МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____ Олена НЕЧИПОРУК
«_____»_____2025р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ВИЩОЇ ОСВІТИ
ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»

Тема: «Програмний засіб обробки сигналів у реальному часі в системах інтелектуального відеоспостереження»

Виконавець: Олександр ПОБУТА

Керівник: Олена НЕЧИПОРУК

Нормоконтролер: Євгеній ТУПОТА

Київ 2025

«ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»

Факультет комп'ютерних наук та технологій
Кафедра інтелектуальних кібернетичних систем
Спеціальність 126 «Інформаційні системи та технології»
(шифр, найменування)
Освітньо-професійна програма « Інформаційні системи та технології»
Форма навчання денна

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Олена НЕЧИПОРУК

«_____»_____2025 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи
Побути Олександра Миколайовича
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема кваліфікаційного проєкту «Програмний засіб обробки сигналів у реальному часі в системах інтелектуального відеоспостереження»
затверджена наказом ректора від «28» серпня 2025 р. № 1762/ст
2. Термін виконання роботи (проекту): з 29.09.2025 по 31.12.2025
3. Вихідні дані до роботи (проекту): Програмний модуль, відеоаналітика.
4. Зміст пояснювальної записки:
1) Аналіз предметної області.
2) Структура програмного засобу.
3) Програмна реалізація.
5. Перелік обов'язкового графічного (ілюстративного) матеріалу:
1) Програмний модуль (Схема алгоритму).
2) Структура програмного засобу.
3) Верифікація моделі.
4) *ER* діаграма бази даних.

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1	Ознайомитись з постановкою задачі кваліфікаційної роботи	01.11.2025 – 10.11.2025	
2	Проаналізувати джерела за темою кваліфікаційної роботи та зробити аналіз аналогічних рішень	11.11.2025 – 16.11.2025	
3	Написати перший розділ	17.11.2025 – 20.11.2025	
4	Провести аналіз вимог користувача та спроектувати компоненти програмного засобу	21.11.2025 – 28.11.2025	
5	Написати другий розділ	29.11.2025 – 01.12.2025	
6	Розробити базу даних для програмного засобу	02.12.2025 – 08.12.2025	
7	Написати третій розділ	09.12.2025 – 10.12.2025	
8	Оформити пояснювальну записку	11.12.2025 – 19.12.2025	
9	Підготувати графічний матеріал	20.12.2025 – 24.12.2025	
10	Захистити кваліфікаційну роботу	26.12.2025	

7. Дата видачі завдання: «29» «вересня» 2025р.

Керівник кваліфікаційного проєкту _____ Олена НЕЧИПОРУК
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Олександр ПОБУТА
(підпис здобувача вищої освіти) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Програмний засіб обробки сигналів у реальному часі в системах інтелектуального відеоспостереження»: 87 с., 10 рис., 16 табл., 22 літературних джерел.

Ключові слова: Інтелектуальна система відеоспостереження, комп'ютерний зір, детекція об'єктів, відеоаналітика, *python*, *yolo*, *opencv*, *sqlite*, трекінг, реєстрація подій, безпека.

Об'єктом дослідження є процес відеоспостереження та моніторингу об'єктів у режимі реального часу.

Предметом дослідження є методи обробки сигналів у реальному часі.

Мета – розробити програмний засіб обробки сигналів у реальному часі в системах інтелектуального відеоспостереження.

Наукова новизна полягає в удосконаленні підходу до реєстрації подій у системах відеоспостереження за рахунок поєднання нейронної мережі для детекції об'єктів, алгоритмів трекінгу та аналізу входу в контрольовану зону, що забезпечує підвищення точності фіксації подій у реальному часі.

ЗМІСТ

ВСТУП 6
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 8
1.1. Сучасний стан та тенденції розвитку систем відеоспостереження 8
1.2. Архітектура та принципи функціонування інтелектуальних систем відеоспостереження 9
1.3. Технологічні аспекти та сучасні методи обробки відеосигналів у реальному часі 11
1.4. Огляд існуючих систем та програмних рішень у сфері інтелектуального відеоспостереження 25
1.5. Висновки по розділу 38
РОЗДІЛ 2 СТРУКТУРА ПРОГРАМНОГО ЗАСОБУ 40
2.1. Аналіз вимог до програмного засобу 40
2.2. Вибір інструментальних засобів та технологій розробки 43
2.3. Алгоритмізація моделі системи і її машинна реалізація 50
2.4. Проєктування бази даних програмного засобу 54
2.5. Проєктування користувацького інтерфейсу програмного засобу 67
2.6. Висновки по розділу 70
РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ 72
3.1. Загальна структура програмної системи.. 72
3.2. Використані технології та середовище розробки. 73
3.3. Архітектура та принципи взаємодії модулів 75
3.4. Реалізація основних компонентів 76
3.5. Тестування системи 79
3.6. Результати тестування 82
3.7. Висновки по розділу 82
ВИСНОВКИ 84
СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ 86
ДОДАТОК А 88

ВСТУП

Сучасні системи відеоспостереження перестали бути лише пасивними засобами фіксації зображення. З розвитком обчислювальних технологій та програмних методів обробки даних вони дедалі частіше виконують роль інтелектуальних систем, здатних аналізувати відеопотік у режимі реального часу та автоматично виявляти події, що потребують уваги[15; 16]. Такий підхід дозволяє значно зменшити навантаження на оператора та підвищити загальну ефективність контролю за об’єктами спостереження.

Особливо актуальним це є для об’єктів, де постійний ручний контроль відеопотоку є складним або неможливим з практичної точки зору. Велика кількість камер, тривалі періоди спостереження та монотонність процесу призводять до зниження уваги оператора та ймовірності пропуску важливих подій. Саме тому зростає інтерес до програмних засобів, які здатні автоматизувати процес аналізу відеоданих та своєчасно повідомляти про зафіксовані події.

У зв’язку з цим важливим є дослідження можливостей створення простих та доступних програмних рішень, які не потребують дорогого апаратного забезпечення або складної інфраструктури. Використання сучасних бібліотек комп’ютерного зору та нейромережевих моделей дозволяє реалізувати базову відеоаналітику навіть на звичайних персональних комп’ютерах. Саме такий підхід дає змогу поєднати практичну корисність системи з її простотою у використанні та налаштуванні.

Актуальність теми полягає в тому, що інтелектуальні системи відеоспостереження сьогодні широко застосовуються у різних сферах діяльності людини: у побуті, бізнесі, транспортній інфраструктурі, на промислових об’єктах та в система.

Інтеграція алгоритмів обробки відеосигналів та елементарної аналітики дозволяє значно підвищити функціональні можливості таких систем без залучення дорогого обладнання або складних серверних рішень.

Це відкриває можливість створення локальних, бюджетних програмних засобів, які здатні ефективно вирішувати реальні прикладні задачі користувачів.

Предметом дослідження є методи обробки сигналів у реальному часі.

Мета – розробити програмний засіб обробки сигналів у реальному часі в системах інтелектуального відеоспостереження. Наукова новизна полягає в удосконаленні підходу до реєстрації подій у системах відеоспостереження за рахунок поєднання нейронної мережі для детекції об’єктів, алгоритмів трекінгу та аналізу входу в контрольовану зону, що забезпечує підвищення точності фіксації подій у реальному часі.

Завдання на кваліфікаційну роботу

- Проаналізувати сучасні системи відеоспостереження та методи автоматичного аналізу відеоданих.
- Дослідити існуючі підходи та алгоритми комп’ютерного зору для детекції та відстеження об’єктів.
- Розробити архітектуру програмного модуля інтелектуальної системи відеоспостереження.
- Реалізувати програмний модуль обробки відеопотоку з використанням методів машинного навчання.
- Забезпечити автоматичну реєстрацію подій та збереження результатів аналізу в базі даних.
- Провести тестування та оцінку коректності роботи розробленої системи.

У даній роботі створено програмний засіб, який працює локально, не потребує складного налаштування та орієнтований на використання однією людиною. Такий підхід є доцільним для невеликих офісів, приватних домоволодінь, майстерень, складів або інших приміщень, де відсутня необхідність у масштабних та дорогих системах відеонагляду, але при цьому важливо забезпечити базовий контроль і аналіз подій.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Сучасний стан та тенденції розвитку систем відеоспостереження

Системи відеоспостереження за останні роки суттєво змінилися під впливом розвитку цифрових технологій, зростання рівня урбанізації та підвищення вимог до безпеки. Традиційні рішення на базі аналогових камер, які забезпечували лише запис відео для подальшого перегляду, уже не відповідають сучасним потребам. Сьогодні важливо не просто фіксувати події, а отримувати оперативну інформацію та аналітичні висновки в режимі реального часу. Це зумовлює необхідність переходу до інтелектуальних систем, здатних самостійно аналізувати відеодані та зменшувати навантаження на оператора.

Перехід до сучасного покоління відеоспостереження став можливим завдяки появі цифрових камер високої роздільної здатності, розвитку мережевих технологій та доступності обчислювальних ресурсів. Сучасні системи активно використовують інструменти комп’ютерного зору, машинного навчання та алгоритми автоматичного аналізу відео. Це дає змогу виявляти та класифікувати об’єкти, розпізнавати дії, аналізувати поведінку людей і оперативно повідомляти про підозрілу активність. Використання таких підходів дозволяє значно підвищити точність та швидкість реагування систем відеоспостереження у порівнянні з традиційними методами.

Однією з ключових тенденцій є перехід від “реактивних” до “проактивних” систем. Якщо раніше відеозаписи здебільшого слугували доказовою базою після інциденту, то сьогодні відеоаналітика дозволяє попереджати небажані ситуації. Системи здатні автоматично виявляти потенційно небезпечні сценарії та формувати попередження ще до настання критичних подій.

Крім того, сучасні системи інтегруються з охоронними комплексами, контролем доступу, транспортною інфраструктурою та елементами “розумного міста”, утворюючи цілісне інформаційне середовище.

Важливим напрямом розвитку є зміна підходів до зберігання відеоданих. Якщо раніше домінували локальні архіви, то зараз багато рішень переходять до хмарних технологій. Це спрощує масштабування, дозволяє отримувати доступ до архівів з будь-якої точки та інтегрувати сервіси штучного інтелекту. Разом із тим, такий підхід висуває додаткові вимоги до надійності передачі даних та стабільності мережевої інфраструктури. Водночас зростає актуальність питань інформаційної безпеки та захисту персональних даних, оскільки обсяг відео зі чутливою інформацією постійно збільшується.

Разом із функціями безпеки відеоаналітика все частіше використовується в інших сферах: роздрібній торгівлі, виробництві, транспорті, медицині, логістиці. Інтелектуальні системи допомагають оптимізувати бізнес-процеси, аналізувати поведінку відвідувачів, керувати потоками людей і транспорту, контролювати якість сервісу та дотримання вимог безпеки. Таким чином, відеоспостереження стає важливим інструментом підтримки управлінських рішень у різних галузях. Тобто відбувається трансформація від вузькопрофільних охоронних систем до багатофункціональних універсальних платформ.

Узагальнюючи, сучасні системи відеоспостереження дедалі більше основуються на штучному інтелекті та автоматизації аналізу даних. Це формує запит на нові програмні рішення, здатні працювати в режимі реального часу, адаптуватися до умов застосування та забезпечувати високу ефективність виявлення подій.

1.2. Архітектура та принципи функціонування інтелектуальних систем відеоспостереження

Інтелектуальні системи відеоспостереження (ІСВС) поєднують програмні та апаратні засоби, призначені для автоматизованого збору, аналізу та інтерпретації відеоданих. Їх основна мета – підвищення рівня безпеки, підтримка прийняття рішень та оптимізація різних процесів.

На відміну від традиційних систем, що зосереджувалися переважно на записі та зберіганні відео, ІСВС включають аналітичні модулі, здатні обробляти відеопотік у реальному часі та формувати корисні висновки для користувача.

Типова архітектура ІСВС складається з кількох взаємопов'язаних компонентів, які забезпечують повний цикл опрацювання відеосигналу – від моменту його отримання до формування аналітичних результатів:

Підсистема збору даних – відеокамери, що можуть відрізнятися за роздільною здатністю, чутливістю, мережею передачі даних та функціональністю. Сучасні *IP*-камери нерідко мають вбудовані алгоритми попередньої обробки або штучного інтелекту, що частково зменшує навантаження на сервер.

Транспортна інфраструктура забезпечує передавання відеоданих між компонентами системи. У залежності від масштабу та вимог застосовують дротові (*Ethernet*, оптика) або бездротові канали зв'язку. Ключовим є забезпечення достатньої пропускної здатності та мінімальних затримок при передаванні потокового відео.

Підсистема зберігання відео може бути реалізована на локальних (*DVR*, *NVR*), мережевих (*NAS*) або хмарних платформах. Вибір залежить від вимог до обсягу архівів, швидкості доступу, безпеки та можливостей масштабування.

Аналітичний модуль – головна відмінність ІСВС від класичних рішень. Він відповідає за розпізнавання об'єктів, відстеження їх руху, класифікацію подій, виявлення нетипової поведінки та формування сповіщень. Аналітика може виконуватися на камері (*edge*-комп'ютинг), на локальному сервері або у хмарі.

Підсистема керування та інтерфейс користувача забезпечують взаємодію оператора з системою: перегляд відео, доступ до архівів, налаштування аналітики, перегляд журналів та звітів.

Функціонування ІСВС базується на послідовному процесі обробки: захоплення відео, передавання, аналіз, інтерпретація,збереження, реагування. Недоліки:

Ступінь автоматизації може відрізнятися залежно від завдань та технічних можливостей системи.

Однією з провідних тенденцій розвитку є перехід від централізованих до розподілених і гібридних моделей обробки даних.

Використання *edge*-комп’ютингу дозволяє виконувати частину аналітики безпосередньо на пристроях збору інформації, що зменшує затримки та навантаження на мережу. Це особливо важливо для систем, де потрібна швидка реакція – наприклад, у транспортній інфраструктурі, промисловій безпеці чи аеропортах.

Ще одна важлива риса ІСВС – їх модульність і масштабованість. Системи можуть ефективно працювати як у вигляді локального рішення з кількома камерами, так і як розподілена мережа з тисячами вузлів. Це робить їх придатними для використання як у приватному секторі, так і в державних структурах.

Окремої уваги заслуговує питання навантаження на оператора системи відеоспостереження. У традиційних системах без автоматизованого аналізу відео людина змушена постійно переглядати відеопотік або архів записів, що призводить до втоми, зниження концентрації та збільшення ймовірності пропуску важливих подій. Саме цей фактор став однією з основних причин переходу від пасивних систем відеофіксації до інтелектуальних систем відеоспостереження.

Інтелектуальні системи відеоспостереження дозволяють автоматизувати процес аналізу відеоданих шляхом використання алгоритмів комп’ютерного зору та машинного навчання. Такі системи здатні самостійно виявляти об’єкти, відстежувати їх рух, аналізувати поведінку та формувати події без безпосередньої участі людини. У результаті оператор отримує не суцільний відеопотік, а структуровану інформацію у вигляді повідомлень про конкретні події.

1.3. Технологічні аспекти та сучасні методи обробки відеосигналів у реальному часі

Цифрова обробка відеосигналів є сукупністю методів і процедур, що забезпечують перетворення, аналіз та підготовку відеоданих для подальшої інтерпретації та автоматизованої аналітики.

В основі будь-якої інтелектуальної системи відеоспостереження лежить процес трансформації сирого відеопотоку у структуровану інформацію, придатну для аналізу алгоритмами комп’ютерного зору та штучного інтелекту. Цифровий відеосигнал можна розглядати як тривимірну функцію інтенсивності пікселів, що залежить від

просторових координат та часу. Формально відеопотік задають як послідовність дискретних кадрів.

Кожен кадр описується матрицею розміром $M \times N$, де M – висота зображення, N – ширина. Піксель з координатами (x,y) має інтенсивність, що може бути скалярною (для монохромних зображень) або векторною (для кольорових).

Шум може бути викликаний недоліками сенсора, низьким освітленням, перешкодами перед камерою або компресією при передаванні даних.

Окрім наявності шумів, важливим аспектом цифрової обробки відеосигналів є необхідність приведення відеоданих до формату, придатного для подальшого автоматизованого аналізу. Це включає нормалізацію яскравості, корекцію кольору, масштабування зображень та перетворення колірних просторів. Такі операції дозволяють зменшити вплив зовнішніх факторів та підвищити стабільність роботи алгоритмів комп'ютерного зору.

В процесі попередньої обробки відеосигналу також застосовуються методи фільтрації, спрямовані на зменшення шуму та підвищення інформативності зображення. До них належать згладжувальні, медіанні та гаусівські фільтри, які дозволяють покращити якість кадрів без значної втрати корисних деталей.

Це є особливо важливим для систем, що працюють у складних умовах освітлення або з використанням недорогих відеокамер.

Наступним етапом цифрової обробки відеосигналів є виділення інформативних ознак, необхідних для задач детекції та аналізу об'єктів. У сучасних системах цей етап здебільшого реалізується за допомогою нейронних мереж, які автоматично формують багаторівневі ознакові представлення зображень. Таким чином, ручне проєктування ознак поступово замінюється навчанням моделей на великих обсягах даних. Також важливим аспектом цифрової обробки відеосигналів є врахування обмежень реального часу. Алгоритми обробки повинні бути достатньо швидкими, щоб забезпечувати аналіз кожного кадру без суттєвих затримок, що особливо критично для систем відеоспостереження з функціями реагування на події. Це накладає вимоги на оптимізацію обчислень та ефективне використання апаратних ресурсів.

