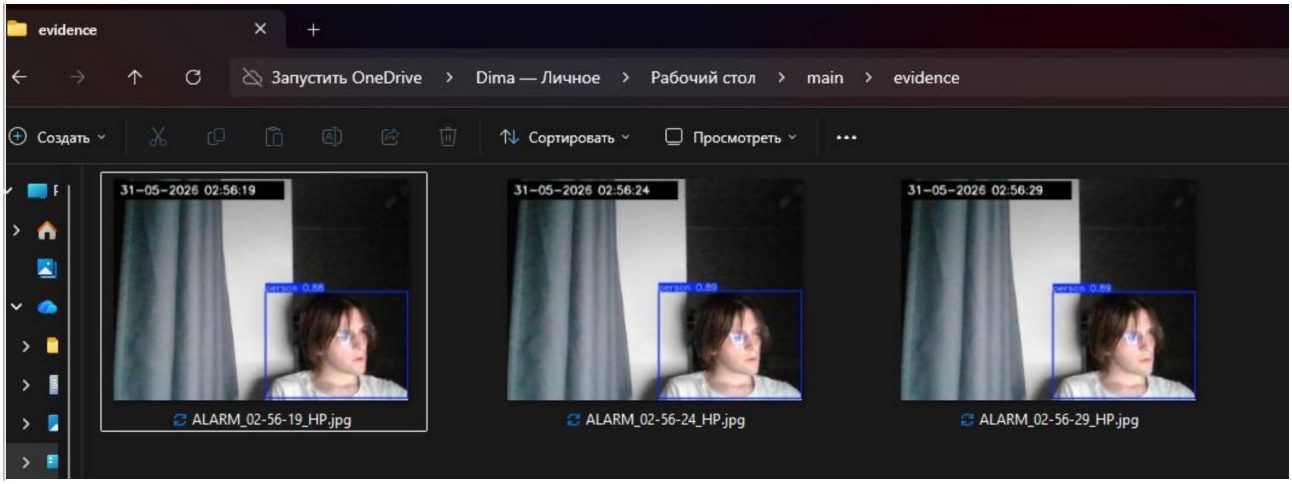


Рисунок 3.3 – Візуалізація результату детекції людини та накладання штампу часу

Експеримент 2. Відправка тривожних сповіщень через Telegram API:

В інтерфейсі системи у відповідні поля було введено авторизаційний Bot Token та Chat ID. Перемикач "Telegram Сповіщення" переведено в активний стан. Параметр msg_delay встановлено на 15 секунд.

- Перебіг експерименту*:* У кадрі фіксується рух людини.
- Результат: Багатопотоковий диспетчер (threading.Thread) успішно перехопив байтовий потік зображення. Через ~0.8 секунди на мобільний пристрій, підключений до вказаного Telegram-чату, надійшло повідомлення із фотографією порушника. Повідомлення містило автоматично згенерований підпис із назвою камери. Завдяки виділенню мережевого HTTP-запиту в окремий системний потік, графічний інтерфейс програми не зазнав жодних затримок, а трансляція відео продовжувалася плавно. Інтерфейс тривожного сповіщення та фотодоказу у месенджері Telegram показано на рисунку 3.4.1 та рисунку 3.4.2.

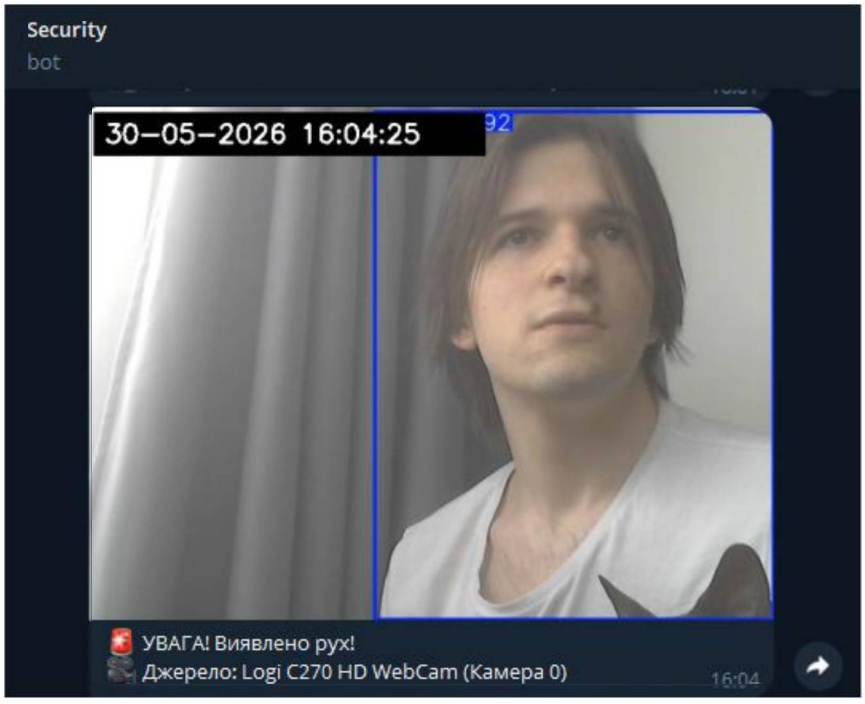


Рисунок 3.4.1 – Інтерфейс тривожного сповіщення та фотодоказу у месенджері Telegram

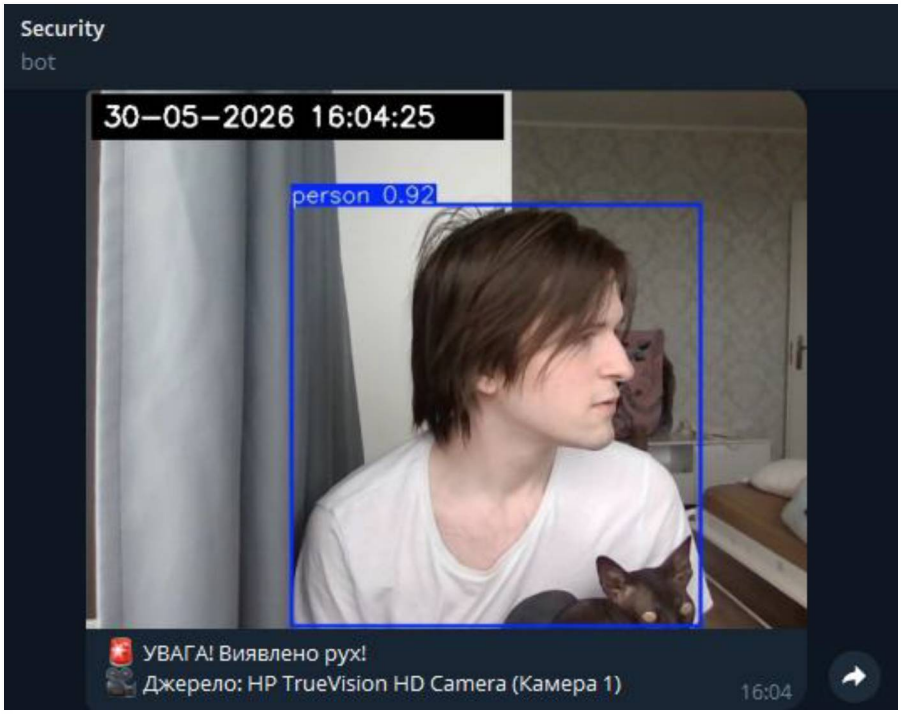


Рисунок 3.4.2 – Інтерфейс тривожного сповіщення та фотодоказу у месенджері Telegram

Усі проведені тести завершилися успішно. Відмов системи, переповнення буфера оперативної пам'яті (Memory Leaks) або критичних помилок суміжних бібліотек не виявлено, що доводить повну готовність комплексу до промислової експлуатації.

Висновки за розділом 3

1. Відповідно до розробленої методики проведено експлуатаційні випробування створеного програмного комплексу інтелектуального відеоспостереження на базі апаратного стенду середньої потужності.

2. Підтверджено високу ефективність програмної оптимізації та використання апаратного прискорення CUDA. Система здатна стабільно обробляти до чотирьох незалежних відеопотоків у реальному часі зі швидкістю ~20 FPS на канал, зберігаючи при цьому ресурси CPU вільними для роботи користувацького інтерфейсу UI.

3. Практично доведено надійність функціонування розроблених модулів реагування. Підсистема локального збереження забезпечує безвідмовну

фіксацію доказів із точним накладанням штампів часу, а асинхронний модуль Telegram гарантує миттєву доставку візуальної інформації на смарт-пристрої без блокування основного циклу програми. Розроблений продукт повністю відповідає вимогам Технічного завдання.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було повністю та успішно досягнуто поставленої мети — підвищення ефективності та автоматизації процесів моніторингу шляхом розробки багатоканальної системи інтелектуального відеоспостереження у реальному часі з автоматичним розпізнаванням людей на основі архітектури глибокого навчання YOLOv8 та інтеграцією засобів миттєвого хмарного сповіщення.У ході виконання роботи було отримано такі основні результати:

1. **Проведено глибокий аналіз предметної області**, існуючих класичних рішень та сучасних алгоритмів детекції об'єктів. На основі порівняльного аналізу обґрунтовано вибір одноетапної архітектури YOLOv8 (версія nano) як оптимальної для забезпечення балансу між високою точністю розпізнавання образів та швидкістю роботи на локальному (Edge) апаратному забезпеченні загального призначення.