Крім того, цифрова обробка відеосигналів тісно пов'язана з компромісом між якістю зображення та обчислювальною складністю.

Надмірна фільтрація або надто агресивні перетворення можуть призводити до втрати корисної інформації, що негативно впливає на точність детекції об’єктів. Тому вибір методів попередньої обробки має здійснюватися з урахуванням специфіки задачі та умов експлуатації системи.

В умовах інтелектуального відеоспостереження цифрова обробка відеосигналів слугує проміжною ланкою між апаратним рівнем захоплення даних та високорівневими алгоритмами аналізу. Саме від якості реалізації цього етапу значною мірою залежить стабільність роботи нейронних мереж, точність виявлення об’єктів і надійність реєстрації подій у системі.

Таблиця 1.1

Оцінювання якості відеопотоку

Параметр	Позначення	Опис
Роздільна здатність	(*M x N*)	Кількість пікселів у кадрі
Частота кадрів	*FPS* (*frames per second*)	Кількість кадрів за секунду
Глибина кольору	*bit depth*	Кількість біт для зберігання кольору
Бітрейт	*bitrate*	Обсяг даних відео за секунду
Динамічний діапазон	*HDR*	Різниця між найтемнішим та найсвітлішим елементом

Для систем відеоспостереження типові роздільності: 1280×720 (*HD*), 1920×1080 (*Full HD*), 2560×1440 (2*K*) та 3840×2160 (4*K*). Частота кадрів залежить від задачі: 8-12 *FPS* достатньо для охорони периметра, 25-30 *FPS* – для детекції людей, 50-120 *FPS* використовується на транспортних вузлах та у швидкісній аналітиці. **Етапи цифрової обробки відеосигналу**. Обробку відеоданих здійснюють у кілька послідовних етапів, які забезпечують покращення якості та підготовку кадрів до аналізу алгоритмами штучного інтелекту. Типова схема включає:

– оцифрування та декодування відеопотоку;

– попередню обробку сигналу (*preprocessing*);

– виділення ознак сцени;

– аналіз та інтерпретацію даних.

Попередню обробку застосовують з метою мінімізації шумів, компенсації освітлення, нормалізації яскравості та покращення контрастності.

Попередня фільтрація шумів

Фільтрація відеосигналу спрямована на усунення шумів $Nt(x,y)$, представлених у формулі загальної моделі кадру. Найчастіше застосовуються фільтри згладжування.

Гаусів фільтр реалізує згладжування шляхом згортки з ядром Гауса

Перевага – ефективне усунення високочастотних шумів; недолік – розмиття контурів. Медіанний фільтр замінює значення пікселя медіаною сусідніх, ефективний проти імпульсного шуму.

Нормалізація кадру. Для забезпечення коректної роботи алгоритмів комп'ютерного зору необхідна нормалізація інтенсивності та контрастності кадру.

Для аналітики найбільш уживаними є моделі:

– *RGB* – стандартна модель сенсорів камер;

– *HSV* – стійка до змін освітлення;

– *YCrCb* – розділяє яскравість і колір, ефективна для фону і людської шкіри.

У більшості алгоритмів детекції спочатку застосовується перетворення *RGB*, *YCrCb* або *HSV* для підвищення стійкості.

Методи виявлення руху та змін у кадрі. Виявлення руху є базовим етапом відеоаналітики, що дає змогу виокремити області кадру, у яких відбулися зміни, пов'язані з появою або переміщенням об'єктів. Результатом аналізу є маска руху, що відображає пікселі, інтенсивність яких суттєво змінилася між послідовними кадрами.

Цей етап є критично важливим для подальшої детекції та класифікації об'єктів, оскільки зменшує обсяг інформації, що підлягає обробці алгоритмами штучного інтелекту, і підвищує ефективність системи в режимі реального часу.

Існує три основні підходи до виявлення руху:

– методи, засновані на різниці кадрів;

– методи побудови моделі фону (*background subtraction*);

– оптичний потік (*optical flow*).

Метод різниці кадрів. Простий підхід, який передбачає порівняння двох або більше послідовних кадрів. Рух визначають за пороговими змінами інтенсивності пікселів між кадрами, що дозволяє швидко виявляти динамічні області зображення. Такий підхід часто застосовується як базовий етап попередньої обробки або у простих системах відеоспостереження без складної аналітики. Переваги методу полягають у простоті реалізації та високій швидкодії, що робить його придатним для слабких обчислювальних пристроїв та вбудованих систем. Водночас недоліками є висока чутливість до шумів, різких змін освітлення та вібрацій камери, а також потреба у стабільному відеопотоці, де камера повинна залишатися статичною.

Переваги методу – простота реалізації та висока швидкодія, що робить його придатним для слабких обчислювальних пристроїв. Недоліки – чутливість до шумів, зміни освітлення та потреба у стабільному відео (камера повинна бути статичною).

Методи віднімання фону (*Background Subtraction*): ці методи ґрунтуються на побудові та оновленні моделі фону сцени.

Пікселі, що суттєво відрізняються від моделі фону, розглядають як об’єкти переднього плану (*foreground*). Фон можна формувати як:

– статичний середній кадр;

– адаптивну статистичну модель;

– модель на основі параметричних розподілів.

Модель змішування гауссових розподілів (*MOG*, *MOG2*) – Одним із найпоширеніших алгоритмів є *MOG2* (*Mixture of Gaussians v.2*), що представляє інтенсивність кожного пікселя як суміш *KKK* гауссових розподілів.

Система визначає, чи належить нове значення пікселя до фону, порівнюючи його зі статистичною моделлю. *MOG2* забезпечує адаптацію до змін фону, наприклад, коливань освітлення чи руху дерев.

Алгоритм *KNN* (*K-Nearest Neighbors*)

Метод класифікації пікселів за аналогією з класифікаційним алгоритмом *kkk*-найближчих сусідів. Зберігає останні *NNN* значень пікселя та класифікує нове спостереження як фон чи об’єкт за кількістю близьких значень.

Перевага – стійкість до шумів, недолік – більші витрати пам’яті порівняно з *MOG2*.

Таблиця 1.2

Порівняння систем відеоспостереження

Критерій	*MOG2*	*KNN*	Різниця кадрів
Адаптація до змін освітлення	Так	Частково	Ні
Швидкодія	Висока	Середня	Дуже висока
Якість виділення об’єктів	Висока	Висока	Низька
Потреба в пам’яті	Середня	Висока	Низька

У системах відеоспостереження *MOG*2 є де-факто стандартом попереднього виділення руху, оскільки поєднує відносну простоту реалізації та достатню ефективність для роботи в реальному часі. Алгоритм дозволяє адаптуватися до поступових змін сцени, таких як коливання освітлення або поява статичних об’єктів, що з часом переходять у фон. Завдяки цьому *MOG*2 широко використовується як початковий етап аналізу відеопотоку у багатьох прикладних системах.

Методи на основі оптичного потоку. Оптичний потік – це векторне поле, яке описує швидкість і напрям переміщення пікселів між сусідніми кадрами відеопослідовності. На відміну від методів фонового моделювання, даний підхід ґрунтується на аналізі локальних змін інтенсивності зображення та дозволяє більш точно визначати характер руху в сцені. Це робить оптичний потік корисним для задач, де важливо не лише зафіксувати факт руху, а й оцінити його напрямок та швидкість. Найбільш відомими алгоритмами є:

Лукас-Канаде (*Lucas–Kanade*) обчислює оптичний потік для малих блоків, припускаючи сталість руху в околі.

Фарнебек (*Farneback*) апроксимує інтенсивність поліномом другого порядку для оцінки руху.

Метод забезпечує високу точність для дрібних та повільних об’єктів, але є обчислювально витратним.

Таблиця 1.3

Вибір алгоритму від умов середовища

Сценарій	Рекомендований метод
Стаціонарна камера, фон змінюється	*MOG2* або *KNN*
Камера рухається чи вібрує	Оптичний потік
Обмежені ресурси обчислення	Різниця кадрів
Висока точність і деталізація	Поєднання *MOG2* + *Optical Flow*

У сучасних *ІВС* часто використовується комбінований підхід, коли *MOG2* застосовується для виділення об’єктів, а оптичний потік – для уточнення траєкторій руху.

Сучасні моделі та алгоритми комп’ютерного зору для детекції об’єктів

Детекція об’єктів є однією з ключових задач у системах відеоспостереження, оскільки забезпечує автоматичне виявлення та локалізацію об’єктів на відеокадрі з подальшою класифікацією за типом.

Еволюція алгоритмів детекції пройшла шлях від традиційних методів комп’ютерного зору до сучасних моделей глибокого навчання, які відповідають вимогам аналізу відеопотоків у режимі реального часу.

Класифікація методів детекції об’єктів – у сучасній науковій літературі методи детекції умовно поділяють на дві групи:

Класичні (традиційні) методи комп’ютерного зору – базуються на описі об’єктів за допомогою інженерних ознак (*feature engineering*). Методи, засновані на глибинному навчанні (*Deep Learning*) використовують згорткові нейронні мережі (*CNN*) для автоматичного витягнення ознак та класифікації об’єктів.

Ці два підходи мають різні можливості, обчислювальні вимоги та точність.

До появи нейронних мереж детекція здійснювалася шляхом ручного виділення ознак і подальшої класифікації. Найбільш відомі методи:

Метод Виоли–Джонса (*Haar Cascades*) – метод базується на використанні *Haar*-подібних ознак, що обчислюються як різниця середніх значень інтенсивності у прямокутних областях зображення. Для прискорення використовується інтегральне зображення, що дає змогу обчислювати ознаки за 0(1) час.

Алгоритм формує «каскад класифікаторів» – послідовність слабких класифікаторів, які поетапно відсіюють нецільові області. Технологія забезпечила прорив у детекції облич у 2000-х роках.

Переваги:

– висока швидкість на *CPU*, можливість роботи в реальному часі.

Недоліки:

– низька точність у складних умовах, чутливість до ракурсу та освітлення.

HOG (*Histogram of Oriented Gradients*) – метод гістограм градієнтів Видала та Даллала передбачає обчислення локальних гістограм напрямів градієнтів, що описують структуру контурів об’єкта. Часто поєднується з *SVM*-класифікатором.

Переваги:

– добре працює для детекції людей (*pedestrian detection*).

Недоліки:

– не стійкий до значних деформацій об’єкта.

Таблиця 1.4

Порівняння методів відеоспостереження

Метод	Повна назва	Особливості
SIFT	*Scale-Invariant and Feature Transform*	Інваріантний до масштабу і повороту
SURF	*Speeded-Up and Robust Features*	Швидший аналог *SIFT*
ORB	*Oriented FAST and Rotated BRIEF*	Придатний для реального часу, *open-source*

Ці методи використовуються для пошуку ключових точок та відстеження об’єктів, але не забезпечують повноцінної детекції об’єктів.

Методи детекції, засновані на глибинному навчанні: поява згорткових нейронних мереж (*CNN*) революціонізувала сферу комп'ютерного зору.

Замість ручного конструювання ознак *CNN* навчається виділяти їх автоматично. Це забезпечило суттєве підвищення точності, масштабованості та адаптивності систем відеоаналізу.

Алгоритми детекції поділяють на два класи:

- двостадійні (*two-stage detectors*) спочатку генерують регіони-кандидати, потім класифікують їх;
- одностадійні (*single-stage detectors*) виконують детекцію та класифікацію за один крок.

Двостадійні моделі: найвідомішим представником є *Faster R-CNN*, у якій перший модуль (*Region Proposal Network*) пропонує області, ймовірно, відповідні об'єктам, а другий здійснює їх класифікацію та регресію координат.

Ці моделі демонструють високу точність, проте характеризуються відносно низькою швидкістю.

Переваги:

- найкраща якість детекції у складних сценаріях.

Недоліки:

- не оптимальні для відеопотоків у реальному часі, особливо без *GPU*.

Одностадійні моделі: найбільш поширені для систем відеоспостереження моделі: *SSD*, *YOLO*, *RetinaNet*.

SSD (*Single Shot MultiBox Detector*):

- Виконує детекцію та регресію координат на кількох масштабах, що забезпечує ефективність у виявленні об'єктів різного розміру.
- Працює швидше, ніж *Faster R-CNN*, але трохи поступається точністю.

YOLO (*You Only Look Once*):

- представляє детекцію як задачу регресії: модель ділить зображення на сітку та для кожної осередки формує набір передбачень (*bounding boxes* + клас);
- забезпечує високу швидкодію, що робить її стандартом де-факто для систем відеоаналітики.

Останні покоління моделі – *YOLOv5*, *YOLOv7*, *YOLOv8* – забезпечують оптимальний баланс між точністю та обчислювальною складністю.

За даними незалежних тестів *Roboflow*(2023), *YOLOv8* показує на 14–18% кращу точність порівняно *з YOLOv5* на багатьох наборах даних.

Трансформерні моделі *Vision Transformer* (*ViT*): з 2020 року активно розвиваються моделі, побудовані на архітектурі трансформерів. Вони демонструють високу точність у задачах розпізнавання образів, проте для відеоаналітики у реальному часі поки що менш ефективні через потребу у значних обчислювальних ресурсах.

Сучасні алгоритми комп'ютерного зору базуються на використанні згорткових нейронних мереж, які демонструють високу точність при роботі з зображеннями та відеопослідовностями. Основною перевагою таких підходів є здатність автоматично виділяти значущі ознаки об'єктів без необхідності ручного налаштування параметрів.

Процес детекції об'єктів полягає у визначенні на зображенні областей, що відповідають заданим класам, наприклад людям або транспортним засобам.

На відміну від класичних методів, нейронні мережі здатні працювати в умовах зміни освітлення, часткових перекриттів та різних ракурсів зйомки.

Після детекції важливим етапом є трекінг об'єктів, який дозволяє встановити відповідність між об'єктами на сусідніх кадрах відеопотоку. Це дає змогу аналізувати траєкторію руху, фіксувати момент входу або виходу з контрольованої зони, а також зменшувати кількість хибних спрацьовувань системи.

Відстеження об'єктів є одним із ключових компонентів інтелектуальних систем відеоспостереження, що забезпечує визначення траєкторії руху об'єкта в часі після його первинної детекції. На відміну від детекції, яка аналізує об'єкти окремо на кожному кадрі, трекінг дозволяє встановити відповідність між об'єктами на сусідніх кадрах та формує їхні унікальні ідентифікатори (*ID*), що необхідно для уникнення дублювання та формування історії руху об'єктів.

Завдяки трекінгу можливо:

- прогнозувати напрям та швидкість руху;
- здійснювати підрахунок об'єктів;
- виявляти аномалії (раптові зупинки, вторгнення, зміна траєкторії);

– мінімізувати кількість викликів до моделі детекції, тим самим оптимізуючи обчислювальні ресурси.

Методи відстеження умовно поділяють на дві групи:

Однооб'єктний трекінг (*single-object tracking*, *SOT*) – відстеження одного об'єкта.

Багатооб'єктний трекінг (*multiple-object tracking*, *MOT*) – відстеження декількох об'єктів одночасно.

У системах відеоспостереження переважно застосовують багатооб'єктний трекінг (*MOT*), оскільки сцени зазвичай містять множину об'єктів (людей, транспорт, тварин).

Однооб'єктний трекінг: класичні алгоритми

До появи сучасних *CNN*-методів найчастіше використовували фільтраційні та кореляційні алгоритми відстеження. Найбільш поширені з них:

Алгоритм *KCF* (*Kernelized Correlation Filter*): алгоритм використовує кореляційні фільтри у просторовій та частотній областях для прогнозування позиції об'єкта між кадрами. Основна ідея полягає у побудові функції подібності, що шукає найбільш схожу область на наступному кадрі.

Переваги:

– дуже швидкий та легкий для *CPU*;

– працює у реальному часі (60 *FPS*+).

Недоліки:

– погано справляється із частковими перекриттями об'єктів;

– не підтримує повторну ідентифікацію.

Алгоритм *CSRT* (*Channel and Spatial Reliability Tracker*): є удосконаленням *KCF*, оскільки використовує просторову та канальну надійність для більш точного відстеження.

Переваги:

– висока точність у випадках з перекриттями;

– стійкість до зміни масштабу об'єкта.

Недоліки:

– нижча швидкість порівняно з *KCF*.

Висновок: *KCF* – для швидкості, *CSRT* – для точності.

Багатооб’єктний трекінг (*MOT*)

При відстеженні декількох об’єктів постає проблема асоціації – необхідно визначати, який об’єкт на кадрі відповідає певному *ID* з попередніх кадрів. *MOT*-алгоритми використовують детекцію як вхід і будують траєкторії на основі прогнозування та повторної ідентифікації об’єктів.

SORT (*Simple Online and Realtime Tracking*) – один із найпопулярніших алгоритмів *MOT*. Він поєднує детекцію об’єктів із використанням Калман-фільтра для прогнозування позицій і алгоритму Хангара для асоціації.

Алгоритм працює за схемою:

– детекція;

– прогноз позицій;

– асоціація;

– оновлення треків.

Переваги:

– висока швидкість обробки (часто 250 *FPS*+);

– простота реалізації.

Недоліки:

– не враховує візуальні ознаки об’єкта, лише геометрію рамки;

– некоректно працює при тимчасовому зникненні об’єкта або часткових перекриттях.

DeepSORT – розширення *SORT* із використанням глибинних ознак (*embeddings*) для повторної ідентифікації об’єктів (*Re-ID*).

Переваги:

– висока точність асоціації об’єктів між кадрами;

– підтримка *Re-ID* при тимчасовій втраті об’єкта;

– стійкість до перекриттів та зміни ракурсу.