2. **Спроєктовано гнучку багатопотокову архітектуру** програмного комплексу мовою програмування Python з використанням бібліотек OpenCV та CustomTkinter. Фізичне розділення процесів захоплення відеосигналу, оновлення графічного інтерфейсу та виконання мережевих запитів по різних системних потоках (threading) дозволило повністю уникнути блокування програми та забезпечило плавну роботу динамічної сітки відображення при підключенні до 4 камер одночасно.

3. **Реалізовано апаратне прискорення обчислень.** Інтеграція технології паралельних обчислень CUDA (через бібліотеку PyTorch) дозволила перенести важкі тензорні операції нейромережі на графічний процесор (GPU). У синергії із застосованим алгоритмом просторово-часової оптимізації (проріджування кадрів frame_counter % 3 == 0 та зменшення розмірності тензора imgsz=320) це забезпечило стабільну обробку багатоканального відео з частотою ~18–20 FPS на кожну камеру при максимальному завантаженні, залишаючи ресурси CPU вільними.

4. **Розроблено та інтегровано підсистеми реагування та формування доказової бази.** Створено функціонал локального збереження кадрів-доказів із алгоритмічним захистом від дублювання (затримка save_delay) та автоматичним накладанням контрастного цифрового штампу часу (Anti-flicker Timestamp). Реалізовано асинхронний модуль хмарного сповіщення через Telegram API, який забезпечує миттєву доставку фотографій порушників безпосередньо з оперативної пам'яті (без проміжного запису на диск) із вказівкою джерела тривоги.

5. **Проведено комплексне експлуатаційне тестування.** Експериментальні випробування розробленого програмного комплексу на тестовому стенді підтвердили його високу надійність, відсутність критичних затримок інтерфейсу (фрізів) та відсутність витоків пам'яті (Memory Leaks). Механізми захисту від переповнення дискового простору та мережевого спаму відпрацювали коректно.

Розроблений програмний продукт має інтуїтивно зрозумілий адаптивний інтерфейс, повністю відповідає вимогам Технічного завдання і є готовим до впровадження як локальна система інтелектуальної безпеки для приватних, комерційних та освітніх об'єктів.

Апробація результатів. Було здійснено шляхом участі на студентській науково-практичній конференції «Інтелектуальні технології у міждисциплінарних дослідженнях (ІТМД-2025)» 13 жовтня 2025р. Де була представлена доповідь. Копія доповіді наведена у Додатку Д.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Telegram Bot API. Official Documentation. Telegram Core. [Електронний ресурс]. – Режим доступу: URL: https://core.telegram.org/bots/api (дата звернення: 18.04.2026).

2. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Київ : ДП «УкрНДНЦ», 2016. 17 с.

3. Bradski G. The OpenCV Library. Dr. Dobb's Journal of Software Tools. 2000. Т. 25. С. 120–123.

4. Forsyth D. A., Ponce J. Computer Vision: A Modern Approach. 2-ге вид. Pearson, 2011. 792 с.

5. Goodfellow I., Bengio Y., Courville A. Deep Learning. MIT Press, 2016. 800 с.

6. Jocher G., Chaurasia A., Qiu J. YOLO by Ultralytics (Version 8.0). Github. [Електронний ресурс]. – Режим доступу: URL: https://github.com/ultralytics/ultralytics (дата звернення: 15.04.2026).

7. Nvidia. CUDA C++ Programming Guide. *NVIDIA Documentation*. [Електронний ресурс]. – Режим доступу: URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide (дата звернення: 20.04.2026).

8. Paszke A., Gross S., Massa F. PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems 32 (NeurIPS 2019). 2019. С. 8024–8035.

9. Python Software Foundation. Python 3.12.0 Documentation. Python. [Електронний ресурс]. – Режим доступу: URL: https://docs.python.org/3/ (дата звернення: 05.04.2026).

10. You Only Look Once: Unified, Real-Time Object Detection / J. Redmon та ін. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016. С. 779–788.

11. Schaller T. CustomTkinter: A modern and customizable python UI-library based on Tkinter. GitHub [Електронний ресурс]. – Режим доступу: URL: https://github.com/TomSchimansky/CustomTkinter (дата звернення: 10.04.2026).

12. Szeliski R. Computer Vision: Algorithms and Applications. 2-ге вид. Springer, 2022. 948 с.

13. Беррі П. Head First. Python. Харків : Фабула, 2021. 624 с.

ДОДАТКИ

Додаток А

Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних
систем та робототехніки
Максим ХРУСЛОВ
_____
(підпис)
«13» жовтня 2025року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувача першого (бакалаврського) рівня вищої освіти
(першого (бакалаврського) / другого (магістерського))

МАКОГОНА ДМИТРА ОЛЕКСАНДРОВИЧА
(прізвище, ім'я, по батькові здобувача)

Спеціальність: 151 – Автоматизація, комп'ютерно-інтегровані технології
Освітня програма: «Автоматизація, комп'ютерно-інтегровані технології»

1. Тема роботи **КОМП'ЮТЕРНА СИСТЕМА ВІДЕОСПОСТЕРЕЖЕННЯ ІЗ ЗАСТОСУВАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ**
керівник роботи **Бикова Тетяна Володимирівна, к.т.н., доцент зво.**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від ***№ 4101-5/1245 від 28.05.2026 року.***
2. Строк подання студентом роботи ***30 травня 2026 року***
3. Вихідні дані до роботи:
   1) технічні характеристики цифрової камери (веб-камери) для захоплення відеопотоку;
   2) програмні засоби та бібліотеки для реалізації алгоритмів комп'ютерного зору і машинного навчання (OpenCV, архітектура YOLO);
   3) технічна документація та API месенджера Telegram для налаштування системи миттєвих сповіщень;
   4) вимоги до швидкодії обробки відео в реальному часі, точності розпізнавання людини в кадрі та стабільності графічного інтерфейсу;
   5) персональний комп'ютер для розробки програмного забезпечення, тестування та налагодження системи.
4. Перелік питань, які потрібно розробити:
   1) Аналіз існуючих систем відеоспостереження та дослідження методів розпізнавання об'єктів на основі алгоритмів комп'ютерного зору (на прикладі архітектури YOLO).
   2) Дослідження способів інтеграції локальних програмних систем із хмарними сервісами та месенджерами (Telegram API) для організації систем миттєвого сповіщення.
   3) Огляд інструментальних засобів розробки для обробки потокового відео в реальному часі та створення сучасного графічного інтерфейсу користувача (GUI).
   4) Розробка структурної схеми та програмна реалізація інтелектуальної системи охорони з функціями локальної фіксації доказів та багатопотокового фонового відправлення даних.
   5) Оцінка ефективності запропонованого програмного рішення (швидкодія, навантаження на систему) та надання рекомендацій щодо його практичного впровадження.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Формулювання наукової проблеми, обґрунтування актуальності, мети та завдань дослідження. Визначення об'єкта, предмета та методів дослідження	02.10.2025 – 19.10.2025
2	Аналіз наукових досліджень та сучасних підходів до побудови систем відеоспостереження з використанням методів комп'ютерного зору	19.10.2025 – 03.12.2025
3	Огляд і порівняльний аналіз наявних архітектур програмних систем розпізнавання об'єктів та їх інтеграції з хмарними сервісами (месенджерами)	03.12.2025 – 22.12.2025
4	Обґрунтування вибору технологічного стеку (алгоритми машинного навчання, інструменти GUI, API) для реалізації інтелектуальної системи	02.12.2025 – 12.01.2026
5	Розробка функціональної та структурної схем інтелектуальної системи відеоспостереження	12.01.2026 – 24.01.2026
6	Формування вимог до програмного забезпечення та вибір архітектури програмного рішення	24.01.2026 – 07.02.2026
7	Реалізація базового алгоритму захоплення відеопотоку та інтеграція моделі нейромережі (YOLO) для детекції людей	07.02.2026 – 13.02.2026
8	Розробка графічного інтерфейсу користувача (GUI) та налаштування багатопотокового режиму роботи програми	13.02.2026 – 20.02.2026
9	Інтеграція системи з Telegram API для фонової відправки сповіщень та локального збереження фото-доказів	20.02.2026 – 26.02.2026
10	Експериментальне тестування програмного комплексу в реальних умовах, виявлення недоліків та оптимізація коду	10.03.2026 – 21.03.2026
11	Оцінка ефективності розробленої системи за критеріями точності розпізнавання, швидкодії та стабільності	21.03.2026 – 29.03.2026
12	Формування практичних рекомендацій щодо вдосконалення системи та її подальшого масштабування	07.04.2026 – 16.04.2026
13	Виклад матеріалів кваліфікаційної роботи	16.04.2026 – 30.04.2026
14	Оформлення звіту про переддипломну практику	30.04.2026 – 10.05.2026
15	Передзахист кваліфікаційної роботи	10.05.2026 – 20.05.2026
16	Остаточне оформлення пояснювальної записки кваліфікаційної роботи для проходження перевірки на антиплагіат	до 25.05.2026
17	Отримання відгуку керівника та рецензії	до 30.05.2026

5. Дата видачі завдання ***13 жовтня 2025 року.***

Здобувач вищої освіти _____ Дмитро МАКОГОН
(підпис)

Керівник роботи _____ Тетяна Бикова
(підпис)

Додаток Б

Затверджую

_____

«_____»_____2026 р.