Недоліки:

– більші обчислювальні витрати;

– залежність від якості моделі ознак.

DeepSORT є стандартом для трекінгу у сучасних системах безпеки, коли важлива стійкість і точність.

Оптимізація продуктивності та апаратне прискорення обробки відеосигналів: однією з головних вимог до систем інтелектуального відеоспостереження є забезпечення обробки відеоданих у режимі реального часу. Це особливо важливо для задач превентивної безпеки, де оперативність реагування безпосередньо впливає на ефективність системи.

Обробка відеопотоків, особливо у високій роздільній здатності та з використанням алгоритмів глибокого навчання, є обчислювально складною задачею. Тому критичним аспектом є оптимізація продуктивності шляхом використання апаратних та програмних засобів прискорення.

Принципи оптимізації продуктивності

Оптимізацію системи обробки відеоданих доцільно здійснювати на трьох рівнях:

- Алгоритмічний рівень – вибір ефективних моделей і скорочення зайвих обчислень.
- Програмний рівень – оптимізація коду, багатопотоковість, асинхронність.
- Апаратний рівень – використання прискорювачів обчислень (*GPU*, *TPU*, *NPU*).

На практиці застосування комбінованих підходів забезпечує найбільший приріст продуктивності.

Зменшення частоти викликів моделі детекції

Оскільки детекція є найбільш витратною операцією, поширеним підходом є детекція через *N* кадрів (наприклад, кожні 3–5 кадрів), а проміжні позиції об'єктів визначаються трекером. Використання зменшених вхідних розмірів: зменшення роздільності кадру, поданого на вхід моделі (наприклад, з 1280×720 до 640×360), часто дає можливість підвищити *FPS* на 30 – 60% при незначній втраті точності. **Багатопотоковість та асинхронна обробка.** У системах реального часу важливо забезпечити паралельну обробку декількох відеопотоків.

Для цього застосовуються:

- мультипроцесинг (*Python Multiprocessing*);

– асинхронна обробка (*AsyncIO*);

– черги повідомлень (*RabbitMQ*, *Kafka*).

Архітектура типу *producer–consumer* забезпечує розподілення навантаження між модулями.

Апаратне прискорення обробки відеосигналів: значного прискорення обробки відеоданих можна досягти за рахунок використання спеціалізованого апаратного забезпечення.

GPU (*Graphics Processing Unit*): графічні процесори забезпечують паралельне виконання тисяч операцій, що робить їх ідеальними для обробки нейронних мереж. Фреймворки *TensorFlow* та *PyTorch* використовують технологію *CUDA* (*Compute Unified Device Architecture*), яка дозволяє виконувати паралельні обчислення на *GPU*.

Таблиця 1.5

Типовий приріст продуктивності при переході з *CPU* на *GPU*

Модель	*FPS* на *CPU*	*FPS* на *GPU*
YOLOv5n	6 *FPS*	70–120 *FPS*
YOLOv8s	4 *FPS*	55–95 *FPS*
YOLOv8x	1 *FPS*	25–35 *FPS*

TPU розроблені компанією *Google* для прискорення інференсу моделі *TensorFlow*. Забезпечують високу швидкодію при низькому енергоспоживанні. Найчастіше використовуються у хмарних рішеннях.

NPU / *VPU* (*Neural*/*Vision Processing Unit*)

NPU та *VPU* призначені для локальної обробки *AI*-алгоритмів на периферійних пристроях (*edge computing*). Приклади: *Intel Movidius Myriad X*, *Google Coral Edge TPU*.

Переваги *edge*-підходу:

– низька затримка (без надсилання відео в хмару);

– підвищена конфіденційність;

– менша вартість масштабування.

Прискорення моделей через *TensorRT*, *ONNX* та *OpenVINO*. *ONNX* дозволяє конвертувати моделі з *PyTorch*/*TensorFlow* у нейтральний формат та запускати їх на різних пристроях без переписування коду.

Розподілена обробка відеопотоків: для великих систем відеоспостереження (10+ камер) доцільно застосовувати кластерну або хмарну обробку:

- *Kubernetes* кластери для *AI*;
- *NVIDIA DeepStream SDK;*
- *Kafka* + *Spark Streaming*.

Підхід *edge–fog–cloud* забезпечує баланс між локальною обробкою та централізованою аналітикою.

1.4. Огляд існуючих систем та програмних рішень у сфері інтелектуального відеоспостереження.

Класифікація сучасних систем інтелектуального відеоспостереження: *с*истеми відеоспостереження за останні два десятиліття пройшли значну трансформацію – від локальних аналогових систем до масштабованих інтелектуальних мережевих платформ з елементами штучного інтелекту та машинного навчання. Сучасний ринок рішень відеоаналітики є надзвичайно широким і різноманітним, що зумовлено зростанням потреб у безпеці, автоматизації та цифровій трансформації бізнес-процесів у приватному й державному секторах.

Для систематизації існуючих продуктів доцільно класифікувати системи інтелектуального відеоспостереження за низкою ознак:

- за архітектурою та способом зберігання даних;
- за рівнем інтелектуалізації;
- за сферою застосування;
- за типом ліцензування та моделі доступу.

Класифікація за архітектурою системи:

- Локальні системи (*On-Premise*)

Обробка та зберігання відеоданих здійснюється локально, на серверах або відеореєстраторах підприємства. Переваги – автономність, контроль над даними, висока конфіденційність. Недоліки – висока вартість обладнання та обмежена масштабованість.

– Хмарні системи (*Cloud-Based Video Surveillance*).

Відеодані зберігаються та аналізуються у хмарній інфраструктурі. Переваги – масштабованість, мінімальні початкові витрати, доступ з будь-якоїточки. Недоліки – залежність від інтернету, ризики конфіденційності.

Гібридні системи (*Hybrid / Edge + Cloud*). Поєднують локальну передобробку (*edge processing*) з хмарною аналітикою. Це найбільш сучасна модель, рекомендована для систем з *AI*. Переваги – оптимальне навантаження, низька затримка, підвищена безпека.

Класифікація за рівнем інтелектуалізації:

Класичні системи відеоспостереження (*CCTV*). Включають лише перегляд та запис відео без автоматичного аналізу. Їх роль у сучасних системах зведена до мінімуму.

Системи з базовою відеоаналітикою. Містять прості алгоритми виявлення руху, перетину лінії, вторгнення в зону, розпізнавання облич на основі класичних методів (*Haar*, *HOG*). Недолік – низька точність і велика кількість хибних спрацювань.

Інтелектуальні системи відеоспостереження (*Intelligent Video Surveillance*, *IVS*).

Використовують алгоритми *Deep Learning* для аналізу відео, включаючи:

– детекцію та класифікацію об’єктів;

– трекінг та поведінковий аналіз;

– розпізнавання подій та аномалій;

– прогнозування ризикових ситуацій.

Платформи з *AI*-аналітикою у сфері відеоспостереження та їхні обмеження: поява і швидкий розвиток технологій глибокого навчання сприяли формуванню нового сегменту ринку – *AI*-орієнтованих систем відеоаналітики, які забезпечують автоматичне виявлення, класифікацію та аналіз об’єктів і подій у відеопотоці без участі оператора.

На відміну від традиційних *VMS*, ці системи роблять акцент на розумінні контексту сцени і здатні виконувати складні аналітичні функції, включаючи аналіз поведінки, виявлення аномалій та формування звітів.

Серед відомих рішень цієї категорії – *BriefCam*, *Ivideon*, *Camlytics*, *DeepView*, *Sighthound*, *Clarifai*, *Face++*, *AnyVision*.

Попри суттєвий технологічний прорив, навіть провідні *AI*-платформи мають недоліки, що відкривають можливості для створення більш адаптивних та гнучких рішень.

BriefCam – одна з найвідоміших систем відеоаналітики преміального класу, що використовує *AI* для аналізу відео з можливістю швидкого перегляду подій (*Video Synopsis*) та уточненого пошуку за атрибутами.

Рис. 1.1. Логотип компанії *BriefCam*

Ключові можливості:

- виявлення об’єктів за типами (люди, транспорт, тварини);
- фільтрація за атрибутами (одяг, колір, напрямок руху, локація);
- пошук за часовими інтервалами;
- аналітика поведінки об’єктів;
- інструменти для розслідувань (пост-фактум аналіз подій).

Недоліки та обмеження:

- висока вартість ліцензій та обслуговування, орієнтація лише на корпоративний сегмент.

– закритість алгоритмів і відсутність можливості кастомізації моделей під нестандартні сценарії.
– система більше орієнтована на пост-фактум аналіз, а не на оперативну аналітику в реальному часі.

Ivideon – популярна хмарна платформа, що надає відеоспостереження за моделлю *SaaS* із базовими та розширеними *AI*-функціями.

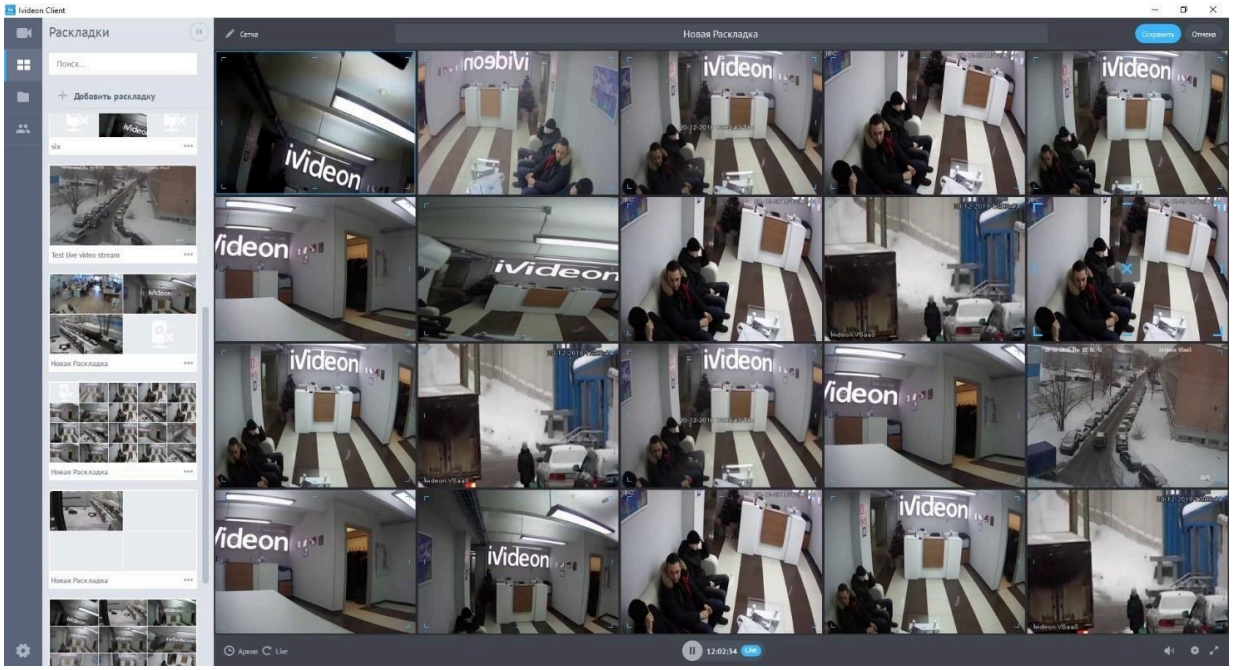

Рис. 1.2. Приклад роботи хмарної платформи *Ivideon*

Переваги:

– простота впровадження;
– наявність хмарної аналітики;
– мультиплатформність і мобільний доступ;
– підпискова модель без великих стартових витрат.

Недоліки та обмеження:

– більшість *AI*-функцій доступні лише у дорогих тарифах.
– обмежена точність та набір *AI*-модулів порівняно з конкурентами (особливо щодо трекінгу та поведінкового аналізу).

– залежність від хмарної інфраструктури та інтернет-каналу, що викликає проблеми в умовах нестабільності мережі або високих вимог до конфіденційності.

Camlytics – система відеоаналітики, орієнтована переважно на бізнес-аналітику, підрахунок відвідувачів та маркетингові метрики.

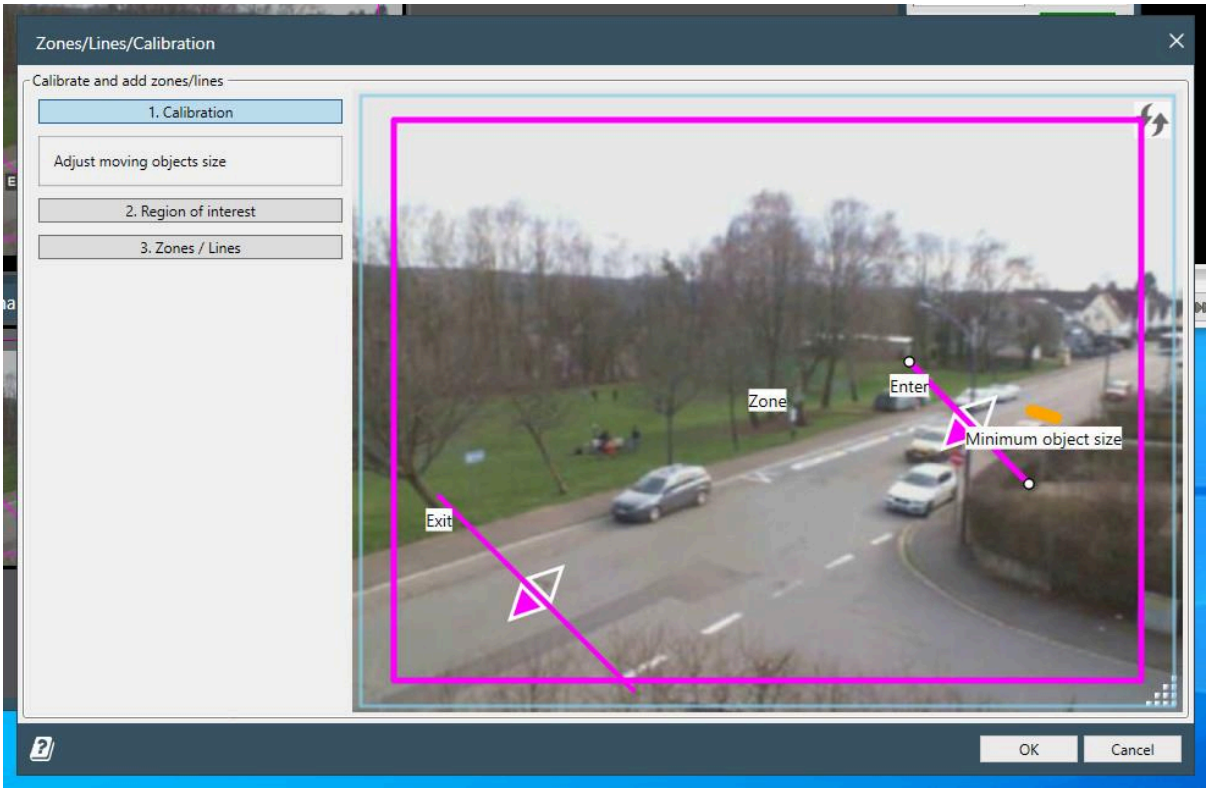

Рис. 1.3. Приклад роботи система відеоаналітики *Camlytics*

Переваги:

– широкий інструментарій аналітики для ритейлу;

– порівняно доступна вартість;

– гнучка система звітів і бізнес-метрик.

Недоліки та обмеження:

Система не орієнтована на безпекові сценарії та має обмежені можливості в розпізнаванні складних подій. Обмежена масштабованість при роботі з великою кількістю камер. Відсутність глибокої *AI*-аналітики поведінки, прогнозування.

DeepView позиціонується як *AI*-платформа для розпізнавання облич, номерних знаків, об’єктів та поведінки.

Переваги:

– висока точність розпізнавання облич;

– аналітика для бізнесу і безпеки;

– гнучкі інтеграційні можливості.

Недоліки та обмеження:

– фокус на обмеженій кількості *AI*-задач (обличчя, номери авто), чого недостатньо для комплексних систем відеоспостереження;

– відсутність повноцінного ядра *VMS*, що потребує додаткового програмного забезпечення для керування камерами;

– висока вартість для малого бізнесу.

Таблиця 1.6

Узагальнення недоліків *AI*-платформ

Недолік	Суть
Висока вартість *AI*-функцій	*SaaS*-моделі роблять повну аналітику недоступною для широких користувачів
Закритість моделей та *API*	Немає можливості навчати чи адаптувати моделі під власні задачі
Хмарна залежність	Відсутність або слабка підтримка *edge*-обчислень
Низька масштабованість для МСП	Продукти або надто складні, або надто дорогі
Обмежена гнучкість	Системи не дозволяють налаштовувати аналітику користувачем

Open-source рішення у сфері інтелектуального відеоспостереження та їхні обмеження: *Open-source* платформи відеоспостереження відіграють важливу роль у розвитку галузі, оскільки надають розробникам та організаціям можливість впроваджувати системи без значних ліцензійних витрат.

Відкритий вихідний код забезпечує гнучкість, можливість кастомізації та інтеграції із сучасними технологіями комп’ютерного зору та машинного навчання.

Найбільш відомими *open-source* рішеннями є *ZoneMinder*, *Shinobi*, *Kerberos.io*, *OpenCV-based* системи та *NVIDIA DeepStream*.

Незважаючи на популярність, більшість відкритих платформ мають значні обмеження щодо масштабованості, точності аналітики, продуктивності тазручності використання, що ускладнює їхню інтеграцію як готового комерційного продукту.