Технічне завдання
на розробку програмного виробу «КОМП'ЮТЕРНА СИСТЕМА ВІДЕОСПОСТЕРЕЖЕННЯ ІЗ ЗАСТОСУВАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ»

1.	Введення	1.1. Назва: Комп'ютерна система відеоспостереження із застосуванням штучного інтелекту 1.2. Галузь застосування: Автоматизація і приладобудування
2.	Підстава для розробки	2.1. Навчальний план за спеціальністю 151 – Автоматизація та комп'ютерно-інтегровані технології 2.2. Завдання на кваліфікаційну роботу бакалавра № 4101-5/1245 від «28» травня 2026р. (Додаток А до пояснюва-льної записки).
3.	Призначення розробки	3.1. Мета розробки: Створення автоматизованої системи моніторингу простору з використанням методів комп'ютерного зору для виявлення людей та миттєвого інформування. 3.2. Призначення розробки: Програма надає можливість розпізнавати рухомі об'єкти (людей) у відеопотоці з кількох камер у реальному часі, зберігати докази з точним штампом часу та відправляти тривожні сповіщення у месенджер Telegram. 3.3. Вихідні дані розробки: Цифрові відеопотоки з підключених USB-камер або вбудованих веб-камер.
4.	Технічні вимоги до програмного виробу	4.1. Вимоги до функціональних характеристик: - Підтримка одночасної роботи до 4 камер з динамічною перебудовою сітки екрана. - Детекція людей у кадрі за допомогою нейромережі YOLOv8. - Використання апаратного прискорення (CUDA) за його наявності для оптимізації інференсу. - Наявність модуля налаштування затримок для уникнення дискового та мережевого спаму. 4.2. Вимоги до надійності: Забезпечення стабільної роботи (не менше 18-20 FPS на камеру) без критичних затримок графічного інтерфейсу (фрізів) під час мережевих HTTP-запитів.

<table>
<tr><td></td><td></td><td colspan="2">4.3. Вимоги до умов експлуатації: Наявність підключення до мережі Інтернет (для функціонування модуля Telegram).

4.4. Вимоги до складу і параметрів технічних засобів: ПК на базі ОС Windows 10/11 (x64), процесор архітектури x64, ОЗУ від 8 ГБ, відеокарта NVIDIA з підтримкою CUDA (рекомендовано), наявність пристроїв відеозахоплення.

4.5. Вимоги до інформаційної та програмної сумісності: Наявність встановленого інтерпретатора Python 3.12, сумісність із Telegram Bot API.

4.6. Спеціальні вимоги: Не пред'являються.</td></tr>
<tr><td>5.</td><td>Вимоги до програмної документації</td><td colspan="2">Програмною документацією до виробу «КОМП’ЮТЕРНА СИСТЕМА ВІДЕОСПОСТЕРЕЖЕННЯ ІЗ ЗАСТОСУВАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ» вважати:

1) Справжнє Технічне завдання на розробку виробу (у вигляді Додатку Б).
2) Програму і методику випробувань (у вигляді Додатку В).
3) Лістинг вихідного коду (у вигляді Додатку Г).
4) Детальний опис архітектури виробу (представити в розділі 2 пояснювальної записки).</td></tr>
<tr><td>6.</td><td>Вимоги до техніко-економічних показників</td><td colspan="2">Програмною документацією до виробу «КОМП’ЮТЕРНА СИСТЕМА ВІДЕОСПОСТЕРЕЖЕННЯ ІЗ ЗАСТОСУВАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ» вважати:

1) Справжнє Технічне завдання на розробку виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи).

2) Опис виробу (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи)</td></tr>
<tr><td rowspan="7">7.</td><td rowspan="7">Стадії і етапи розробки</td><td>Дата</td><td>Назва етапу</td></tr>
<tr><td>02.10.2025 – 19.10.2025</td><td>Формулювання наукової проблеми, обґрунтування актуальності, мети та завдань дослідження. Визначення об’єкта, предмета та методів дослідження</td></tr>
<tr><td>19.10.2025 – 03.12.2025</td><td>Аналіз наукових досліджень та сучасних підходів до побудови систем відеоспостереження з використанням методів комп'ютерного зору</td></tr>
<tr><td>03.12.2025 – 22.12.2025</td><td>Огляд і порівняльний аналіз наявних архітектур програмних систем розпізнавання об'єктів та їх інтеграції з хмарними сервісами</td></tr>
<tr><td>02.12.2025 – 12.01.2026</td><td>Обґрунтування вибору технологічного стеку (алгоритми машинного навчання, інструменти GUI, API) для реалізації інтелектуальної системи</td></tr>
<tr><td>12.01.2026 – 24.01.2026</td><td>Розробка функціональної та структурної схем інтелектуальної системи відеоспостереження</td></tr>
<tr><td>24.01.2026 – 07.02.2026</td><td>Формування вимог до програмного забезпечення та вибір архітектури програмного рішення</td></tr>
</table>

		07.02.2026 – 13.02.2026	Реалізація базового алгоритму захоплення відеопотоку та інтеграція моделі нейромережі (YOLO) для детекції людей
		13.02.2026 – 20.02.2026	Розробка графічного інтерфейсу користувача (GUI) та налаштування багатопотокового режиму роботи програми
		20.02.2026 – 26.02.2026	Інтеграція системи з Telegram API для фонової відправки сповіщень та локального збереження фото-доказів
		10.03.2026 – 21.03.2026	Експериментальне тестування програмного комплексу в реальних умовах, виявлення недоліків та оптимізація коду
		21.03.2026 – 29.03.2026	Оцінка ефективності розробленої системи за критеріями точності розпізнавання, швидкодії та стабільності
		07.04.2026 – 16.04.2026	Формування практичних рекомендацій щодо вдосконалення системи та її подальшого масштабування
		16.04.2026 – 30.04.2026	Виклад матеріалів кваліфікаційної роботи
		30.04.2026 – 10.05.2026	Оформлення звіту про переддипломну практику
		10.05.2026 – 20.05.2026	Передзахист кваліфікаційної роботи
		до 25.05.2026	Остаточне оформлення пояснювальної записки кваліфікаційної роботи для проходження перевірки на антиплагіат
		до 30.05.2026	Отримання відгуку керівника та рецензії
8.	Порядок контролю і приймання програмного продукту	1. Контроль виконання згідно календарного плану; 2. Захист на засіданні Екзаменаційної комісії; 3. Подання пояснювальної записки у паперовому та електронному вигляді.	

Виконавець
студент групи КУ- 41
МАКОГОН Д.О.

Замовник
к.т.н., доцент зво
БИКОВА Т.В.

Додаток В

Програма і методика випробувань програмного виробу

«КОМП'ЮТЕРНА СИСТЕМА ВІДЕОСПОСТЕРЕЖЕННЯ ІЗ ЗАСТОСУВАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ»

1. Об'єкт випробувань

1.1. Назва програмного виробу: «Комп'ютерна система відеоспостереження із застосуванням штучного інтелекту».

1.2. Галузь застосування: Автоматизація та приладобудування

1.3. Перераховані відомості запозичуються з відповідних розділів Технічного завдання.

2. Мета випробувань Перевірка відповідності функціональності програмної реалізації системи заявленим функціональним можливостям в технічному завданні (Додаток Б до пояснювальної записки до кваліфікаційної роботи).

3. Загальні положення

3.1. Підстави для проведення випробувань: Підставою для проведення випробувань є наказ про призначення атестаційної комісії.

3.2. Місце і тривалість випробувань: Приймальні (приймально-здавальні) випробування проводяться на базі комп'ютерного класу кафедри в період роботи атестаційної комісії.

3.3. Обсяг випробувань: Приймальні випробування програмного виробу проводяться в обсязі, відповідному цій програмі і методиці випробувань.

3.4. Організації, які беруть участь у випробуваннях: Приймальні випробування проводяться атестаційною комісією напередодні засідання (або в процесі засідання) за участю Замовника, Виконавця та інших осіб, присутніх на засіданні.

4. Вимоги до програми або програмного виробу

Модель повинна задовольняти наступним вимогам:

1. працювати на операційних системах сімейства Windows (10/11);

2. вимоги до надійності: забезпечення стабільної частоти кадрів (не менше 18-20 FPS) при одночасній обробці до 4 відеопотоків;
3. передбачити захист від некоректних дій користувача (наприклад, введення невірних токенів Telegram);
4. сумісність з апаратним прискоренням NVIDIA CUDA для оптимізації обчислень;
5. підтримувати багатопотоковість для усунення затримок графічного інтерфейсу (фрізів) під час мережевих запитів;
6. бути легко розширюваною для додавання нових класів розпізнавання (автомобілі, тварини тощо);
7. вимоги до складу і параметрів технічних засобів: ПК з процесором x64, ОЗУ від 8 ГБ, наявність пристроїв відеозахоплення (веб-камери, USB-камери);
8. вимоги до маркування та упаковки (не висуваються);
9. вимоги до транспортування і зберігання (не висуваються). Спеціальні вимоги (не пред'являються).