ZoneMinder є одним з найстаріших *open-source* рішень для відеоспостереження, яке функціонує як повноцінна *VMS*-платформа.

Переваги:

- повністю безкоштовна система;
- підтримка великої кількості *IP* – та аналогових камер;
- гнучкі можливості конфігурації та модульна архітектура;
- активна спільнота користувачів.

Недоліки та обмеження:

- застаріла архітектура та інтерфейс, що ускладнює використання;
- низька продуктивність при роботі з великою кількістю камер;
- *AI*-аналітика відсутня «з коробки», потребує складної інтеграції сторонніх рішень;
- складність розгортання та налаштування для нефахових користувачів.

Shinobi позиціонується як сучасніша альтернатива *ZoneMinder* з кращою продуктивністю та *API*-орієнтованою архітектурою.

Переваги:

- сучасний веб-інтерфейс та *API-first* підхід;
- легкість інтеграції з *Node.js*-екосистемою;
- підтримка мультиплатформності та хмарних розгортань;
- активний розвиток та спільнота.

Недоліки та обмеження:

- базова версія – без *AI*-аналітики, а модулі штучного інтелекту є платними або потребують складного налаштування;
- складність масштабування без використання контейнеризації та оркестрації;

– обмежена якість вбудованого детектора руху, що призводить до великої кількості хибних тривог.

Kerberos.io – легковага система із фокусом на простому розгортанні.

Переваги:

– простота встановлення (підтримка *Docker*, *Raspberry Pi*, *ARM*);

– мінімальні вимоги до ресурсів;

– гарно підходить для *edge*-рішень.

Недоліки та обмеження:

– мінімальний функціонал відеоаналітики;

– обмежена масштабованість;

– відсутність повноцінної трекінг-аналітики, *Re-ID*, поведінкового аналізу.

OpenCV є найпопулярнішим *open-source* фреймворком для комп'ютерного зору, який часто використовується для побудови кастомних систем відеоаналітики[18].

Переваги:

– величезна бібліотека алгоритмів *CV*, включаючи класичні та *CNN*-моделі;

– можливість повної кастомізації;

– інтеграція з *Python*, *C++*, *Java*, *Node.js*;

– доступність та активна спільнота розробників.

Недоліки та обмеження:

– *OpenCV* – не *VMS*-система, а набір інструментів;

– відсутність готового *UI*/інтерфейсу для операторів;

– необхідність значного обсягу розробки "від нуля" для створення повноцінного продукту;

– потреба оптимізації для реального часу та *GPU*.

DeepStream – SDK для створення високопродуктивних систем відеоаналітики на *GPU* з підтримкою *edge*-та *cloud*-розгортань.

Переваги:

– надвисока продуктивність на *NVIDIA GPU*;

– вбудовані конвеєри для детекції, трекінгу, аналітики;

– підтримка *YOLO*, *EfficientDet*, *DeepSORT* та інших моделей;

– можливість масштабування для *Smart City* та корпоративних рішень.

Недоліки та обмеження:

– високий поріг входу для розробників без досвіду з *CUDA* та *GStreamer*.

– відсутність інтуїтивного *UI* – потрібна власна розробка *VMS*-інтерфейсу.

– залежність від *NVIDIA* обладнання.

Таблиця 1.7

Загальні висновки та недоліки *open-source* рішень

Недолік	Як проявляється
Відсутність *ready-to-use AI*	*AI* потрібно інтегрувати окремо, що складно і довго
Слабка масштабованість	Системи не розраховані на 50+ камер без глибокої оптимізації
Відсутність єдиного продукту	Є фреймворки або часткові рішення, але не комплексні *IVS*
Високі вимоги до техзнань	Потрібні навички *DevOps*, *CV*, *ML* та системного адміністрування
Недостатня *UX*-орієнтованість	Інтерфейси складні для звичайних користувачів

Висновки та формування технічних вимог до нового програмного засобу*:* на основі проведеного аналізу комерційних, *AI*-орієнтованих та *open-source* систем відеоспостереження можна зробити висновок, що сучасний ринок інтелектуальних відеорішень перебуває у фазі активного розвитку, але характеризується низкою системних недоліків та обмежень, які стримують широке впровадження високоточної відеоаналітики у різних сферах. Жодна з розглянутих груп рішень не забезпечує оптимального балансу між інтелектуальністю, доступністю, безпекою, гнучкістю та простотою використання.

Головні виявлені проблеми ринку. Надмірна вартість та ліцензійні бар’єри. Висока вартість комерційних систем та окремих *AI*-модулів робить їх недоступними

для малого бізнесу, приватного сектору та бюджетних установ. Ринок потребує доступних рішень із прозорою ціновою політикою.

Закритість екосистем і відсутність гнучкості. Більшість платформ не надають доступу до моделей, *API* або можливості адаптації аналітики під власні сценарії. Користувачі прагнуть мати контроль над логікою обробки відеоданих та можливість кастомізації.

Хмарна залежність і ризики конфіденційності. *AI*-платформи орієнтовані на хмарну обробку, що не підходить для систем із вимогами до приватності, оборонних або критичних об'єктів. Ринок потребує *edge*-рішень з локальною обробкою даних без втрати конфіденційності.

Складність впровадження та використання *open-source* систем. *Open-source* платформи потребують значних технічних компетенцій, *DevOps*-інфраструктури та інтеграцій, що унеможливлює їх масове застосування в ролі готових продуктів. Потрібний готовий інструмент, який поєднує доступність *open-source* з простотою комерційних рішень.

Обмежена точність та стійкість *AI*-аналітики у бюджетному сегменті. Дешеві системи мають високий рівень *false-positive* та *false-negative*, що робить їх малоефективними у реальних умовах. Потрібне рішення з оптимальним співвідношенням точності та продуктивності.

Необхідні функціональні характеристики нового програмного засобу: на підставі проаналізованих недоліків сформульовано базовий набір вимог до нового програмного засобу для обробки відеосигналів у реальному часі:

Вбудована інтелектуальна відеоаналітика *(AI/Deep Learning)*:

- детекція та класифікація об'єктів у реальному часі;
- трекінг та повторна ідентифікація *(Re-ID)*;
- аналіз поведінки та виявлення аномалій;
- можливість навчання моделей під специфічні сценарії.

Підтримка *edge*-обробки даних:

- локальна обробка відео без необхідності відправлення в хмару;
- використання *GPU*/*NPU* (за наявності) для прискорення.

Інтуїтивний графічний інтерфейс *(UI/UX)*:

– орієнтований на масового користувача, без потреби глибоких технічних знань;
– інтерактивні панелі, сповіщення, перегляд подій та аналітики.

Гнучка архітектура та розширюваність:

– модульна структура (плагінна система або мікросервіси);
– відкриті *API* для інтеграції з іншими системами;
– можливість додавання власних моделей та аналітичних модулів.

Масштабованість:

– підтримка різної кількості камер: від домашнього застосування (1–4 камери) до корпоративного (50+ камер);
– можливість розгортання у локальному або гібридному середовищі.

Безпека та конфіденційність даних:

– шифрування потоків та даних;
– відповідність стандартам кібербезпеки *(ISO/IEC 27001, GDPR-ready)*;
– контроль доступу та логування подій.

Порівняльний аналіз існуючих систем відеоспостереження

На сьогоднішній день існує велика кількість систем відеоспостереження, які відрізняються за функціональними можливостями, архітектурою, рівнем автоматизації та сферою застосування.

Більшість сучасних рішень спрямовані на забезпечення відеофіксації подій, зберігання відеоархівів та базовий аналіз зображень.

Традиційні системи відеоспостереження, як правило, виконують лише функції запису та відтворення відеопотоку.

Аналіз відео в таких системах здійснюється оператором вручну, що значно знижує ефективність роботи та підвищує ймовірність пропуску важливих подій. Крім того, постійний перегляд відеопотоків вимагає значних людських ресурсів.

Сучасні інтелектуальні системи відеоспостереження частково вирішують ці проблеми за рахунок використання алгоритмів комп’ютерного зору та машинного навчання. Вони здатні автоматично виявляти об’єкти у кадрі, аналізувати рух, визначати зони інтересу та фіксувати події без безпосередньої участі оператора.

Проте такі системи часто мають складну архітектуру, потребують значних обчислювальних ресурсів і складні в налаштуванні.

Одним із ключових напрямів розвитку є підвищення точності детекції та аналізу об’єктів у складних умовах, таких як низька освітленість, динамічні сцени або наявність перешкод у кадрі.

Перспективним напрямом є інтеграція глибших нейромережевих моделей, здатних не лише виявляти об’єкти, а й виконувати більш складний семантичний аналіз сцени. Це дозволяє перейти від простого фіксування подій до прогнозування потенційно небезпечних ситуацій.

Значну роль відіграє розвиток локальних систем відеоаналізу, які не потребують постійного підключення до хмарних сервісів. Такі рішення є більш безпечними з точки зору збереження конфіденційності даних та можуть застосовуватися у приватних або обмежених середовищах.

Окрему увагу приділяють оптимізації алгоритмів для роботи в реальному часі. Зменшення обчислювального навантаження дозволяє використовувати інтелектуальні системи відеоспостереження на пристроях з обмеженими ресурсами без втрати функціональності.

Таким чином, розвиток інтелектуальних систем відеоспостереження спрямований на підвищення автономності, надійності та адаптивності програмних рішень, що підтверджує актуальність обраної тематики дослідження.

Комерційні рішення зазвичай є закритими системами, що обмежує можливість їх модифікації та адаптації під конкретні завдання. Крім того, вартість ліцензійного програмного забезпечення та обладнання може бути досить високою, що ускладнює їх використання в навчальних або малобюджетних проєктах.

На відміну від існуючих рішень, у даній дипломній роботі розробляється програмний засіб, орієнтований на обробку відеосигналів у реальному часі з використанням відкритих програмних бібліотек і попередньо навченої нейромережевої моделі. Запропонований підхід дозволяє зменшити складність системи, забезпечити гнучкість налаштувань та спростити інтеграцію в різні середовища.

Таким чином, аналіз існуючих систем відеоспостереження показує доцільність розробки власного програмного засобу, який поєднує можливості інтелектуального аналізу відео з простотою реалізації та практичною спрямованістю.

Особливості використання обробки відеосигналів у реальному часі

Обробка відеосигналів у реальному часі є ключовою особливістю сучасних інтелектуальних систем відеоспостереження.

На відміну від офлайн-аналізу, де відеоматеріал опрацьовується після його запису, системи реального часу виконують аналіз кожного кадру безпосередньо під час надходження відеопотоку. Це дозволяє оперативно реагувати на події, що відбуваються в зоні спостереження, та формувати відповідні повідомлення або записи без затримок.

Однією з головних вимог до таких систем є висока швидкодія. Алгоритми комп'ютерного зору та нейронні мережі повинні працювати з достатньою частотою кадрів, щоб не втрачати важливу інформацію.

Для цього застосовуються оптимізовані моделі детекції об'єктів, зменшені розміри вхідних зображень, а також використання апаратного прискорення, зокрема графічних процесорів.

Важливу роль відіграє попередня обробка відеосигналу. До неї належать операції масштабування, нормалізації, фільтрації шумів та перетворення кольорових просторів.

Такі дії дозволяють зменшити обчислювальне навантаження та підвищити стабільність роботи алгоритмів детекції. Попередня обробка також сприяє більш точному визначенню об'єктів у складних умовах освітлення або при низькій якості відео.

У системах відеоспостереження реального часу часто використовується поняття контрольованої зони (*ROI*). Аналіз відеопотоку обмежується визначеною ділянкою кадру, що дозволяє зосередити обчислювальні ресурси лише на важливих областях. Це знижує кількість хибних спрацювань та підвищує загальну ефективність системи.

Ще однією особливістю є необхідність стабільного трекінгу об'єктів між послідовними кадрами. Трекінг дозволяє визначати траєкторію руху об'єктів, їх

перебування у контрольованій зоні та тривалість події. Саме поєднання детекції та трекінгу забезпечує коректну реєстрацію подій і запобігає дублюванню записів.

Таким чином, обробка відеосигналів у реальному часі вимагає збалансованого поєднання швидкодії, точності та оптимального використання ресурсів. Реалізація таких підходів дозволяє створювати ефективні програмні засоби, здатні працювати автономно та забезпечувати надійний аналіз відеоданих у практичних умовах експлуатації.

1.5. Висновки по розділу

У ході виконання дипломної роботи було проведено ґрунтовний аналіз сучасних систем відеоспостереження, які активно застосовуються у різних сферах діяльності, зокрема в системах безпеки, комерційних об’єктах, транспортній та критичній інфраструктурі. Особливу увагу приділено архітектурним підходам до побудови таких систем, способам організації відеопотоків та методам обробки великих обсягів відеоданих у режимі реального часу.

Розглянуто основні принципи функціонування систем відеоспостереження, їх ключові функціональні можливості та сучасні тенденції розвитку. Зокрема, відзначено перехід від традиційних пасивних систем, що виконують лише функцію фіксації подій, до інтелектуальних відеосистем з елементами автоматичного аналізу відеоданих. Такий підхід дозволяє не лише зберігати відеоінформацію, а й здійснювати її оперативну обробку з метою виявлення потенційно небезпечних ситуацій.

Встановлено, що використання алгоритмів комп’ютерного зору суттєво підвищує ефективність обробки відеопотоків, зменшує кількість хибних спрацювань та значно скорочує залежність від постійного контролю з боку оператора. Автоматизовані методи аналізу дозволяють своєчасно реагувати на події та забезпечують більш стабільну роботу системи за умов обмежених людських ресурсів.

Також у роботі досліджено існуючі підходи та алгоритми комп’ютерного зору, що застосовуються для детекції та відстеження об’єктів у відеопотоці. Проведено порівняльний аналіз класичних методів та сучасних нейромережевих рішень з точки

зору точності, обчислювальної складності та можливості використання в режимі реального часу. Окремо розглянуто особливості інтеграції алгоритмів трекінгу для підтримки ідентифікації об’єктів між кадрами.

За результатами проведеного аналізу зроблено висновок, що сучасні нейромережеві методи детекції у поєднанні з алгоритмами трекінгу є найбільш доцільними для застосування в інтелектуальних системах відеоспостереження. Таке поєднання забезпечує оптимальний баланс між точністю розпізнавання, швидкодією та стабільністю роботи системи, що є критично важливим для практичного використання в реальних умовах експлуатації.

РОЗДІЛ 2
СТРУКТУРА ПРОГРАМНОГО ЗАСОБУ

2.1. Аналіз вимог до програмного засобу

Метою розділу є формалізація вимог до ПЗ для обробки відеосигналів у реальному часі на базі стеку *Python* + *OpenCV* + *Ultralytics* (*YOLOv8*/*YOLO11*) + *PySide6*/*PyQt* із десктопним інтерфейсом. Вимоги сформульовано з урахуванням результатів Розділу 1, практичних сценаріїв експлуатації (1–16 камер), обмежень обчислювальних ресурсів і нормативних обмежень щодо конфіденційності.

Функціональні вимоги:

Підключення джерел відео. Підтримка локальних пристроїв (*USB*/*UVC*-камери). Підтримка мережевих потоків *RTSP*/*HTTP*(*MJPEG*). Керування джерелами: додати/видалити/тимчасово вимкнути; збереження профілів підключення. Перевірка доступності джерела та автоматичний *reconnect* при втраті зв'язку.

Попередня обробка відеосигналу *(DSP).* Конфігурована роздільність інференсу (наприклад, 640×… апскейл у віджет). Опції нормалізації/фільтрації, деблюр/денойз. Конвертації колірних просторів *(RGB/HSV/YCrCb)* – за потреби до алгоритмів.

Детекція та класифікація об'єктів (*AI*). Інференс моделей *YOLOv8*/*YOLO11* (*Ultralytics*) у реальному часі. Налаштування порогів *conf*/*IoU*, вибір класів для моніторингу (люди, транспорт тощо). Підтримка перевантаження моделі користувацькими вагами (*fine-tuned weights*). Експорт/імпорт профілів налаштувань детекції.

Відстеження (трекинг) об'єктів:

– Інтеграція *DeepSORT*/*SORT* (або *ByteTrack*) для призначення *ID* траєкторіям.

– Збереження атрибутів об’єкта (час входу/виходу, позиція, швидкість оцінн.).

– Облік зникнень/повторних появ (*Re-ID*) у межах однієї сесії.

Правила подій та сповіщення:

– Події: “вторгнення в зону”, “перетин лінії”, “зупинка/біг”, “залишений предмет” (мінімальний набір).

– Налаштування *ROI*/зон (полігони) та виртуальних ліній у графічному режимі.

– Генерація сповіщень (*toast* у *UI*, звук), опціонально – запис у журнал подій.

Запис та архівація:

– Безперервний запис або подієвий (*motion*/детекція).

– Формати: контейнер *MP4*/*MKV*, кодек *H.264*/*H.265* (за можливості системи).

– Керування ротацією архіву (макс. обсяг/дні зберігання), швидкий експорт фрагментів.

Аналітичні панелі та журнал подій:

– Онлайн-статистика: кількість об’єктів/класів, *FPS*, затримка.

– Журнал подій (час, камера, тип, параметри), пошук/фільтрація/експорт (*CSV*).

Управління конфігурацією

– Збереження конфігів у *SQLite* (локально): джерела, профілі, зони, правила.

– Імпорт/експорт конфігурації для перенесення між машинами.

Адміністрування та безпека доступу

– Локальна авторизація (мінімум: пароль адміністратора).

– Ролі (*Admin*/*Operator* – опціонально): доступ до зміни параметрів/перегляду.