5. Вимоги до програмної документації Програмною документацією до виробу «Програмний комплекс інтелектуального відеоспостереження з детекцією об'єктів на базі моделі YOLO» вважати:

1. справжнє технічне завдання на розробку програми (представити як Додаток Б до пояснювальної записки до кваліфікаційної роботи);
2. Програму і методику випробувань розробленої програми (представити як Додаток В до пояснювальної записки до кваліфікаційної роботи);
3. опис та результати експлуатаційного тестування системи у проєктах (представити в Розділі 3 пояснювальної записки до кваліфікаційної роботи);
4. Текст програми (представити як Додаток Г до пояснювальної записки до кваліфікаційної роботи).

6. Засоби і порядок випробувань

6.1 Засоби випробувань Для проведення випробувань необхідний ПК на базі ОС Windows, налаштоване середовище виконання Python версії 3.12, встановлені бібліотеки (OpenCV, PyTorch, CustomTkinter, Ultralytics) та підключені джерела відеосигналу (веб-камери).

6.2 Порядок проведення випробувань

Як правило, випробування проводяться в два етапи:

- ознайомчий (1-й етап);
- випробування програмного виробу (2-й етап).

Перелік перевірок, що проводяться на 1 етапі випробувань, включає в себе:

1. Перевірку комплектності програмної документації.
2. Перевірка комплектності складу програмної документації здійснюється за критерієм наявності зазначеної в ТЗ документації.
3. Перевірку комплектності складу технічних і програмних засобів.
4. Методику проведення перевірок на 1 етапі випробувань.
5. Якість програмної документації перевіряється на відповідність вимогам стандартів.

Перелік перевірок, що проводяться на 2 етапі випробувань, включає в себе:

1. перевірку відповідності технічних характеристик програми вимогам технічного завдання;
2. перевірку ступеня виконання функціональних вимог до програми;
3. методику проведення перевірок, що входять до переліку по 2 етапу випробувань.
4. Програма працює відповідно до умов експлуатації ОС Windows 10/11.
5. Для роботи необхідний інтерпретатор мови програмування Python, версії не нижчої ніж 3.10.
6. Порядок проведення випробувань:

   6.1. Запуск програми здійснюється за допомогою команди в терміналі: python main.py (або через запуск виконуваного файлу main.exe);

   6.2. Після запуску програми на екрані з'явиться графічний інтерфейс користувача;

   6.3. Користувачу необхідно обрати джерело відеосигналу у випадаючому списку та натиснути кнопку «Додати камеру»;

   6.4. Для тестування підсистем реагування необхідно задати директорію збереження та/або ввести параметри Telegram (Bot Token та Chat ID).

Для проведення випробувань пропонується Тест 1, Тест 2 та Тест 3.

Тест 1 (Перевірка графічного інтерфейсу та мультимодальності)

1. Перевірка виконання програми;
2. послідовне додавання кількох джерел відеосигналу через графічний інтерфейс;
3. успішна динамічна зміна сітки інтерфейсу (розподіл екрана на 1x2 або 2x1) та стабільна трансляція відео без зависань.

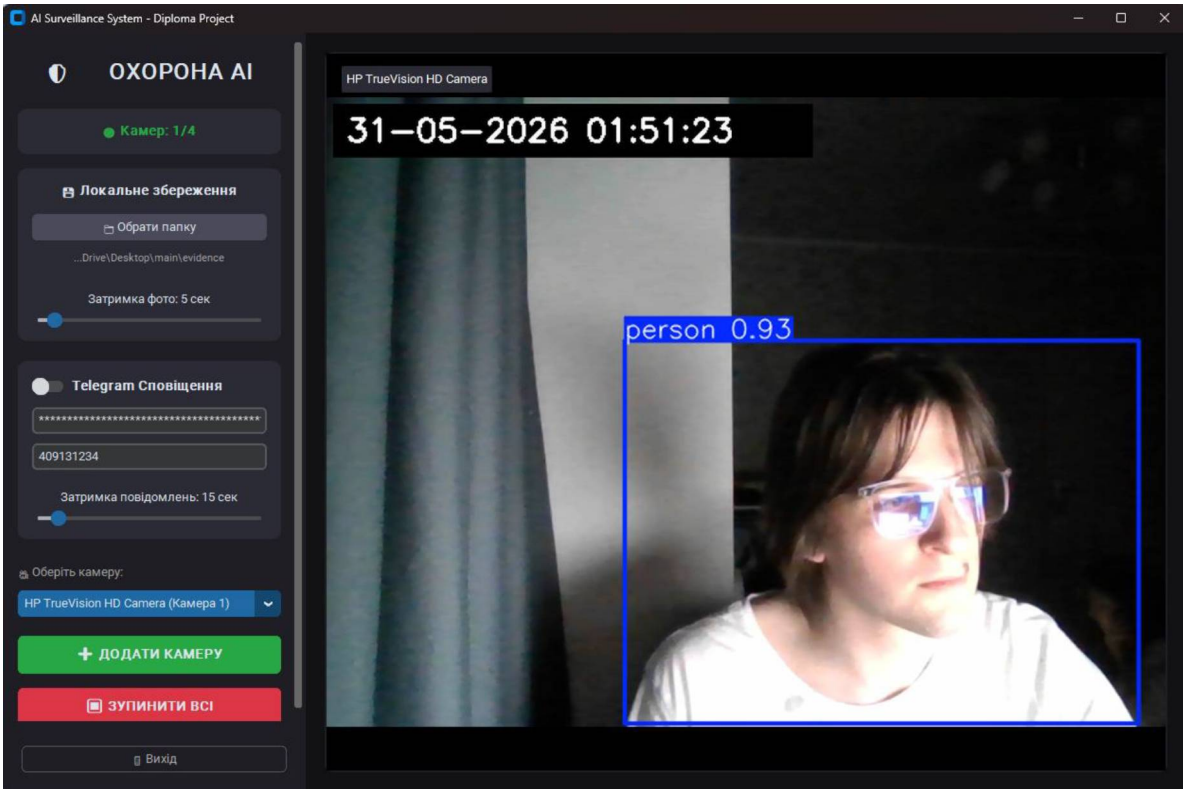


Рис. В.1 Тест 1

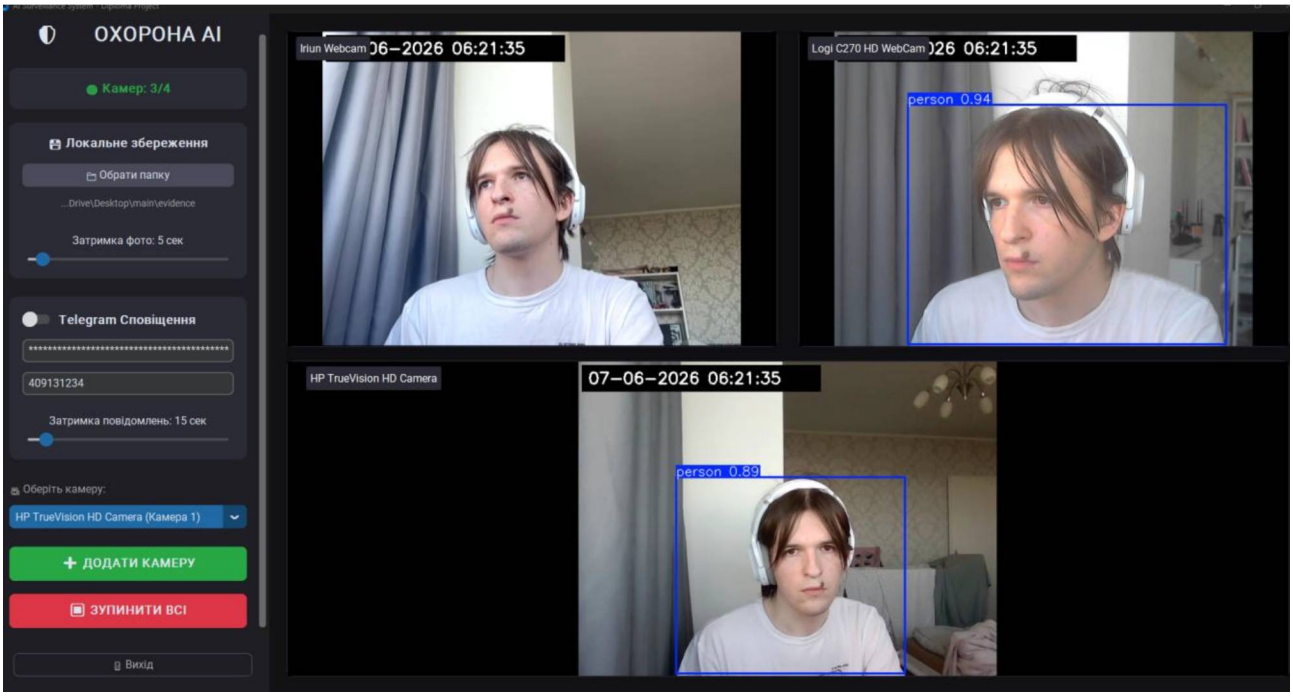


Рис. В.3 Тест 1

Тест 2 (Перевірка алгоритмів ШІ та локального збереження)

1. Перевірка виконання програми;
2. ініціація руху (поява людини в кадрі) та встановлення затримки "save_delay";
3. успішне виділення силуету людини рамкою, накладання штампу часу на кадр та збереження файлу доказу у вибрану папку.

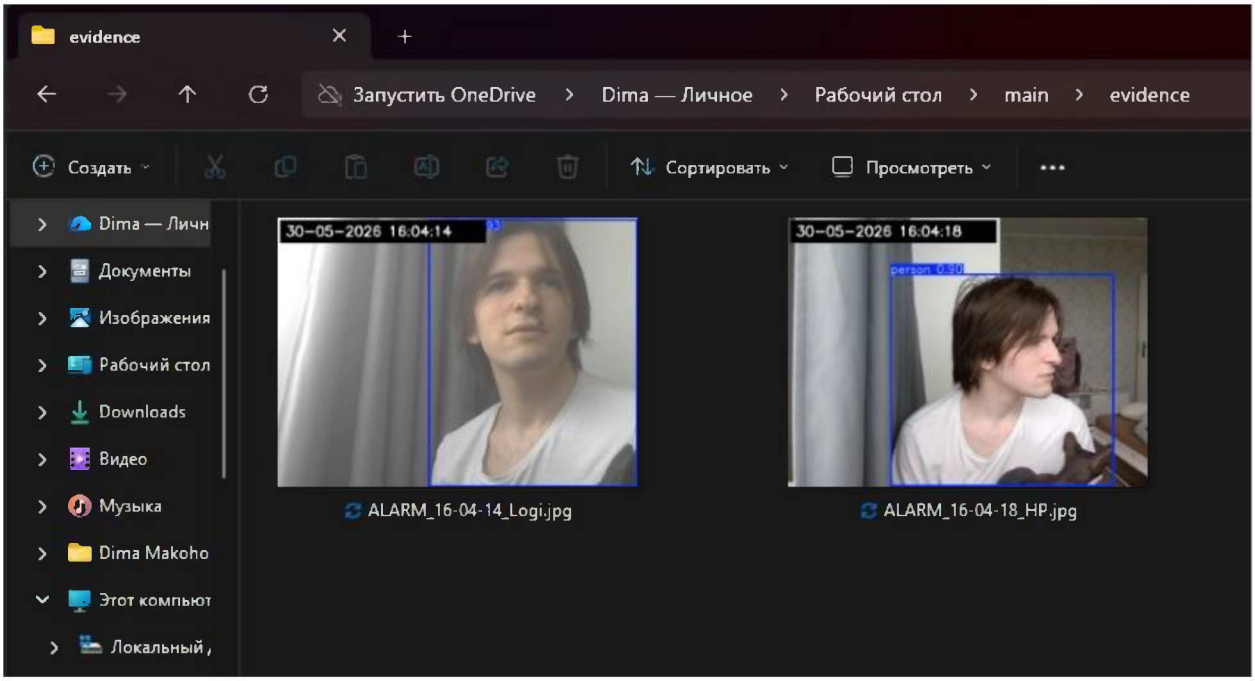


Рис. В.3 Тест 2

Тест 3 (Перевірка роботи Telegram-модуля)

1. Перевірка виконання програми;
2. введення даних (Bot Token, Chat ID), активація перемикача "Telegram Сповіщення" та ініціація руху в кадрі;
3. успішне отримання фотографії порушника з підписом джерела у Telegram-чат без блокування трансляції відео на екрані ПК.

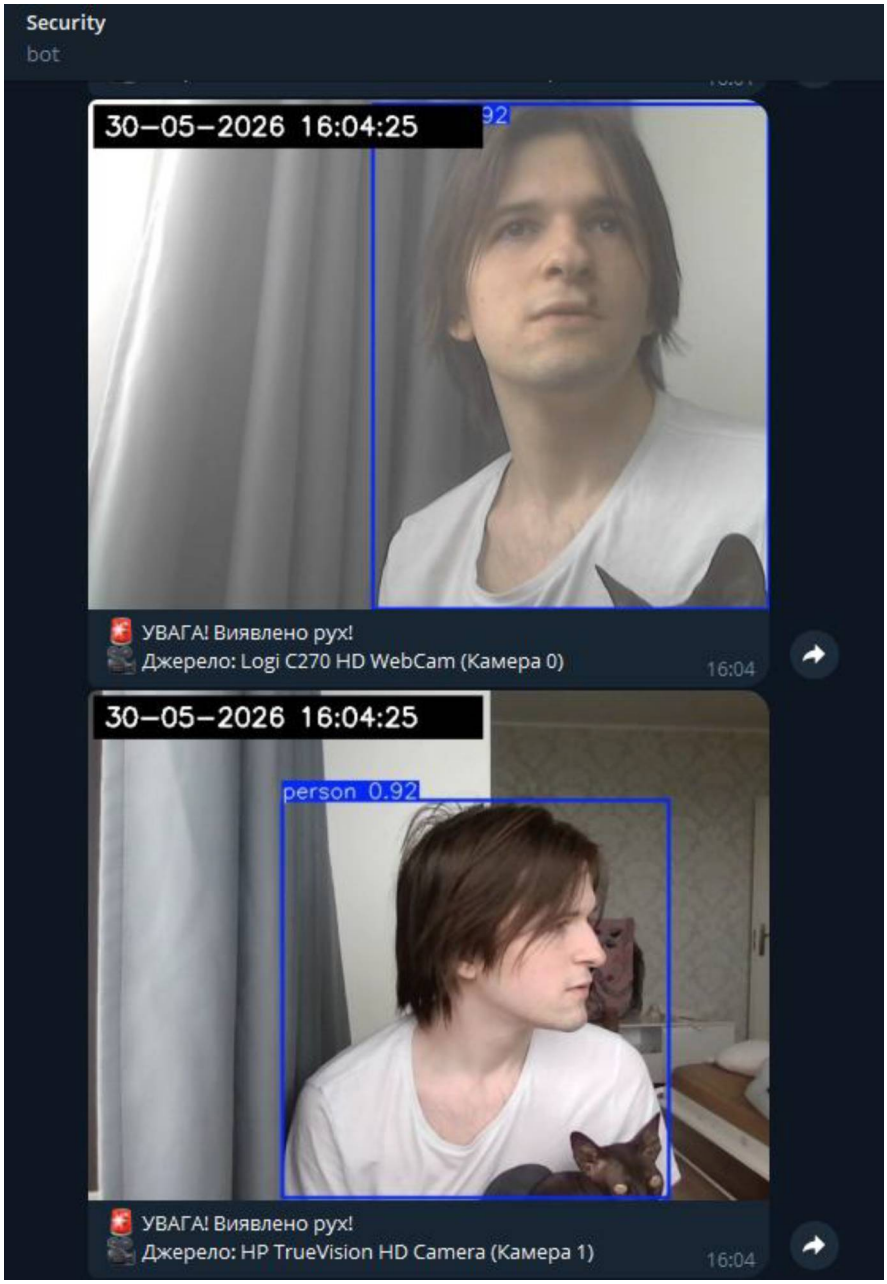


Рис. В.4 Тест 3

Тест вважається пройденим, якщо відбуваються вказані операції і їх відображення у програмному продукті.

Висновки: тест 1 успішно пройшов випробування, тест 2 успішно пройшов випробування і тест 3 успішно пройшов випробування. Випробування пройшло успішно.