Нефункціональні вимоги:

Надійність і відмовостійкість:

– Автоматичне відновлення потоків (*reconnect* і *backoff*).

– Логування помилок (файл/консоль), журнал системних подій.

– Контроль некерованих виключень.

Масштабованість та розширюваність:

– Модульна архітектура: ядро відеоконвеєра, модуль *AI*, модуль трекінгу, модуль подій, модуль запису, *UI*.

– Відкриті інтерфейси між модулями (класи/сигнали), можливість додати нові детектори або замінити трекер.

Безпека та конфіденційність:

– Локальне зберігання архіву (без хмари за замовчуванням).

– Обмеження доступу до архіву (пароль, ролі).

– Маскування конфіденційних зон (*privacy masks*) на етапі відображення/запису – опція.

– Дотримання принципів мінімізації даних (налаштовувані строки зберігання).

Юзабіліті:

– Час першого запуску з додаванням камери – ≤ 5 хв для некваліфікованого користувача.

– Адекватна швидкодія *UI* при 4-х одночасних прев'ю 720*p* (грайд/тайл).

Сумісність:

– *Windows 10/11*, додатково опційно *Linux* (*Ubuntu 22.04+*).

– Підтримка *GPU NVIDIA* (*CUDA*), працездатність на *CPU* – із деградацією продуктивності.

Вимоги до інтерфейсу користувача (*UI*/*UX*):

Загальні принципи:

– Мінімалістичний десктопний *UI* на *PySide6*/*PyQt* з адаптивними панелями.

– Основні сценарії «у два кліки»: додати камеру, побачити детекцію, поставити зону/подію і в кінці отримати сповіщення.

Основні екрани/вікна

Головне вікно (*Dashboard*):

– сітка прев'ю (1/2×2/3×3), швидке перемикання розкладок.

– індикатори *FPS*/*latency*, статуси потоків.

– глобальна панель керування (старт/стоп, профілі).

Джерела відео:

– додати *RTSP*/локальну камеру; перевірити підключення; назва/група.

– конфіги декодера (буферизація, розмір кадру на інференсі).

Аналітика (*AI*):

– вибір моделі *YOLO* (*v8n*/*s*/*m*/*x*), шлях до ваг.

– пороги *conf*/*IoU*, класи; частота детекції (кожен *N*-й кадр).

Ергономіка:

– Кольорове кодування класів об’єктів і типів подій.

– «Підказки» (*tooltips*) з коротким описом кожного параметра.

– Станова панель (*status bar*) з короткими логами/помилками.

– Не блокувати *UI* під час інференсу – використовувати потоки/черги.

Зберігання та архів

– Швидкий *SSD* для індексу та метаданих; окремий *HDD*/*SSD* для відеоархіву.

Безпека

– Локальні облікові дані, шифрування конфігів (паролі *RTSP*) хоча б базове.

– Обмеження прав доступу до файлів архіву (*FS ACL*).

2.2. Вибір інструментальних засобів та технологій розробки

Вибір інструментальних засобів є ключовим етапом проєктування програмного забезпечення, оскільки він визначає можливості реалізації функціоналу, продуктивність, підтримуваність коду та перспективи подальшого розширення системи. Для реалізації програмного засобу обрано мову програмування *Python* у поєднанні з бібліотеками *OpenCV*, *Ultralytics* (*YOLO*) та фреймворком *PySide6*/*PyQt* для побудови графічного інтерфейсу користувача[3]. Дані інструменти забезпечують оптимальний баланс між швидкістю розробки, функціональністю та якістю реалізації відеоаналітики.

Обґрунтування вибору *Python:*

Python є однією з найпоширеніших мов програмування у сфері комп’ютерного зору, штучного інтелекту та прототипування систем обробки сигналів. Основними факторами вибору *Python* для реалізації програмного засобу є:

– Широка екосистема інструментів для комп’ютерного зору та машинного навчання. *Python* має прямий доступ до провідних бібліотек, що використовуються у відеоаналітиці:

Під час розробки програмного модуля інтелектуальної системи відеоспостереження важливим етапом є обґрунтований вибір програмних засобів та бібліотек. Мова програмування *Python* була обрана завдяки своїй простоті, широкій підтримці та наявності великої кількості бібліотек для обробки відеосигналів і машинного навчання.

Бібліотека *OpenCV* використовується для роботи з відеопотоком, оскільки вона забезпечує ефективні інструменти для захоплення, обробки та попереднього аналізу кадрів. Для реалізації детекції об’єктів було обрано модель *YOLO*, яка поєднує високу швидкість роботи та прийнятну точність.

Для збереження результатів аналізу використовується база даних *SQLite*, яка не потребує окремого сервера та легко інтегрується в локальні програмні рішення. Такий вибір дозволяє забезпечити простоту використання та мінімізувати вимоги до апаратних ресурсів.

Сукупність обраних технологій дозволяє створити ефективний програмний засіб, який відповідає поставленим вимогам і може бути використаний у практичних умовах.

OpenCV – базові та високорівневі функції обробки зображень і відео, *Ultralytics/YOLOv8–YOLO11*, *PyTorch*, *TensorFlow* – моделі глибинного навчання, *NumPy*, *SciPy* – оптимізація обчислень.

Швидкість розробки та легкість підтримки коду *Python* має високу читабельність, мінімальну кількість синтаксичного «шуму» та дозволяє швидко створювати робочі прототипи, що є критично важливим у межах дипломного проєкту.

Підтримка крос-платформеності. Код, написаний на *Python*, може бути розгорнутий у середовищах *Windows*, *Linux* та *macOS* з мінімальними адаптаціями, що спрощує тестування та експлуатацію. Активна спільнота та наявність готових рішень *Python* має велику спільноту розробників, що забезпечує доступ до документації, прикладів, форумів та полегшує вирішення технічних проблем.

Попри те, що *Python* поступається мовам *C++* та *Rust* у швидкості виконання низькорівневих операцій, оптимізація може бути досягнута за рахунок використання *GPU*-інференсу, векторизації обчислень, паралельної обробки потоків та інтеграції *C++*-модулів у критичних ділянках.

OpenCV як інструмент цифрової обробки відеосигналів

OpenCV (*Open Source Computer Vision Library*) є стандартом де-факто у сфері класичної цифрової обробки зображень та відео. Основні причини вибору *OpenCV*:

- реалізація ключових алгоритмів *DSP* та *CV*: фільтрація, нормалізація, геометричні трансформації, контурний аналіз, віднімання фону;
- підтримка апаратного прискорення (*OpenCL*, *CUDA* – для окремих операцій);
- інтеграція з інструментами відеодекодування та використання *FFmpeg*;
- можливість обробки відеопотоку в реальному часі.

OpenCV є базовим компонентом для попередньої обробки кадрів перед передаванням до *AI*-моделі.

Ultralytics (*YOLO*) як ядро інтелектуальної аналітики

Для детекції та класифікації об’єктів вибрано фреймворк *Ultralytics YOLO* (версії *v*8 або *v*11). Це зумовлено наступними факторами:

- висока точність і швидкодія на *GPU* у реальному часі;
- підтримка різних масштабів моделей (*n*/*s*/*m*/*l*/*x*) для балансування точності та *FPS*;
- можливість донавчання моделі під власні дані;
- підтримка експорту у *ONNX*, *TensorRT*, *CoreML*, *OpenVINO*;
- простота інтеграції у *Python*-середовище.

Таким чином, *Ultralytics* забезпечує інтелектуальне ядро системи, здатне до ефективної роботи в рамках десктопного застосунку.

PySide6 / *PyQt* як інструмент побудови *UI*

Для створення графічного інтерфейсу обрано *PySide6* (*Qt for Python*) або аналогічний *PyQt6*, що забезпечує:

Сучасний, інтуїтивно зрозумілий *UI* у стилі десктопних систем. Широкі можливості для побудови динамічних вікон, таблиць, панелей, відеовіджетів.

Підтримку багатопоточності (*QThread*) для відокремлення інференсу та *UI*. Добру продуктивність при рендерингу відеокадрів та графічних оверлеїв. *PySide6* обрано як пріоритетний варіант через більш гнучку ліцензію (*LGPL*), що робить систему юридично простішою для можливого поширення.

Вибір фреймворків для відеообробки та штучного інтелекту. Проєктування інтелектуального програмного засобу для обробки відеосигналів у реальному часі передбачає використання спеціалізованих фреймворків для обробки зображень, керування відеопотоками та застосування моделей глибинного навчання. Важливою вимогою є досягнення високої продуктивності та точності аналізу, при цьому забезпечуючи можливість масштабування, адаптації та оновлення компонентів моделі без суттєвої переробки системи.

Вибір фреймворку для опрацювання відеосигналів: *OpenCV*

Основним інструментом для роботи з відеопотоками обрано бібліотеку *OpenCV*[6].

Вона забезпечує повний набір функцій для обробки відео, починаючи з декодування потоків та закінчуючи попередньою обробкою кадрів перед подачею в нейромережу.

Ключові переваги *OpenCV* для даного проєкту:

– Універсальність і широке поширення. *OpenCV* є стандартом у галузі комп'ютерного зору, має відкритий код, активну спільноту та детальну документацію. Це гарантує стабільність, сумісність і надійність при реалізації лабораторних та промислових рішень.

Підтримка потокового відео в реальному часі. *OpenCV* має інтеграцію з *FFmpeg* та *GStreamer*, що дозволяє обробляти відеопотоки з *USB*-камер, *IP*-камер та *RTSP*-джерел із мінімальними затримками.

Модулі для попередньої обробки зображень (*DSP*). *OpenCV* реалізує широкий спектр алгоритмів фільтрації, корекції, морфологічних операцій, сегментації та детекції контурів, що дозволяють підвищити якість вхідних даних для нейромережі. Підтримка апаратного прискорення. Деякі операції в *OpenCV* можуть виконуватися на *GPU* (*CUDA*-модулі) або з використанням апаратної оптимізації *OpenCL*, що допомагає підвищити *FPS*.

Вибір фреймворку для *AI*-моделі детекції: *Ultralytics YOLO*

Для інтелектуальної відеоаналітики – детекції, класифікації та аналізу об'єктів – обрано архітектуру *YOLO* (*You Only Look Once*) у реалізації *Ultralytics*. Вибір зумовлений високою точністю, здатністю працювати в реальному часі та простотою інтеграції.

Переваги застосування *Ultralytics YOLO* в даному проєкті:

Готовність до реального часу (*Real-Time*).*YOLOv8*/*YOLO11* демонструє чудове співвідношення точності й швидкості: легкі моделі (*n, s*) забезпечують обробку 25–60 *FPS* на сучасних *GPU*, що повністю відповідає вимогам системи.

Підтримка різних рівнів точності та продуктивності. Наявність шкали моделей (*n*, *s*, *m*, *l*, *x*) дозволяє обрати оптимальну версію в залежності від апаратних ресурсів користувача.

Простота інтеграції та розширення. *Ultralytics* надає високорівневий *API* для запуску інференсу, тренування та експорту моделей у різні середовища. Це скорочує час розробки та знижує поріг входу.

Можливість донавчання моделі (*transfer learning*). За потреби система може бути адаптована під специфічні класи об'єктів (наприклад, виробничі дефекти, транспортні засоби, каски працівників тощо), що підсилює прикладну цінність ПЗ.

Широкі можливості оптимізації та портативності. *Ultralytics* дозволяє експортувати моделі у формати *ONNX*, *TensorRT*, *OpenVINO*, що відкриває можливість подальшої оптимізації під різні апаратні платформи (*GPU*, *CPU*, *VPU*, *edge*-пристрої).

Модуль відстеження об'єктів: *DeepSORT* / *ByteTrack*

Детекція об'єктів сама по собі не достатня для аналізу поведінки, побудови статистики та виявлення подій. Тому до проєкту включено модулі трекінгу.

DeepSORT обрано як базовий варіант, оскільки він забезпечує:

- стійке відстеження об'єктів між кадрами;
- присвоєння унікальних *ID*;
- коректну обробку перетинів траєкторій і часткових перекриттів;
- можливість використання *Re-ID* векторів.

Допоміжні фреймворки та бібліотеки: *о*крім основних інструментів, до складу системи можуть бути включені: *NumPy*/*SciPy*, *FFmpeg* (через *OpenCV*), *SQLAlchemy*/*SQLite*, *QtMultimedia*, *ONNX Runtime*

ByteTrack розглядається як спрощений та високопродуктивний варіант для систем із масовими сценами (вулиці, транспортні потоки). Можливе використання *ByteTrack* як альтернативного трекера у майбутніх версіях системи.

Оцінка можливих архітектур програмного засобу. Архітектура програмного забезпечення визначає логіку взаємодії між модулями, характер обробки відеоданих, продуктивність системи та можливість її подальшого масштабування.

Для визначення оптимальної архітектури програмного засобу було розглянуто три концептуальні підходи: монолітна архітектура, модульна (шарова) архітектура та мікросервісна архітектура. Кожен підхід має свої переваги та обмеження, які оцінено з урахуванням вимог, сформульованих у підрозділі 2.1.

Монолітна архітектура. Монолітний підхід передбачає створення єдиного застосунку, в якому функціональні компоненти тісно пов'язані між собою та функціонують у межах спільного середовища виконання.

Переваги монолітної архітектури:

- проста початкова реалізація та швидкий старт розробки;
- відсутність складності з міжпроцесною взаємодією та мережевими протоколами;
- спрощене тестування та налагодження на ранніх етапах;
- єдина точка розгортання і мінімальні вимоги до інфраструктури.

Недоліки монолітної архітектури:

- низька масштабованість: складність розширення функціоналу без зміни всієї системи;
- ризик деградації продуктивності при зростанні навантаження;
- слабка гнучкість для оновлення окремих компонентів (будь-яка зміна – впливає на весь застосунок);
- обмежена можливість паралельної роботи над різними модулями командою.

Модульна (шарова) архітектура. Модульна архітектура передбачає поділ системи на логічні модулі (шари), які виконують окремі функції і взаємодіють між собою через стандартизовані інтерфейси.

Типовий поділ для інтелектуальної відеосистеми включає: модуль отримання відеопотоку, модуль попередньої обробки, модуль *AI*-аналітики, модуль трекінгу, модуль подій і правил, модуль запису та архіву, *UI*-модуль

Переваги модульної архітектури:

- можливість незалежної розробки й удосконалення модулів;
- спрощене тестування окремих компонентів;
- гнучкість у додаванні нових функцій (наприклад, заміна *YOLO* на іншу модель без зміни *UI*);
- підтримка масштабування шляхом оптимізації окремих модулів або їх переведення в окремі процеси/потоки.

Недоліки модульної архітектури:

- потребує більш ретельного проєктування інтерфейсів взаємодії між модулями;
- може ускладнити підтримку початківцям без чіткої документації.

Мікросервісна архітектура. Мікросервісний підхід передбачає розподіл функціоналу на незалежні сервіси, що взаємодіють через мережеві протоколи. У контексті відеоаналітики можливе розділення на сервіси: відеозахоплення, *AI*-аналітика, збереження, *UI*, статистика, нотифікації тощо.

Переваги мікросервісної архітектури:

- максимальна масштабованість та гнучкість;
- можливість розподіленого розгортання на кількох вузлах;
- оновлення окремих сервісів без зупинки системи;
- підходить для *Smart City* та великих систем (100+ камер).

Недоліки мікросервісної архітектури:

- висока складність розробки, розгортання та підтримки;
- необхідність використання *DevOps*-інфраструктури (*Docker*, *Kubernetes*, *Kafka*);

– значні вимоги до продуктивності мережі при передачі відеопотоків;

– надлишкова для дипломного проєкту або для систем малої/середньої потужності.

Обґрунтування вибору архітектури. З огляду на цілі проєкту, вимоги до продуктивності, необхідність гнучкого розширення функціоналу та можливість розвитку системи у майбутньому, для реалізації програмного засобу обрана модульна (шарова) архітектура.

Цей підхід дозволяє:

– незалежно розробляти та тестувати компоненти системи (*AI*, трекінг, події, *UI*);

– змінювати або оновлювати окремі модулі без впливу на всю систему;

– легко інтегрувати нові моделі нейронних мереж та алгоритми;

– підготувати фундамент для можливого наступного переходу до мікросервісної архітектури, якщо система буде масштабована до корпоративного рівня.

2.3. Архітектура програмного засобу

Запропонована модульна архітектура забезпечує гнучкість системи та спрощує її подальше розширення або модифікацію. Завдяки чіткому розмежуванню функціональних шарів кожен компонент може розвиватися незалежно від інших, що зменшує складність супроводу програмного коду та підвищує надійність системи в цілому.

Використання шарової моделі також дозволяє адаптувати систему до різних умов експлуатації, зокрема змінювати джерела відео, алгоритми детекції або методи зберігання даних без необхідності повної переробки програмного засобу.

Такий підхід є особливо важливим для систем реального часу, де стабільність роботи та можливість масштабування відіграють ключову роль.

Крім того, асинхронна взаємодія між модулями через черги та сигнали дозволяє ефективно використовувати обчислювальні ресурси та забезпечує стійку роботу системи навіть за нерівномірного навантаження.

Це робить архітектуру придатною як для локальних рішень, так і для подальшого розвитку у напрямку більш складних інтелектуальних систем відеоспостереження.