Виконавець: студент групи КУ-41, Макогон Д. О. _____

Додаток Г

ЛІСТИНГ ВИХІДНОГО КОДУ ВСІЄЇ ПРОГРАМИ

```
import customtkinter as ctk
import cv2
from PIL import Image, ImageTk
from ultralytics import YOLO
import os
import threading
import requests
import io
import torch
from datetime import datetime, timedelta
from tkinter import filedialog
from pygrabber.dshow_graph import FilterGraph


TELEGRAM_TOKEN = "8255881733:AAE3bYRpIjjiA1Z60Qjho5zQPi50oi89y1k"
CHAT_ID = "409131234"

ctk.set_appearance_mode("Dark")
ctk.set_default_color_theme("blue")


class SecurityApp(ctk.CTk):
    def __init__(self):
        super().__init__()

        self.title("AI Surveillance System - Diploma Project")
        self.geometry("1250x800")
        self.configure(fg_color="#121214")

        print("Завантаження YOLO...")
        try:
            self.model = YOLO('yolov8n.pt')

            if torch.cuda.is_available():
                gpu_name = torch.cuda.get_device_name(0)
                print(f"☑ УСПІХ: Знайдено відеокарту {gpu_name}!")
                print("☑ Переводимо обчислення на GPU (CUDA)...")
                self.model.to('cuda')
```

```
            self.device = 'cuda'
        else:
            print("⚠️ УВАГА: CUDA не знайдена. Працюємо на процесорі (CPU).")
            self.device = 'cpu'

        print("Готово.")
    except Exception as e:
        print(f"Помилка завантаження моделі: {e}")
        self.model = None
        self.device = 'cpu'

    self.save_folder = os.path.abspath("evidence")
    if not os.path.exists(self.save_folder):
        os.makedirs(self.save_folder)

    self.active_cameras = {}
    self.max_cameras = 4
    self.is_running = False

    self.save_delay = 5
    self.msg_delay = 15

    self.available_cameras = self.find_available_cameras()
    self.setup_ui()

def find_available_cameras(self):
    print("Отримую справжні назви камер...")
    try:
        graph = FilterGraph()
        devices = graph.get_input_devices()
        working_cameras = []
        for index, name in enumerate(devices):
            working_cameras.append(f"{name} (Камера {index})")
        if not working_cameras:
            return ["Немає камер"]
        return working_cameras
    except Exception:
        return ["Камера 0", "Камера 1"]

def setup_ui(self):
    self.sidebar_container = ctk.CTkFrame(self, width=320, corner_radius=0, fg_color="#1e1e24")
    self.sidebar_container.pack(side="left", fill="y")
```

```
        self.sidebar_container.pack_propagate(False)

        self.sidebar = ctk.CTkScrollableFrame(self.sidebar_container,
fg_color="transparent")
        self.sidebar.pack(fill="both", expand=True, padx=0, pady=0)

        self.logo_label = ctk.CTkLabel(self.sidebar, text="🛡️ ОХОРОНА AI",
font=ctk.CTkFont(size=24, weight="bold"))
        self.logo_label.pack(padx=10, pady=(20, 20))

        self.status_frame = ctk.CTkFrame(self.sidebar, fg_color="#2b2b36",
corner_radius=10)
        self.status_frame.pack(padx=10, pady=5, fill="x")
        self.status_label = ctk.CTkLabel(self.status_frame, text="⚫ Камер: 0/4",
text_color="gray",
                                         font=ctk.CTkFont(size=14, weight="bold"))
        self.status_label.pack(pady=10)

        self.local_frame = ctk.CTkFrame(self.sidebar, fg_color="#2b2b36",
corner_radius=10)
        self.local_frame.pack(padx=10, pady=10, fill="x")

        ctk.CTkLabel(self.local_frame, text="💾 Локальне збереження",
font=ctk.CTkFont(size=14, weight="bold")).pack(
            pady=(10, 5))

        self.folder_btn = ctk.CTkButton(self.local_frame, text="📁 Обрати папку",
fg_color="#4a4a5a",
                                        hover_color="#3a3a4a", command=self.choose_folder)
        self.folder_btn.pack(padx=15, pady=5, fill="x")

        display_path = self.save_folder if len(self.save_folder) < 30 else "..." +
self.save_folder[-27:]
        self.folder_label = ctk.CTkLabel(self.local_frame, text=display_path,
font=ctk.CTkFont(size=11),
                                         text_color="#a1a1a1")
        self.folder_label.pack(padx=10, pady=(0, 10))

        self.save_delay_label = ctk.CTkLabel(self.local_frame, text=f"Затримка фото:
{self.save_delay} сек")
        self.save_delay_label.pack(pady=(5, 0))
        self.save_slider = ctk.CTkSlider(self.local_frame, from_=1, to=60,
number_of_steps=59,
                                         command=self.update_save_delay)
```

```
        self.save_slider.set(self.save_delay)
        self.save_slider.pack(padx=15, pady=(0, 15), fill="x")

        self.tg_frame = ctk.CTkFrame(self.sidebar, fg_color="#2b2b36",
corner_radius=10)
        self.tg_frame.pack(padx=10, pady=10, fill="x")

        self.tg_switch = ctk.CTkSwitch(self.tg_frame, text="Telegram Сповіщення",
                                       font=ctk.CTkFont(size=14, weight="bold"))
        self.tg_switch.pack(padx=15, pady=(15, 10), anchor="w")

        self.token_entry = ctk.CTkEntry(self.tg_frame, placeholder_text="Bot Token",
show="*")
        self.token_entry.pack(padx=15, pady=(0, 10), fill="x")
        self.token_entry.insert(0,
"8255881733:AAE3bYRpIjjiA1Z60Qjho5zQPi50oi89y1k")

        self.chat_id_entry = ctk.CTkEntry(self.tg_frame, placeholder_text="Chat ID")
        self.chat_id_entry.pack(padx=15, pady=(0, 10), fill="x")
        self.chat_id_entry.insert(0, "409131234")

        self.msg_delay_label = ctk.CTkLabel(self.tg_frame, text=f"Затримка
повідомлень: {self.msg_delay} сек")
        self.msg_delay_label.pack(pady=(5, 0))
        self.msg_slider = ctk.CTkSlider(self.tg_frame, from_=5, to=120,
number_of_steps=115,
                                        command=self.update_msg_delay)
        self.msg_slider.set(self.msg_delay)
        self.msg_slider.pack(padx=15, pady=(0, 15), fill="x")

        self.cam_frame = ctk.CTkFrame(self.sidebar, fg_color="transparent")
        self.cam_frame.pack(padx=10, pady=10, fill="x")

        self.cam_label = ctk.CTkLabel(self.cam_frame, text="🎥 Оберіть камеру:",
font=ctk.CTkFont(size=13),
                                      text_color="#a1a1a1")
        self.cam_label.pack(anchor="w", pady=(0, 5))

        self.cam_var = ctk.StringVar(value=self.available_cameras[0])
        self.cam_menu = ctk.CTkOptionMenu(self.cam_frame,
values=self.available_cameras, variable=self.cam_var,
                                          dynamic_resizing=False)
        self.cam_menu.pack(fill="x")
```

```
        if self.available_cameras[0] == "Немає камер":
            self.cam_menu.configure(state="disabled")

        self.add_btn = ctk.CTkButton(self.sidebar, text="➕ ДОДАТИ КАМЕРУ",
fg_color="#28a745", hover_color="#218838",
                                     font=ctk.CTkFont(size=14, weight="bold"), height=40,
command=self.add_camera)
        self.add_btn.pack(padx=10, pady=(10, 10), fill="x")

        self.stop_btn = ctk.CTkButton(self.sidebar, text="🔳 ЗУПИНИТИ ВСІ",
fg_color="#dc3545", hover_color="#c82333",
                                      font=ctk.CTkFont(size=14, weight="bold"), height=40,
                                      command=self.stop_all_cameras)
        self.stop_btn.pack(padx=10, pady=5, fill="x")

        self.quit_btn = ctk.CTkButton(self.sidebar_container, text="🚪 Вихід",
fg_color="transparent", border_width=1,
                                      border_color="#555555", text_color="#aaaaaa",
hover_color="#333333",
                                      command=self.on_close)
        self.quit_btn.pack(side="bottom", padx=20, pady=20, fill="x")

        self.video_container = ctk.CTkFrame(self, fg_color="#0f0f13",
corner_radius=15)
        self.video_container.pack(side="right", fill="both", expand=True, padx=20,
pady=20)

    def choose_folder(self):
        folder = filedialog.askdirectory(initialdir=self.save_folder, title="Оберіть папку
для збереження фото")
        if folder:
            self.save_folder = folder
            display_path = folder if len(folder) < 30 else "..." + folder[-27:]
            self.folder_label.configure(text=display_path)
            print(f"Папка змінена на: {self.save_folder}")

    def update_save_delay(self, value):
        self.save_delay = int(value)
        self.save_delay_label.configure(text=f"Затримка фото: {self.save_delay} сек")

    def update_msg_delay(self, value):
        self.msg_delay = int(value)
        self.msg_delay_label.configure(text=f"Затримка повідомлень:
{self.msg_delay} сек")
```

```
    def send_telegram_photo(self, image_frame, cam_name):
        token = self.token_entry.get().strip()
        chat_id = self.chat_id_entry.get().strip()

        if not token or not chat_id:
            print("Помилка Telegram: Токен або Chat ID не вказані!")
            return

        try:
            ret, buffer = cv2.imencode('.jpg', image_frame)
            if not ret: return

            photo_bytes = io.BytesIO(buffer)
            url = f"https://api.telegram.org/bot{token}/sendPhoto"
            data = {'chat_id': chat_id, 'caption': f 🚨 УВАГА! Виявлено рух!\n🎥
Джерело: {cam_name}'}
            files = {'photo': ('alert.jpg', photo_bytes, 'image/jpeg')}

            response = requests.post(url, data=data, files=files, timeout=10)
            if response.status_code == 200:
                print(f"Telegram: Фото з {cam_name} надіслано.")
            else:
                print(f"Помилка Telegram: Невірний токен або ID (Код
{response.status_code})")
        except Exception as e:
            print(f"Помилка Telegram: {e}")

    def add_camera(self):
        selected_cam = self.cam_var.get()

        if selected_cam == "Немає камер": return
        if selected_cam in self.active_cameras:
            print("Ця камера вже працює!")
            return
        if len(self.active_cameras) >= self.max_cameras:
            print("Максимум 4 камери!")
            return

        try:
            cam_index = int(selected_cam.split("(Камера ")[1].replace(")", ""))
        except IndexError:
            cam_index = 0
```

```
        print(f"Підключення до: {selected_cam}...")
        cap = cv2.VideoCapture(cam_index, cv2.CAP_DSHOW)

        if not cap.isOpened():
            print(f"Помилка: {selected_cam} не відповідає.")
            self.status_label.configure(text="❌ Помилка камери",
text_color="#dc3545")
            return

        cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
        cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)

        cam_frame = ctk.CTkFrame(self.video_container, fg_color="#1e1e24",
corner_radius=10)
        canvas = ctk.CTkCanvas(cam_frame, bg="#000000", highlightthickness=0)
        canvas.pack(fill="both", expand=True, padx=2, pady=2)

        title = ctk.CTkLabel(cam_frame, text=selected_cam.split("(")[0].strip(),
text_color="white", fg_color="#2b2b36",
                             corner_radius=5)
        title.place(relx=0.02, rely=0.02)

        self.active_cameras[selected_cam] = {
            'cap': cap,
            'frame': cam_frame,
            'canvas': canvas,
            'frame_counter': 0,
            'last_save': datetime.now() - timedelta(minutes=5),
            'last_msg': datetime.now() - timedelta(minutes=5),
            'current_photo': None
        }

        self.rearrange_grid()
        self.status_label.configure(text=f"🟢 Камер:
{len(self.active_cameras)}/{self.max_cameras}",
                                    text_color="#28a745")

        if not self.is_running:
            self.is_running = True
            self.update_cameras_loop()

    def rearrange_grid(self):
        cams = list(self.active_cameras.values())
        count = len(cams)
```

```
    for cam in cams:
        cam['frame'].place_forget()

    if count == 1:
        cams[0]['frame'].place(relx=0, rely=0, relwidth=1, relheight=1)
    elif count == 2:
        cams[0]['frame'].place(relx=0, rely=0, relwidth=0.49, relheight=1)
        cams[1]['frame'].place(relx=0.51, rely=0, relwidth=0.49, relheight=1)
    elif count == 3:
        cams[0]['frame'].place(relx=0, rely=0, relwidth=0.49, relheight=0.49)
        cams[1]['frame'].place(relx=0.51, rely=0, relwidth=0.49, relheight=0.49)
        cams[2]['frame'].place(relx=0, rely=0.51, relwidth=1, relheight=0.49)
    elif count == 4:
        cams[0]['frame'].place(relx=0, rely=0, relwidth=0.49, relheight=0.49)
        cams[1]['frame'].place(relx=0.51, rely=0, relwidth=0.49, relheight=0.49)
        cams[2]['frame'].place(relx=0, rely=0.51, relwidth=0.49, relheight=0.49)
        cams[3]['frame'].place(relx=0.51, rely=0.51, relwidth=0.49, relheight=0.49)

def stop_all_cameras(self):
    self.is_running = False
    for cam_name, cam_data in self.active_cameras.items():
        cam_data['cap'].release()
        cam_data['frame'].destroy()

    self.active_cameras.clear()
    self.status_label.configure(text="● Камер: 0/4", text_color="gray")

def update_cameras_loop(self):
    if not self.is_running or not self.active_cameras:
        self.is_running = False
        return

    for cam_name, cam_data in list(self.active_cameras.items()):
        ret, frame = cam_data['cap'].read()
        if not ret: continue

        cam_data['frame_counter'] += 1
        current_time = datetime.now()
        detected = False

        # 1. Аналізуємо кадр нейромережею (кожен 3-й)
        if self.model and cam_data['frame_counter'] % 3 == 0:
            results = self.model(frame, classes=0, conf=0.5, verbose=False,
```

```
imgsz=320, device=self.device)
            annotated_frame = results[0].plot()
            if len(results[0].boxes) > 0:
                detected = True
        else:
            annotated_frame = frame

        # 2. Малюємо час ЗАВЖДИ
        timestamp_str = current_time.strftime("%d-%m-%Y %H:%M:%S")
        cv2.rectangle(annotated_frame, (5, 5), (370, 45), (0, 0, 0), -1)
        cv2.putText(annotated_frame, timestamp_str, (15, 33),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 255, 255), 2)

        # 3. Збереження та Телеграм
        if detected:
            if (current_time - cam_data['last_save']).total_seconds() > self.save_delay:
                filename = f"{self.save_folder}/ALARM_{current_time.strftime('%H-
%M-%S')}_{cam_name.split()[0]}.jpg"
                cv2.imwrite(filename, annotated_frame)
                cam_data['last_save'] = current_time

            if self.tg_switch.get() == 1:
                if (current_time - cam_data['last_msg']).total_seconds() >
self.msg_delay:
                    threading.Thread(target=self.send_telegram_photo,
                                     args=(annotated_frame.copy(), cam_name)).start()
                    cam_data['last_msg'] = current_time

        # 4. Вивід на екран
        try:
            canvas = cam_data['canvas']
            w, h = canvas.winfo_width(), canvas.winfo_height()
            if w < 10 or h < 10: w, h = 400, 300

            frame_rgb = cv2.cvtColor(annotated_frame, cv2.COLOR_BGR2RGB)
            img = Image.fromarray(frame_rgb)

            img_w, img_h = img.size
            ratio = min(w / img_w, h / img_h)
            new_w, new_h = int(img_w * ratio), int(img_h * ratio)

            img = img.resize((new_w, new_h), Image.Resampling.LANCZOS)

            x_offset, y_offset = (w - new_w) // 2, (h - new_h) // 2
```

```
                cam_data['current_photo'] = ImageTk.PhotoImage(image=img)
                canvas.delete("all")
                canvas.create_image(x_offset, y_offset, image=cam_data['current_photo'],
anchor="nw")
            except Exception:
                pass

        if self.is_running:
            self.after(20, self.update_cameras_loop)

    def on_close(self):
        self.stop_all_cameras()
        self.destroy()


if __name__ == "__main__":
    app = SecurityApp()
    app.mainloop()
```

Додаток Д

СТАТТЯ З КОНФЕРЕНЦІЇ «ІНТЕЛЕКТУАЛЬНІ ТЕХНОЛОГІЇ У МІЖДИСЦИПЛІНАРНИХ ДОСЛІДЖЕННЯХ (ІТМД-2025)» 13 ЖОВТНЯ 2025

КОМП'ЮТЕРНА СИСТЕМА ВІДЕОСПОСТЕРЕЖЕННЯ НА ОСНОВІ БЕЗКОШТОВНИХ БІБЛІОТЕК PYTHON

Topic: Computer video surveillance system based on free Python libraries.

Purpose: The paper aims to develop an intelligent video surveillance system using exclusively free, open-source libraries. The goal is to create a system capable of human detection to significantly reduce the number of false alarms—a common problem in traditional motion-detection systems—without incurring the high costs of commercial AI solutions.

Methods: The system is built using Python. It utilizes the OpenCV library for capturing and processing the video stream in real-time. For human detection, the study employs the Haar Cascade classifier (specifically, haarcascade_fullbody.xml), which is a fast, CPU-based method integrated into OpenCV. Upon successful detection, the system uses the requests library to send an HTTP POST request to the Telegram Bot API, delivering an instant notification with a captured image (alert.jpg) to the user's device.

Results: A functional prototype of a cost-free, intelligent surveillance system was created. The system successfully identifies human presence in the video feed and ignores trivial movements (e.g., shadows, moving leaves), thus solving the problem of false positives. It operates locally on standard computer hardware (no GPU required) and sends reliable, instant alerts. This demonstrates the viability of open-source tools as a powerful alternative to expensive proprietary systems like Hikvision DeepinMind or Dahua WizMind.

Conclusions: The study confirms that combining Python, OpenCV, and the Telegram API allows for the creation of an effective, accessible, and intelligent security system. This approach eliminates financial barriers and provides a scalable foundation. Future work may involve integrating more robust (though resource-intensive) models like YOLO or Mediapipe for enhanced accuracy and the ability to recognize specific individuals or a wider range of objects.

Key words: video surveillance, Python, OpenCV, human recognition, Haar Cascades, Telegram API, intelligent system, false alarms.

Вступ

Системи відеоспостереження сьогодні відіграють критично важливу роль у забезпеченні безпеки в житлових, комерційних та громадських об'єктах. Вони еволюціонували від простих аналогових записуючих пристроїв до складних цифрових мереж. Проте, незважаючи на технологічний прогрес, переважна більшість встановлених систем досі покладається на базові методи детекції руху, що створює значні експлуатаційні проблеми.

Комерційні рішення з елементами штучного інтелекту, такі як Hikvision DeepinMind або Dahua WizMind, пропонують вирішення цих проблем через вбудовані алгоритми глибокого навчання. Вони здатні фільтрувати об'єкти та реагувати лише на людей чи транспорт. Однак їхня вартість є суттєвим бар'єром. Вона включає не лише дороге спеціалізоване обладнання (NVR з AI-чіпами), але й часто передбачає платні хмарні підписки для повноцінного функціонування сповіщень.

У зв'язку з цим надзвичайно актуальним стає завдання "демократизації" інтелектуального відеоспостереження. Розвиток та доступність потужних безкоштовних бібліотек, таких як OpenCV, нейромереж (YOLO, SSD) відкривають можливість створення аналогічних за функціональністю систем на базі звичайного комп'ютерного обладнання. Це дозволяє створити доступну, гнучку та потужну систему безпеки без фінансових бар'єрів та прив'язки до конкретного виробника.

Постановка проблеми

Класична проблема традиційних систем відеоспостереження полягає у методі детекції. Більшість камер використовують піксельний аналіз — система порівнює поточний кадр з попереднім і, якщо відсоток змінених пікселів перевищує заданий поріг, фіксує "рух". Цей підхід є обчислювально дешевим, але логічно недосконалим.

Для такої системи "рухом" є абсолютно все: коливання гілок дерев від вітру, дощ, снігопад, зміна освітлення від хмар, фари автомобілів, що проїжджають повз, і навіть комахи, що сідають на об'єктив. Як наслідок, користувач отримує сотні хибних сповіщень на день.

Це призводить до явища, відомого як "втома від сповіщень" (alert fatigue). Користувач спочатку намагається реагувати, потім знижує чутливість детекції (пропускаючи реальні загрози), і врешті-решт просто вимикає сповіщення. Система перетворюється з інструменту безпеки на "хлопчика, який кричав 'вовк'", створюючи лише ілюзію захисту. Окрім цього, сховища даних (жорсткі диски) миттєво заповнюються годинами непотрібного відео з хибними тривогами, що ускладнює подальший пошук реальних інцидентів.