Логічні шари:

- шар доступу до відео (*Capture I/O*): захоплення кадрів з *USB*/*UVC*-камер та *RTSP*-потоків, керування реконектом, буферизація;
- шар попередньої обробки (*Preprocessing*/*DSP*): нормалізація, масштабування до розміру інференсу, шумозниження, перетворення колірних просторів;
- шар інтелектуальної аналітики (*AI Inference*): інференс *Ultralytics YOLO*, пост-процесинг (*NMS*, фільтри *conf*/*IoU*), вибір класів;
- шар трекінгу (*Tracking*): *DeepSORT*/*SORT*: призначення *ID*, асоціація між кадрами, базові атрибути траєкторій;
- шар подій і правил (*Events*/*Rules*): виявлення сценаріїв: перетин лінії, вторгнення в полігон, “зупинка/залишений предмет” (мінімальний набір), генерація сповіщень;
- шар зберігання (*Storage*): відеоархів (*FFmpeg* контейнер *MP4*/*MKV*), *SQLite* для конфігів/журналу подій;
- шар представлення (*UI*): *PySide6*/*PyQt*: сітка прев’ю, оверлеї (*bbox*/*ID*/зони), панелі параметрів, журнал подій, таймлайн архіву.

Компоненти зв’язуються через черги/сигнали (*Qt signals*/*slots* або *thread-safe* черги), щоб не блокувати *UI* і забезпечити незалежність *FPS* інференсу від швидкодії відображення.

Опис основних модулів:

VideoSourceManager – керує списком джерел; відкриття/закриття стрімів; реконект з *backoff*; моніторинг станів. *FrameGrabber* – потіковий читач кадрів (*OpenCV VideoCapture*/*GStreamer*); контроль часових міток; *drop-policy* при відставанні.

Detector – обгортка над *Ultralytics*: завантаження ваг, інференс, *NMS*, фільтрація класів, регулювання частоти детекції (кожен *N*-й кадр).

Tracker – DeepSORT/*SORT*: асоціація, підтримка *ID*, облік зникнень/повторних поява у вікні *T*.

EventEngine – обчислення подій на основі *bbox*/*ID*: тест на перетин сегмента, потрапляння в полігон, часові тригери; публікація нотифікацій.

StorageService – запис відео (безперервно або подійно), ротація, індексація; *SQLite*: конфіги, профілі, журнал подій (*timestamp*, *cam_id*, *event_type*, *attrs*).

UI (*Qt Frontend*) – Головне вікно (сітка прев'ю), діалоги додавання камер, редактор зон/ліній, панель *AI*-параметрів, журнал, перегляд архіву; накладання графіки.

Оптимізація: детекція через *N* кадрів + міжкадровий трекінг, зменшення навантаження на *GPU*/*CPU* без втрати безперервності траєкторій.

Модель взаємодії модулів (потоки даних)

Такий конвеєр розв'язує ключову задачу: розвести частоти (кадрування, детекція, трекінг, рендеринг, запис) і не допустити взаємного блокування.

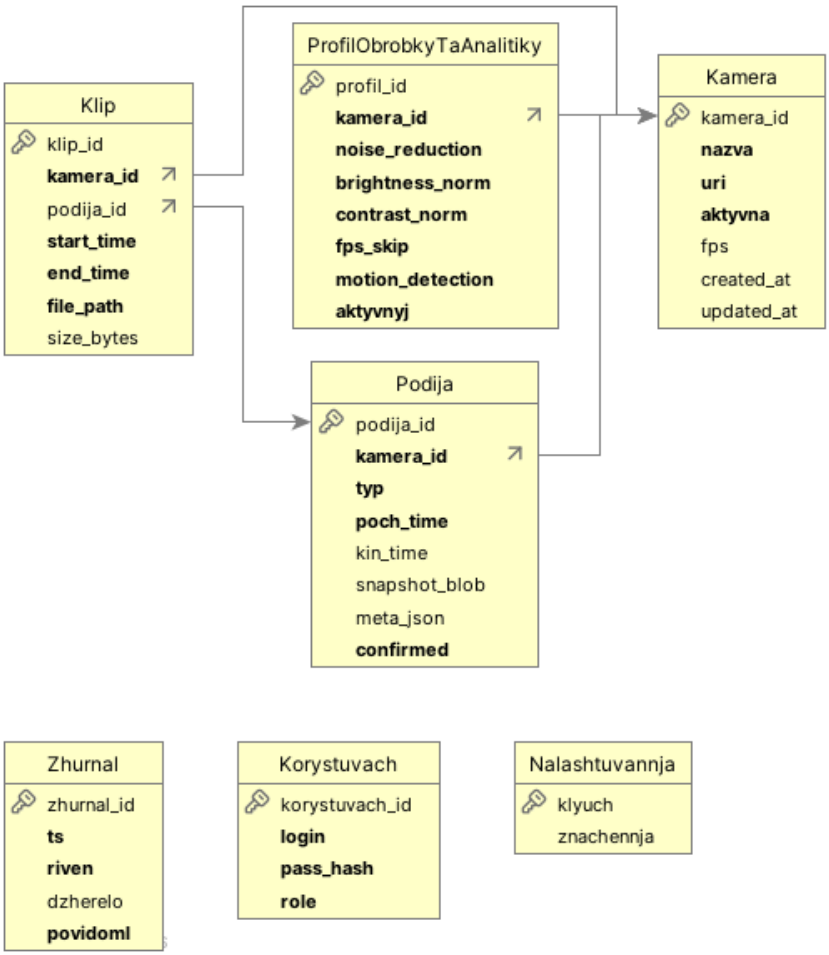

Рис. 2.1. *ER* діаграма бази даних

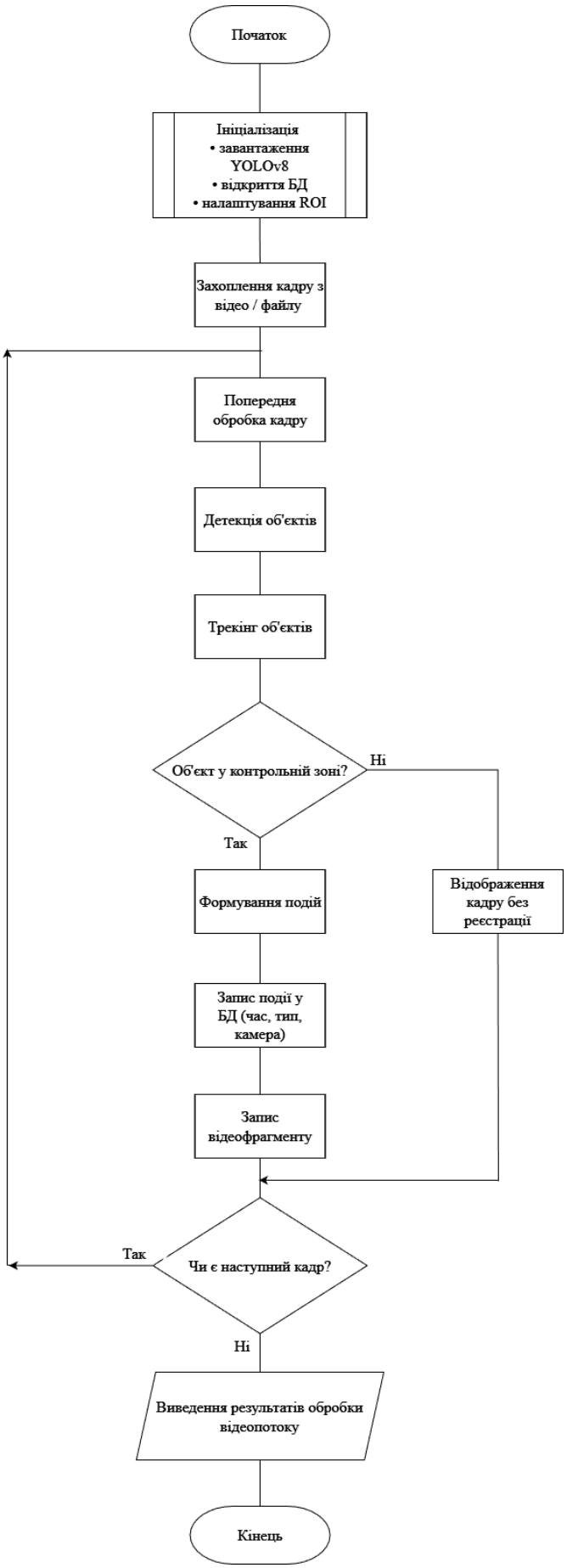

Рис. 2.2. Схема алгоритму (програмний модуль) інтелектуальної системи відеоспостереження

Запропонована архітектура програмного засобу побудована за модульним принципом, що забезпечує гнучкість, масштабованість та спрощення процесу супроводу системи. Кожен модуль відповідає за окрему функціональну частину та може бути змінений або вдосконалений без суттєвого впливу на роботу інших компонентів.

Модуль захоплення відеопотоку забезпечує отримання кадрів як з відеофайлів, так і з камер реального часу. Його основне завдання полягає у стабільній передачі кадрів для подальшої обробки незалежно від джерела сигналу.

Модуль попередньої обробки виконує масштабування, нормалізацію та інші перетворення зображення, необхідні для підвищення точності роботи нейронної мережі. Такий підхід дозволяє зменшити обчислювальне навантаження та підвищити швидкість обробки відео в реальному часі.

2.4. Проєктування бази даних програмного засобу

Для ефективної роботи програмного засобу, що обробляє відеосигнал у режимі реального часу та виконує елементи відеоаналітики, важливо правильно організувати зберігання даних. Система має працювати локально, без складної інфраструктури, та забезпечувати стабільність під час обробки відеопотоку з однієї або двох камер. За таких умов оптимальним рішенням є використання реляційної файлової СУБД *SQLite*[12]. *SQLite* поєднує простоту розгортання та достатню функціональність: БД зберігається у вигляді одного файлу, що значно спрощує встановлення та супровід програмного забезпечення. СУБД підтримує транзакційність, механізми забезпечення цілісності даних, зовнішні ключі та індексацію – цього повністю достатньо для системи даного масштабу. Завдяки відсутності серверної частини зменшується споживання ресурсів, що особливо важливо для застосунків, які паралельно обробляють відеопотоки. Крім того, використання *SQLite* дозволяє легко інтегрувати механізми журналювання та ведення історії подій без додаткових витрат на адміністрування бази даних. Це спрощує налагодження програмного засобу, аналіз помилок та перевірку коректності роботи системи під час тестування і експлуатації. Локальний характер зберігання даних також підвищує автономність рішення та

зменшує залежність від мережевих з’єднань.У системі зберігаються насамперед структуровані дані: параметри камер, налаштування обробки сигналу та аналітики, інформація про зафіксовані події та збережені відеофрагменти. З огляду на це, реляційний підхід виглядає природним та логічним рішенням.

Водночас, щоб не перевантажувати базу даних великими мультимедійними файлами, відео та зображення пропонується зберігати у файловій системі, а в БД фіксувати до них посилання. Такий підхід пришвидшує роботу системи й значно полегшує резервне копіювання.

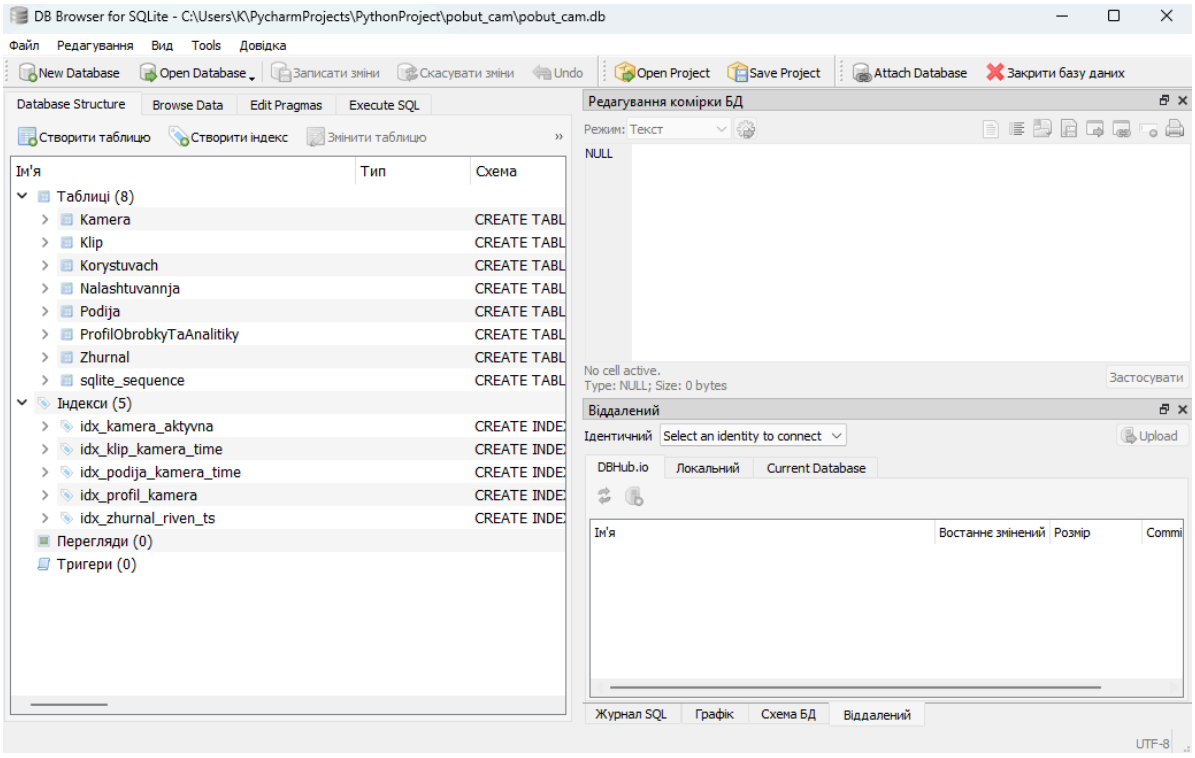

Рис. 2.3. Візуалізація бази даних в системі «*PobutCam*» у *DB Browser for SQLite*

Логічна модель даних визначає перелік сутностей, взаємозв’язки між ними та інформацію, яку необхідно зберігати для повноцінної роботи системи. Враховуючи специфіку проєкту – а саме обробку відеосигналу та виконання елементарної відеоаналітики – модель охоплює дві основні групи даних: Дані попередньої обробки відео (*DSP*): Зберігаються параметри, що впливають на якість і швидкість аналізу відеопотоку: активність шумозниження, нормалізація яскравості та контрасту, режим

пропуску кадрів (*FPS Skip*), а також вказівка, чи використовується детекція руху як тригер.

Дані відеоаналітики та подій: До них належать записи про виявлені події, короткі описи ситуацій, метадані, знімки кадру та посилання на відеофрагменти, збережені під час спрацювання аналітики.

Основні сутності, які складають логічну модель БД:

Kamera – містить ключову інформацію про джерела відеосигналу та параметри їх підключення.

Таблиця *Kamera* пов'язана з іншими сутностями: профілями обробки (*ProfilObrobkyTaAnalitiky*), подіями (*Podija*) та відеокліпами (*Klip*). Один запис у *Kamera* може мати багато пов'язаних записів у цих таблицях.

Таблиця 2.1

Опис таблиці *Kamera*

Поле	Тип	Опис
kamera_id	*INTEGER*	Унікальний ідентифікатор камери
nazva	*TEXT*	Назва або опис камери
created_at	*DATETIME*	Дата створення запису
uri	*TEXT*	Ідентифікатор джерела відео: номер вебкамери ("0"), *RTSP-URL* або шлях до відеофайлу
aktyvna	*INTEGER*	Прапорець активності камери (1 – активна, 0 – вимкнена)
fps	*REAL*	Оціночна частота кадрів для даного джерела (може бути *NULL*)

ProfilObrobkyTaAnalitiky – параметри алгоритмів обробки та відеоаналітики для конкретної камери. Для однієї камери може існувати кілька профілів, але при роботі використовується лише активний профіль.

Таблиця 2.2

Опис таблиці *ProfilObrobkyTaAnalitiky*

Поле	Тип	Опис
profil_id	*INTEGER*	Унікальний ідентифікатор профілю.
kamera_id	*INTEGER*	Посилання на камеру, для якої діє профіль (*Kamera.kamera_id*)
noise_reduction	*INTEGER*	Рівень або прапорець шумозаглушення під час обробки
brightness_norm	*INTEGER*	Рівень нормалізації яскравості
contrast_norm	*INTEGER*	Рівень нормалізації контрасту
fps_skip	*INTEGER*	Кожний *n*-й кадр, який обробляється (1 – усі кадри, 2 – кожний другий, тощо)
motion_detection	*INTEGER*	Прапорець увімкнення аналітики (детекції руху/об’єктів)
aktyvnyj	*INTEGER*	Позначка активності профілю для поточної камери

Podija – подія, що зафіксована системою внаслідок виявлення руху або іншої активності. Одна камера може мати багато подій. Таблиця *Podija* логічно описує інциденти, що відбуваються у зоні спостереження. *profil_id* має велике значення для цієї бази даних.

Таблиця «*Klip*» призначена для зберігання метаданих відеофрагментів, які були збережені системою відеоспостереження в результаті безперервного запису або фіксації подій відеоаналітики.

Таблиця 2.3

Опис таблиці *Podija*

Поле	Тип	Опис
profil_id	*INTEGER*	Унікальний ідентифікатор профілю.
kamera_id	*INTEGER*	Посилання на камеру, для якої діє профіль (*Kamera.kamera_id*)
noise_reduction	*INTEGER*	Рівень або прапорець шумозаглушення під час обробки
brightness_norm	*INTEGER*	Рівень нормалізації яскравості
contrast_norm	*INTEGER*	Рівень нормалізації контрасту
fps_skip	*INTEGER*	Кожний *n*-й кадр, який обробляється (1 – усі кадри, 2 – кожний другий, тощо)
motion_detection	*INTEGER*	Прапорець увімкнення аналітики (детекції руху/об’єктів)
aktyvnyj	*INTEGER*	Позначка активності профілю для поточної камери

Кожен запис таблиці відповідає окремому відеокліпу і містить унікальний ідентифікатор, що використовується для однозначного звернення до конкретного фрагмента. Також у таблиці зберігається ідентифікатор камери, з якої було отримано відео, що дозволяє пов’язати кліп з відповідним джерелом відеопотоку та його налаштуваннями.