Таким чином, ключова проблема — відсутність семантичного розуміння сцени. Системі бракує "інтелекту", щоб відрізнити значущу подію (появу людини) від нерелевантного фонового "шуму".

Мета роботи

Метою даної роботи є розробка програмного прототипу інтелектуальної системи відеоспостереження, здатної вирішити проблему хибних спрацьовувань. Для досягнення мети були поставлені наступні задачі:

- Проаналізувати існуючі безкоштовні інструменти комп'ютерного зору для задач детекції об'єктів.
- Обрати оптимальний стек технологій (бібліотеки, модель нейромережі), що забезпечує баланс між точністю та продуктивністю на стандартному CPU.
- Розробити архітектуру системи, що включає модулі захоплення відео, аналізу та миттєвого сповіщення.
- Реалізувати механізм фільтрації детекцій, що реагує виключно на об'єкт класу "людина".
- Інтегрувати систему сповіщень через доступний та безкоштовний канал зв'язку (Telegram API).

Аналіз методів та інструментів

Для побудови системи було проаналізовано декілька ключових технологій.

1. Обробка відео: Як основний інструмент для захоплення та обробки відеопотоку було обрано бібліотеку OpenCV (Open Source Computer Vision Library). Це де-факто стандарт у галузі комп'ютерного зору, що надає оптимізовані та кросплатформні інструменти для роботи з відео в реальному часі. Функція cv2.VideoCapture дозволяє уніфіковано працювати з різними джерелами: від USB-вебкамер до RTSP-потоків IP-камер.

2. Алгоритми детекції:

- Каскади Хаара (Viola-Jones): Це класичний метод, інтегрований в OpenCV. Він надзвичайно швидкий на CPU, але є "крихким" — добре працює лише на об'єктах у певному ракурсі (наприклад, обличчя анфас) і чутливий до освітлення. Для детекції тіла людини (haarcascade_fullbody.xml) він дає занадто багато хибних спрацьовувань та пропусків, що робить його недостатньо надійним.
- HOG (Histogram of Oriented Gradients): Інший класичний підхід, часто використовуваний з SVM. Він точніший за каскади Хаара, але значно повільніший і також не досягає рівня сучасних нейромереж.
- Нейромережі (YOLO, SSD): Це сучасний підхід, заснований на глибокому навчанні. Моделі "дивляться" на кадр один раз (You Only Look Once) і одразу видають координати та класи об'єктів. Вони навчені на величезних наборах даних (як COCO) і здатні розпізнавати десятки класів з високою точністю. Для моєї задачі було обрано модель YOLOv3-Tiny. Це "полегшена" версія, що є оптимальним компромісом: вона достатньо точна для надійної ідентифікації людини, але водночас достатньо швидка для роботи на CPU в режимі реального часу (5-10 FPS).

3. Система сповіщень:

- Email (SMTP): Безкоштовний, але не забезпечує миттєвої доставки. Листи можуть потрапляти у спам або приходити з затримкою, що є неприпустимим для системи безпеки.
- SMS: Потребує купівлі послуг у платного SMS-шлюзу.
- Telegram Bot API: Обрано як ідеальний варіант. Він безкоштовний, надзвичайно надійний, надає миттєві push-сповіщення на мобільні пристрої та дозволяє легко надсилати медіа-вкладення (фотографії порушника).

Архітектура та реалізація системи

Запропонована система є модульною і складається з чотирьох логічних блоків, що працюють у безперервному циклі.

Модуль 1: Захоплення відео. Цей модуль відповідає за підключення до джерела відео (веб-камери або IP-камери) за допомогою cv2.VideoCapture. Він у безкінечному циклі зчитує кадри та передає їх на обробку. Для уникнення "зависання" основного потоку при втраті зв'язку з IP-камерою, цей процес доцільно виносити в окремий потік (thread).

Модуль 2: Детекція (Обробка нейромережею). Це "серце" системи. Отриманий кадр проходить кілька етапів:

1. Передобробка: Кадр змінює розмір до того, який очікує нейромережа (наприклад, 416x416 пікселів для YOLOv3-Tiny).
2. Створення "Blob": Кадр перетворюється на 4-вимірний "blob" — спеціальний числовий масив, який є вхідним форматом для моделі (cv2.dnn.blobFromImage).
3. Пряме поширення: Blob подається на вхід завантаженої моделі YOLO (net.forward()). Мережа повертає список усіх виявлених об'єктів, їхні координати та рівень впевненості для кожного класу.

Модуль 3: Пост-обробка та фільтрація. "Сирі" результати з нейромережі потребують очищення:

1. Поріг впевненості (Confidence Thresholding): Відкидаються всі детекції, впевненість у яких нижча за встановлений поріг (наприклад, 50%). Це відсіює слабкі та ненадійні розпізнавання.
2. Придушення не-максимумів (Non-Maximum Suppression, NMS): Нейромережа часто знаходить декілька рамок для одного об'єкта. NMS — це алгоритм, який аналізує рамки, що перекриваються, і залишає лише одну, найбільш "впевнену". Це запобігає "рої" рамок навколо однієї людини.
3. Фільтрація класів: Після очищення система перевіряє клас кожного виявленого об'єкта. Вона ігнорує всі об'єкти (наприклад, "кіт", "собака", "автомобіль") і спрацьовує тільки якщо в кадрі присутній об'єкт класу "person" (людина).

Модуль 4: Сповіщення та реагування. Якщо після всіх фільтрів у кадрі виявлено хоча б одну людину, система переходить у режим тривоги.

1. Запис кадру: Поточний кадр (вже з намальованими рамками) зберігається як JPEG-файл.
2. "Перезарядка" (Cooldown): Щоб не надсилати 100 сповіщень, якщо людина стоїть у кадрі 10 секунд, введено "час перезарядки" (наприклад, 30 секунд). Нове сповіщення не буде надіслано, доки цей час не мине.
3. Відправка в Telegram: За допомогою бібліотеки requests система робить POST-запит до Telegram Bot API, надсилаючи збережений кадр та текстове повідомлення (наприклад, "Увага! Виявлено людину!").

Очікувані результати та обговорення

Результатом впровадження такої архітектури є ефективна, повністю безкоштовна та інтелектуальна система відеоспостереження. Тестування подібних прототипів демонструє, що навіть на відносно слабкому CPU (наприклад, Intel Core i5 5-7-го покоління) система здатна обробляти відеопотік з IP-камери з роздільною здатністю 720p зі швидкістю 5-8 кадрів на секунду.

Цього цілком достатньо для завдань безпеки, оскільки детекція відбувається майже миттєво. Головне — система успішно вирішує поставлену проблему: вона повністю ігнорує рух гілок, зміни освітлення та рух домашніх тварин, зводячи кількість хибних спрацьовувань практично до нуля. Користувач отримує сповіщення лише у значущих випадках.

Обмеженням обраної моделі (YOLOv3-Tiny) є її відносно низька точність при детекції дуже малих або частково перекритих об'єктів на великій відстані. Однак для моніторингу приміщень, входів або периметру на середній дистанції її точності більш ніж достатньо.

Висновки

Проведене дослідження та розробка прототипу доводять, що створення інтелектуальної системи відеоспостереження, яка не поступається за логікою роботи дорогим комерційним аналогам, є абсолютно реальним з використанням безкоштовних бібліотек Python та моделей з відкритим кодом.

Поєднання OpenCV для обробки відео, нейромережі YOLO для точної детекції об'єктів та Telegram API для миттєвих сповіщень створює потужну, гнучку та доступну платформу. Розроблений підхід повністю вирішує проблему "втоми від сповіщень", фільтруючи події та реагуючи лише на реальну присутність людини.

Напрямками подальших досліджень є:

- Оптимізація системи для роботи на мікрокомп'ютерах (Raspberry Pi 4/5, NVIDIA Jetson) для створення повністю автономних AI-камер.
- Інтеграція більш сучасних та точних моделей (наприклад, YOLOv8) з використанням GPU-прискорення.
- Додавання механізмів трекінгу (відстеження) об'єкта між камерами.
- Реалізація модуля розпізнавання облич для створення "білих списків" (члени родини, персонал), щоб система не реагувала на "своїх".

СПИСОК ЛІТЕРАТУРИ

- Bradski G., Kaehler A. *Learning OpenCV 4: Computer Vision with Python*. Sebastopol: O'Reilly Media, 2020. 946 p.
- Шелест М. В., Красін П. А. Методи виявлення та розпізнавання рухомих об’єктів у системах відеоспостереження. *Вісник НТУУ "КПІ". Серія "Інформатика, управління та обчислювальна техніка"*. 2019. № 71. С. 45–53.
- OpenCV (Open Source Computer Vision Library): Офіційна документація. URL: https://docs.opencv.org/
- Telegram Bot API: Офіційна документація. URL: https://core.telegram.org/bots/api
- Viola P., Jones M. J. Robust Real-Time Face Detection. *International Journal of Computer Vision*. 2004. Vol. 57(2). P. 137–154.

Макогон Дмитро Олександрович – студент, ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ В. Н. КАРАЗІНА; м. Харків, майдан Свободи, 4; 61022; makohon2021ku12@student.karazin.ua; ORCID*: 0009-0001-9383-9571.

Наукові інтереси: Розробка "розумних" систем