Важливими атрибутами таблиці є часові мітки початку та завершення запису кліпу. Вони дають змогу точно визначити часовий інтервал, який охоплює відеофрагмент, а також забезпечують коректну синхронізацію з подіями, зафіксованими в системі. На основі цих даних реалізується фільтрація та пошук кліпів за датою і часом.

Таблиця 2.4

Опис таблиці *Klip*

Поле	Тип	Опис
klip_id	*INTEGER*	Унікальний ідентифікатор відеокліпу.
kamera_id	*INTEGER*	Посилання на камеру, з якої записано кліп (*Kamera.kamera_id*).
podija_id	*INTEGER*	Посилання на подію, з якою пов'язаний кліп (*Podija.podija_id*), може бути *NULL.*
start_time	*DATETIME*	Час початку запису кліпу.
end_time	*DATETIME*	Час завершення запису кліпу.
file_path	*TEXT*	Шлях до відеофайлу на диску.
size_bytes	*INTEGER*	Розмір файлу у байтах (використовується для контролю обсягу архіву).

Zhurnal – службові повідомлення та записи про роботу системи. Таблиця *Zhurnal* дозволяє відслідковувати історію роботи системи, діагностувати помилки та аналізувати послідовність подій.

Таблиця 2.5

Опис таблиці *Zhurnal*

Поле	Тип	Опис
zhurnal_id	*INTEGER*	Унікальний ідентифікатор запису журналу.
ts	*DATETIME*	Час створення запису (за замовчуванням – поточний час).
riven	*TEXT*	Рівень повідомлення (*INFO*, *ERROR*, *WARNING* тощо).
dzherelo	*TEXT*	Джерело або модуль, який сформував повідомлення.
povidoml	*TEXT*	Текст повідомлення.

Korystuvach – інформація про користувача системи (у даному випадку – одного).

Таблиця 2.6

Опис таблиці *Korystuvach*

Поле	Тип	Опис
korystuvach_id	*INTEGER*	Унікальний ідентифікатор користувача.
login	*TEXT*	Логін користувача, використовується для авторизації.
pass_hash	*TEXT*	Хеш паролю (*SHA-256*), пароль у відкритому вигляді не зберігається.
role	*TEXT*	Роль користувача (наприклад, "*admin*", "*single_user*").

Nalashtuvannja – глобальні параметри роботи програмного засобу. Така структура дозволяє гнучко додавати нові параметри без зміни схеми БД. Наприклад, *ROI* для певної камери зберігається як рядок "*x*1,*y*1,*x*2,*y*2" із ключем виду *roi_<kamera_id>*.

Таблиця 2.7

Опис таблиці *Nalashtuvannja*

Поле	Тип	Опис
klyuch	*TEXT*	Ім’я налаштування (наприклад, "*conf_thres*", "*roi_1*", "*archive_max_gb*").
znachennja	*TEXT*	Значення налаштування у текстовому вигляді (з подальшим перетворенням у потрібний тип у коді).

Розроблена логічна структура забезпечує зручне зберігання та обробку даних, дозволяє швидко знаходити потрібну інформацію та підтримує масштабування системи, якщо в майбутньому буде доцільно додати роботу з більшою кількістю камер або користувачів.

ER-модель(опис) допомагає наочно представити структуру даних та взаємозв'язки між сутностями, необхідними для роботи програмного засобу. У центрі моделі знаходиться сутність *Kamera*, оскільки саме вона є джерелом відеопотоку, що проходить подальшу обробку та аналіз.

До однієї камери може бути прив'язано кілька профілів обробки та аналітики (*ProfilObrobkyTaAnalitiky*). Це створює можливість використовувати різні сценарії роботи – наприклад, один профіль для денного режиму, інший для нічного або економного (коли потрібно зменшити навантаження на систему).

Сутність *Podija* зберігає дані про моменти, коли система зафіксувала активність у кадрі. Кожна подія має часові мітки початку і завершення, може містити знімок кадру та метадані для уточнення характеру виявленої активності. Якщо подія супроводжувалася записом відео, інформація про відповідний фрагмент зберігається в сутності *Klip*. Замість зберігання самого відео в БД фіксується шлях до файлу – це дає змогу не перевантажувати базу та спрощує резервування даних. Окрім цього, у системі передбачені:

- *Nalashtuvannja* – для зберігання загальних параметрів ПЗ у форматі "ключ–значення" (наприклад, мова інтерфейсу чи тема оформлення);
- *Korystuvach* – містить дані про користувача, який працює з системою;
- *Zhurnal* – виконує роль технічного журналу подій, де фіксуються повідомлення, попередження або помилки.

Завдяки такій моделі даних забезпечується впорядкованість інформації та зберігається цілісність зв'язків між об'єктами системи, що спрощує її реалізацію та подальший розвиток.

Файл бази даних: Система використовує вбудовану реляційну СУБД *SQLite*. Усі таблиці зберігаються в одному файлі:

- Назва файлу: *pobut_cam.db.*
- Формат: *SQLite.*

Цей файл містить як структуру (схему) бази даних, так і всі дані: користувачів, камери, профілі обробки, події, відеокліпи, налаштування та журнал.

Відеофайли з кліпами: Відеофрагменти, пов'язані з подіями, зберігаються у каталозі *recordings/* у корені проєкту. Основні характеристики: каталог: *recordings/*;

Формат контейнера: *MP*4; Кодек: *mp*4*v* (створюється за допомогою *OpenCV VideoWriter*); Приклад імені файлу: *kamera1_event*_20251216_201000.*mp*4. Шлях до кожного відеофайлу та його розмір фіксуються в таблиці *Klip*.

Файли експорту (*CSV*)

Для побудови звітності та аналізу подій система підтримує експорт частини даних у форматі *CSV*. Наприклад, файл *events_export.csv* може містити список усіх подій (*Podija*) з додатковою інформацією про камеру та тип події. Формат *CSV* є текстовим і легко відкривається у табличних редакторах (*Excel*, *LibreOffice Calc*).

Поняття «схема» бази даних. У контексті реляційних баз даних термін «схема» (*schema*) має два основні аспекти:

– логічна схема – опис структури даних на рівні сутностей та зв’язків.

– фізична (*DDL*) схема – конкретні *SQL* оператори, які створюють таблиці, індекси та обмеження.

Логічна схема бази даних системи «*PobutCam*» включає такі основні сутності:

– *Korystuvach* – користувачі системи та їхні ролі.

– *Kamera* – джерела відеосигналу.

– *ProfilObrobkyTaAnalitiky* – параметри попередньої обробки та аналітики для кожної камери.

– *Podija* – зареєстровані події (наприклад, вторгнення в зону спостереження).

– *Klip* – відеофрагменти, записані під час подій.

– *Nalashtuvannja* – глобальні налаштування системи у форматі «ключ–значення».

– *Zhurnal* – технічний журнал (лог) роботи системи.

Між таблицями визначені зв’язки типу «один-до-багатьох»: одна камера може мати багато подій та відеокліпів, одна подія може бути пов’язана з кількома кліпами тощо.

Фізична схема бази даних реалізована у вигляді набору операторів *CREATE TABLE* та *CREATE INDEX* для СУБД *SQLite*.

Вони об’єднані в *SQL*-скрипт (константа *SCHEMA_SQL* у вихідному коді), який виконується під час ініціалізації системи. У результаті створюється файл

pobut_cam.db із необхідною структурою та службовими записами за замовчуванням (адміністратор, камера за замовчуванням, базові налаштування аналітики).

Таким чином, під «схемою бази даних» у дипломному проєкті розуміється як логічний опис сутностей та їхніх зв'язків, так і конкретна реалізація цієї структури у вигляді *SQL*-операторів для вбудованої СУБД *SQLite*.

Нормалізація структури бази даних

Створюючи структуру бази даних, важливо було забезпечити не лише зручність у зберіганні та обробці даних, а й їхню цілісність, відсутність дублювання та логічну організованість. Тому модель БД приведена до перших трьох нормальних форм – 1НФ, 2НФ та 3НФ.

Перша нормальна форма (1НФ): Усі поля таблиць містять атомарні значення, що не потребують додаткового розбиття. Допоміжні дані, такі як метадані подій чи додаткова інформація про фрейми, зберігаються у форматі *JSON* (*meta_json*, *snapshot_blob*), що дозволяє структурувати дані без порушення атомарності.

Друга нормальна форма (2НФ): Усі неключові атрибути залежать від повного первинного ключа. Первинні ключі реалізовано у вигляді автонумерації (**_id*), тому часткова залежність відсутня.

Третя нормальна форма (3НФ): Модель не містить транзитивних залежностей. Параметри, що стосуються обробки сигналу та аналітики, винесено в окрему таблицю *ProfilObrobkyTaAnalitiky*, що не лише зменшує дублювання, а й дає змогу змінювати профілі для різних сценаріїв роботи з камерою.

Таким чином, нормалізація зробила базу даних логічною, простою в супроводі та водночас достатньо гнучкою для майбутнього розширення.

SQL-скрипт створення структури бази даних (*UA-TRANS*). Нижче наведено *SQL*-скрипт для створення основних таблиць у *SQLite*.

Він враховує локальний характер використання системи, мінімальне навантаження на ресурси та потребу швидкого доступу до даних.

```
CREATE TABLE IF NOT EXISTS Korystuvach (
    korystuvach_id   INTEGER PRIMARY KEY AUTOINCREMENT,
    login            TEXT NOT NULL UNIQUE,
```

```
    pass_hash       TEXT NOT NULL,
    role            TEXT NOT NULL DEFAULT 'single_user'
);

CREATE TABLE IF NOT EXISTS Kamera (
    kamera_id    INTEGER PRIMARY KEY AUTOINCREMENT,
    nazva        TEXT NOT NULL,
    uri          TEXT NOT NULL,
    aktyvna      INTEGER NOT NULL DEFAULT 1,
    fps          REAL,
    created_at   DATETIME DEFAULT CURRENT_TIMESTAMP,
    updated_at   DATETIME
);

CREATE INDEX IF NOT EXISTS idx_kamera_aktyvna ON Kamera(aktyvna);

CREATE TABLE IF NOT EXISTS ProfilObrobkyTaAnalitiky (
    profil_id          INTEGER PRIMARY KEY AUTOINCREMENT,
    kamera_id          INTEGER NOT NULL,
    noise_reduction    INTEGER NOT NULL DEFAULT 0,
    brightness_norm    INTEGER NOT NULL DEFAULT 0,
    contrast_norm      INTEGER NOT NULL DEFAULT 0,
    fps_skip           INTEGER NOT NULL DEFAULT 1,
    motion_detection   INTEGER NOT NULL DEFAULT 1,
    aktyvnyj           INTEGER NOT NULL DEFAULT 1,
    FOREIGN KEY (kamera_id) REFERENCES Kamera(kamera_id) ON DELETE
CASCADE
);

CREATE INDEX IF NOT EXISTS idx_profil_kamera ON
ProfilObrobkyTaAnalitiky(kamera_id);
```

```
CREATE TABLE IF NOT EXISTS Podija (
    podija_id      INTEGER PRIMARY KEY AUTOINCREMENT,
    kamera_id      INTEGER NOT NULL,
    typ            TEXT NOT NULL,
    poch_time      DATETIME NOT NULL,
    kin_time       DATETIME,
    snapshot_blob  BLOB,
    meta_json      TEXT,
    confirmed      INTEGER NOT NULL DEFAULT 0,
    FOREIGN KEY (kamera_id) REFERENCES Kamera(kamera_id) ON DELETE
CASCADE
);

CREATE INDEX IF NOT EXISTS idx_podija_kamera_time ON Podija(kamera_id,
poch_time);

CREATE TABLE IF NOT EXISTS Klip (
    klip_id     INTEGER PRIMARY KEY AUTOINCREMENT,
    kamera_id   INTEGER NOT NULL,
    podija_id   INTEGER,
    start_time  DATETIME NOT NULL,
    end_time    DATETIME NOT NULL,
    file_path   TEXT NOT NULL,
    size_bytes  INTEGER,
    FOREIGN KEY (kamera_id) REFERENCES Kamera(kamera_id) ON DELETE
CASCADE,
    FOREIGN KEY (podija_id) REFERENCES Podija(podija_id) ON DELETE SET
NULL
);
```

CREATE INDEX IF NOT EXISTS idx_klip_kamera_time ON Klip(kamera_id, start_time);

CREATE TABLE IF NOT EXISTS Nalashtuvannja (
klyuch TEXT PRIMARY KEY,
znachennja TEXT
);

CREATE TABLE IF NOT EXISTS Zhurnal (
zhurnal_id INTEGER PRIMARY KEY AUTOINCREMENT,
ts DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
riven TEXT NOT NULL,
dzherelo TEXT,
povidoml TEXT NOT NULL
);

CREATE INDEX IF NOT EXISTS idx_zhurnal_riven_ts ON Zhurnal(riven, ts);

Особливості реалізації та обмеження. З огляду на локальний характер використання програмного засобу, структура бази даних була спроєктована легкою та невибагливою до ресурсів. Основні особливості реалізації включають:

– локальний однокористувацький режим, що спрощує модель доступу та безпеки;
– зберігання відеофайлів у файловій системі, а в БД – лише посилань на них, що зменшує обсяг бази та пришвидшує роботу;
– використання *JSON*-полів дозволяє зберігати допоміжні дані без створення окремих таблиць;
– можливість створення кількох профілів обробки та аналітики дає користувачу гнучкість у налаштуванні роботи системи без зміни конфігурації камери.

До недоліків можна віднести обмежені можливості *SQLite* щодо паралельного доступу, однак для однокористувацької системи з одним-двома джерелами відеосигналу це не створює проблем.

2.5. Проєктування користувацького інтерфейсу програмного засобу

Продуманий інтерфейс є важливою складовою програмного засобу, оскільки саме через нього користувач взаємодіє з системою обробки відеосигналу та відеоаналітики. Основним завданням при створенні інтерфейсу було забезпечити зручність, простоту і швидкість доступу до базових функцій, зважаючи на те, що ПЗ призначене для використання однією особою та не передбачає складної багаторівневої навігації.

Інтерфейс розроблявся таким чином, щоб користувач міг швидко отримати доступ до головних можливостей: перегляду відео в реальному часі, перемикання між підключеними камерами, зміни налаштувань обробки та аналітики, а також перегляду збережених відеофрагментів. Така побудова мінімізує кількість дій для виконання поширених сценаріїв роботи та зменшує час адаптації до системи.

Основні принципи *UX*-проєктування. Дотримання зазначених принципів *UX*-проєктування є особливо важливим для систем відеоспостереження, які використовуються у повсякденній діяльності та часто експлуатуються без спеціальної підготовки персоналу. Інтерфейс повинен мінімізувати когнітивне навантаження на користувача та дозволяти зосередитися на основному завданні — спостереженні та аналізі відеопотоку.

У процесі проєктування враховувались типові сценарії використання системи, такі як швидка перевірка стану камери, перегляд подій або оперативна зміна параметрів обробки сигналу. Це дозволило сформувати інтерфейс, який залишається зрозумілим навіть за тривалої роботи та не потребує постійного звернення до довідкових матеріалів.

Окрему увагу було приділено узгодженості елементів інтерфейсу: однакові дії виконуються однаковим способом на всіх екранах, а візуальні елементи мають стабільне розташування. Такий підхід підвищує швидкість адаптації користувача та зменшує ймовірність помилкових дій під час роботи з програмним засобом.

При проєктуванні інтерфейсу орієнтувалися на принципи, які роблять взаємодію з ПЗ зрозумілою і комфортною навіть для користувача без технічного досвіду:

– простота і мінімалізм: тільки потрібні елементи на екрані, без перевантаження кнопками та меню;
– логічна та передбачувана навігація: користувачу має бути інтуїтивно зрозуміло, де шукати налаштування, де переглядати записи і як повернутися до головного вікна;
– швидкий доступ до основних дій: ключові функції повинні виконуватися в межах 1–3 кліків;
– взаємодія в реальному часі: зміна параметрів обробки або аналітики не має вимагати перезапуску системи – оновлення застосовуються одразу;
– зручність при перегляді відео: інтерфейс не повинен відволікати від основної задачі – спостереження за відеопотоком.

Структура інтерфейсу користувача. Інтерфейс побудований на основі кількох взаємопов'язаних екранів:

– головне вікно (*Live*-режим): відображає відео з камери, містить кнопки для перемикання між камерами, відкриття налаштувань та переходу до списку збережених фрагментів;
– налаштування камери: містить параметри камери – назву, *URI*, активність і базові налаштування;
– профіль обробки та аналітики: дозволяє увімкнути або вимкнути шумозниження, нормалізацію яскравості та контрасту, встановити *FPS Skip* і керувати детекцією руху;
– список збережених відеофрагментів (спрощений архів): Відображає останні збереження у вигляді переліку кліпів з можливістю їх перегляду та експорту у файл.

Такий підхід до структури інтерфейсу забезпечує швидкий доступ до всіх ключових функцій, не створюючи зайвого навантаження на користувача.

Опис основних екранів інтерфейсу. Головне вікно призначене для перегляду відеопотоку в реальному часі. У верхній частині екрана розташовано панель керування з такими елементами:

– перемикач активної камери;

– кнопка переходу у повноекранний режим;

– кнопка відкриття налаштувань;

– кнопка переходу до списку збережених відеофрагментів.

Основна частина вікна відведена під відеопотік. Будь-які повідомлення системи відображаються у вигляді коротких інформаційних вікон у верхній частині екрана.

Налаштування.камери. Дозволяють користувачу змінювати основні параметри камери. Передбачено валідацію введених даних: у разі некоректного *URI* або недоступності відеопотоку система інформує користувача про помилку.

Профіль обробки та аналітики. Екран дозволяє налаштовувати параметри попередньої обробки сигналу (*DSP*) та відеоаналітики для кожної камери окремо. Користувач може увімкнути або вимкнути шумозниження, нормалізацію яскравості, контрасту, а також встановити параметр пропуску кадрів (*FPS Skip*). Детекція руху може бути активована чи деактивована за потреби.

Список збережених відеофрагментів. Відображає перелік останніх кліпів у вигляді списку. Кожен запис містить інформацію про камеру, дату та часовий інтервал збереження. Передбачено можливість відтворення відеофрагмента та його експорту у файл.

Навігація та структура меню. Головне вікно. Це центральний елемент взаємодії користувача з ПЗ. Основну частину вікна займає відеопотік. У верхній частині розміщено панель керування з кнопками для перемикання камер, відкриття налаштувань та переходу до збережених відеофрагментів.

Повідомлення про події або помилки відображаються короткими інформативними сповіщеннями, що не заважають перегляду відео.

Налаштування камери. Дозволяють змінювати параметри підключення, назву камери та її статус. Передбачена перевірка введених значень, щоб уникнути некоректних підключень або помилок при введенні *URI*.

Профіль обробки та аналітики. Дає змогу керувати параметрами попередньої обробки відео та активацією відеоаналітики. Користувач може швидко налаштувати профіль під конкретні умови – наприклад, зменшити навантаження на систему за допомогою *FPS Skip* або вимкнути шумозниження при достатньому освітленні.

Список збережених відеофрагментів. Містить перелік останніх кліпів, відсортованих за часом збереження. На цьому екрані можна переглянути відео та за потреби експортувати його у файл для подальшого використання.

Вимоги до інтерфейсу:

- Інтерфейс має бути адаптованим для відображення на екранах з мінімальною роздільною здатністю 1280×720 пікселів.
- Усі елементи керування повинні бути доступними не більш ніж за три кліки.
- Оновлення параметрів обробки та аналітики повинно застосовуватись без перезапуску програми.
- Інтерфейс має бути стабільним при зміні активної камери.

План юзабіліті-тестування

Для оцінки зручності користування інтерфейсом системи пропонується провести юзабіліті-тестування за участю 2, 3 користувачів, не знайомих із роботою програмного засобу. Кожен учасник має виконати такі завдання:

- запустити програмний засіб та переглянути відеопотік;
- перемкнутися між відеокамерами;
- змінити один із параметрів профілю обробки та аналітики;
- переглянути список збережених відеофрагментів та відтворити один із них.

Критеріями успішності тестування є:

- виконання завдань без помилок;
- максимальний час виконання кожного завдання – 2 хвилини;

2.6. Висновки до розділу

У другому розділі дипломної роботи було розроблено архітектуру програмного модуля інтелектуальної системи відеоспостереження, яка забезпечує поетапну обробку відеопотоку та логічне розділення функціональних компонентів. Запропонована архітектура включає модулі захоплення відео, попередньої обробки кадрів, детекції об’єктів, аналізу подій та збереження результатів, що дозволяє

забезпечити зрозумілу структуру системи та спростити її подальше розширення або модифікацію.

Також у межах розділу було реалізовано програмний модуль обробки відеопотоку з використанням методів машинного навчання. Для детекції об’єктів застосовано нейронну мережу, яка працює у режимі реального часу та дозволяє автоматично виявляти об’єкти у кадрі. Реалізація модуля підтвердила можливість ефективного використання сучасних методів машинного навчання для задач відеоаналізу без застосування складних апаратних рішень. Отримані результати свідчать про доцільність обраної архітектури та ефективність реалізованого програмного модуля для використання в системах інтелектуального відеоспостереження.

Окрему увагу приділено проєктуванню інтерфейсу користувача.

Його побудовано за принципом простоти та швидкого доступу до основних функцій, що дозволяє мінімізувати час взаємодії з програмою. Завдяки цьому користувач може зосередитися на спостереженні за відео, не витрачаючи зусиль на пошук потрібних опцій. Запропоноване тестування юзабіліті допоможе перевірити інтуїтивність інтерфейсу та за потреби вдосконалити його.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1. Загальна структура програмної системи

Розроблений програмний засіб призначений для обробки відеосигналу в реальному часі з елементами інтелектуального аналізу. Його структура побудована так, щоб забезпечити стабільну роботу навіть на обмежених ресурсах, мінімізувати складність налаштування та надати користувачу інтуїтивний інтерфейс для взаємодії з системою.

Основна ідея побудови полягає у поєднанні модульного підходу з простотою інтеграції. Система поділена на кілька логічних компонентів, кожен з яких виконує власну функцію – від захоплення відео до аналітичної обробки та збереження результатів. Така архітектура забезпечує гнучкість, масштабованість і можливість подальшого розвитку системи без суттєвої зміни її основи.

Загальна структура програмного засобу включає такі ключові модулі:

Модуль захоплення відеосигналу. Відповідає за підключення до відеокамер, зчитування потокових даних та передачу їх на подальшу обробку.

Модуль попередньої обробки. Виконує базові операції з очищення зображення: шумозниження, нормалізацію яскравості та контрасту, а також корекцію кольору. Ці дії дозволяють підготувати відеопотік до подальшого аналізу та зменшити кількість помилкових спрацювань системи. Модуль збереження даних. Реалізований на базі *SQLite* і відповідає за фіксацію подій, зберігання параметрів роботи камер, профілів аналітики та журналу системних повідомлень. Відеофайли не записуються безпосередньо в базу, а зберігаються у файловій системі з прив’язкою до записів у БД.

Модуль інтерфейсу користувача (*UI*). Побудований на основі бібліотеки *PyQt*, забезпечує просту взаємодію користувача із системою: перегляд потокового відео, перемикання камер, налаштування параметрів обробки та аналітики, перегляд архівних записів.

Зв’язок між модулями здійснюється через внутрішні *API*-виклики, що дозволяє передавати відеокадри та дані у вигляді об’єктів, не перевантажуючи систему. Такий підхід спрощує налагодження та забезпечує стійкість до відмов окремих компонентів.

Програмний засіб реалізовано як настільний застосунок, що працює локально без необхідності постійного підключення до мережі. Це робить його придатним для використання у невеликих офісах, лабораторіях або домашніх умовах, де не потрібна централізована інфраструктура.

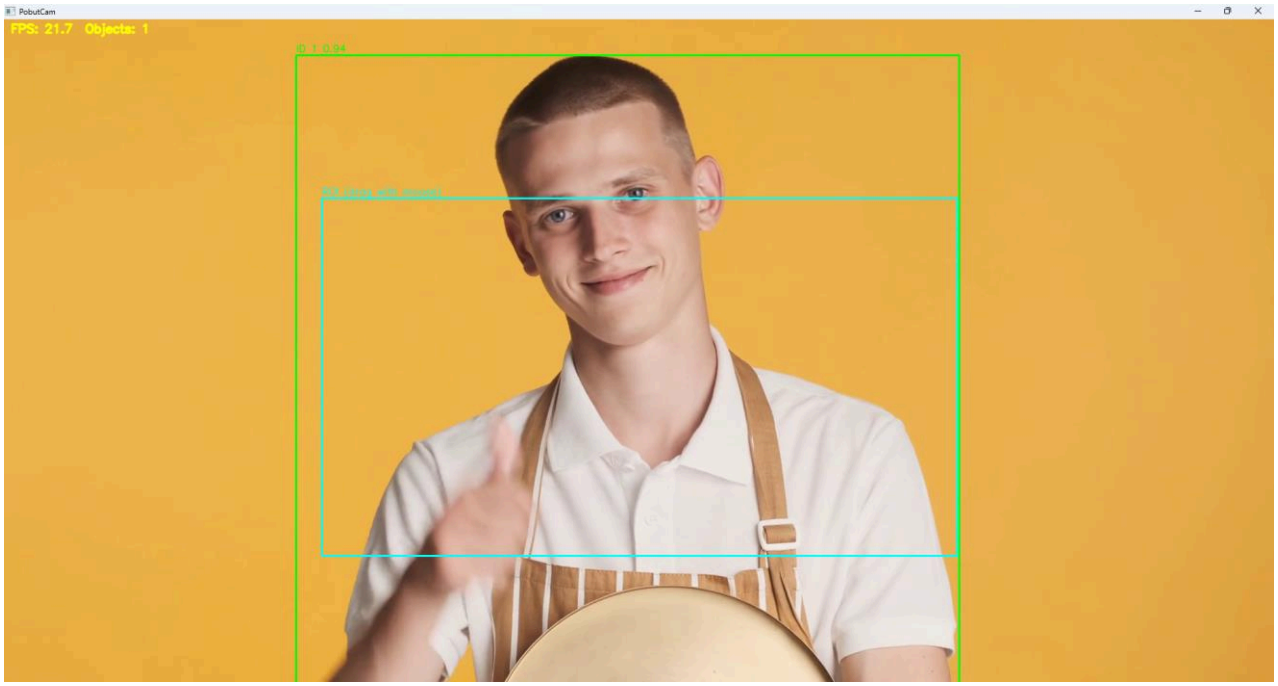

Рис. 3.1. Скріншот виконання програми

3.2. Використані технології та середовище розробки

Для створення програмного засобу було обрано стек технологій, який поєднує простоту реалізації, доступність, кросплатформеність і стабільну роботу в режимі реального часу. Основні вимоги до системи полягали у забезпеченні швидкої обробки

відеосигналу, мінімальному навантаженні на ресурси та легкому розгортанні на будь-якому персональному комп’ютері.

Основні технології:

– *Python* – головна мова програмування системи. Вона забезпечує швидку розробку, має розвинену екосистему бібліотек для роботи з відео, зображеннями, базами даних та графічним інтерфейсом[3]. Інтерфейс програми показано на рис 3.2.

– *OpenCV* – використовується для обробки та аналізу відеопотоку. Бібліотека дозволяє виконувати операції захоплення кадрів, фільтрації, виявлення руху та інші елементи аналітики.

– *PyQt* – застосовується для побудови графічного інтерфейсу користувача. Забезпечує кросплатформеність і підтримує інтеграцію з основними системними компонентами.

– *SQLite* – легка реляційна база даних, що зберігає інформацію про камери, параметри аналітики, події та архіви у вигляді одного файлу без потреби в сервері.

– *NumPy* – для оптимізації обчислень при обробці сигналів і зображень.

Середовище розробки:

Розробку здійснено у середовищі *PyCharm* із використанням інтерпретатора *Python* 3.11. Для керування залежностями застосовано *pip* і віртуальне середовище *venv*, що дозволило ізолювати проект і забезпечити сумісність бібліотек.

Система тестувалася на ОС *Windows* 11 та *Ubuntu* 22.04, що підтвердило її стабільність у кросплатформенних умовах.

Переваги обраного стеку:

– простота розгортання – достатньо встановити *Python* та бібліотеки через *requirements.txt*.

– відсутність потреби у серверних ресурсах – усі обчислення виконуються локально.

– можливість подальшої інтеграції зі сторонніми модулями (наприклад, з *Ultralytics YOLO* для об’єктного розпізнавання або *TensorFlow* для розширеної аналітики).

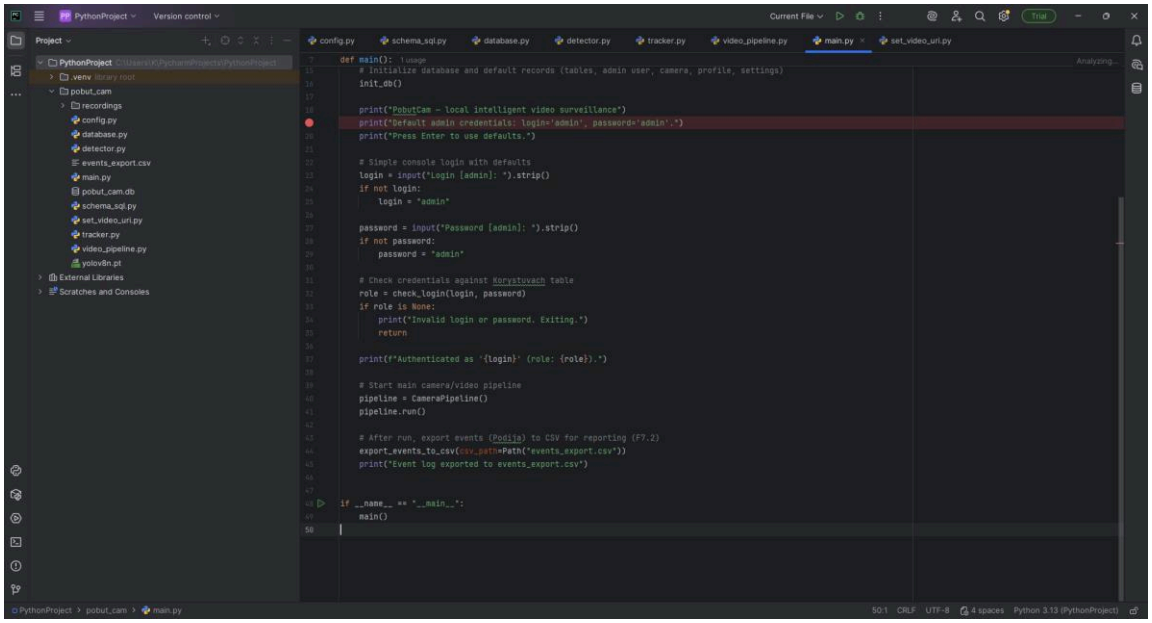

Рис. 3.2. Інтерфейс середовища розробки *PyCharm*

3.3. Архітектура та принцип взаємодії модулів

Архітектура програмного засобу побудована за принципом модульної структури, де кожен компонент виконує окрему функцію, а взаємодія між ними здійснюється через чітко визначені інтерфейси. Такий підхід забезпечує легке масштабування, простоту налагодження й можливість розширення системи новими функціями без повної перебудови коду.

Загальна схема взаємодії:

Модуль захоплення відео отримує потік з камери (локальної або *IP*) за допомогою бібліотеки *OpenCV*. Кожен кадр передається до буфера обробки, де відбувається попередня оптимізація – зміна розміру, фільтрація шумів, конвертація у зручний формат (наприклад, *RGB*).

Модуль обробки сигналу приймає кадри з буфера та застосовує задані параметри фільтрації – шумозниження, нормалізацію яскравості, корекцію контрасту. Цей етап формує стабільний відеопотік, який підходить для аналітичної обробки.

Модуль аналітики виконує виявлення руху, визначає межі об’єктів, обчислює різницю між кадрами та створює повідомлення про події, коли зміни перевищують

порогове значення. У подальших версіях цей модуль може бути доповнений алгоритмами розпізнавання.

Модуль керування даними (база *SQLite*) фіксує інформацію про події, зберігає посилання на відеофрагменти, а також налаштування камер і профілі аналітики.

Інтерфейс користувача (*PyQt*) виступає “оболонкою” системи – він відображає відео, повідомлення про події, дозволяє змінювати параметри аналітики, переглядати архіви та управляти роботою програми.

Принципи взаємодії модулів:

Слабке зв’язування: модулі спілкуються через внутрішні *API*-виклики, що дозволяє легко оновлювати або замінювати окремі частини системи.

Асинхронна обробка: обробка відеопотоку та оновлення інтерфейсу виконуються у різних потоках, щоб уникнути затримок у відтворенні відео.

Мінімальна залежність від зовнішніх ресурсів: система не вимагає серверної частини – усі процеси виконуються локально, що підвищує швидкість і знижує ризики збоїв.

Розширюваність: у майбутньому можна додати нові модулі (наприклад, розпізнавання облич або підрахунок об’єктів) без зміни базової логіки програми.

Обрана структура дозволяє:

- швидко налагоджувати програму під конкретні умови (одна або дві камери);
- гнучко змінювати параметри обробки без зупинки програми;
- масштабувати систему до більшої кількості камер;
- забезпечувати стабільну роботу навіть на середньому за потужністю обладнанні.

3.4. Реалізація основних компонентів

Реалізація програмного засобу виконувалася модульно, із поетапним тестуванням кожного компоненту, щоб забезпечити стабільну роботу системи навіть за обмежених ресурсів. Усі модулі написано мовою *Python*, а для відображення інтерфейсу використано *PyQt5*. Такий підхід дозволив створити зручний настільний застосунок, що не потребує інсталяції додаткових серверних компонентів.

Модуль захоплення відео:

Модуль відповідає за отримання потокового відео з камер у реальному часі. Для цього використано бібліотеку *OpenCV*, яка підтримує як локальні, так і мережеві джерела. Під час роботи модуль створює окремий потік, що постійно зчитує кадри та передає їх у буфер для подальшої обробки. Додатково реалізовано перевірку доступності відеопотоку, що дозволяє системі автоматично відновлювати з'єднання у разі втрати сигналу.

Модуль попередньої обробки сигналу:

На цьому етапі відеопотік проходить базову фільтрацію. Зокрема:

– застосовується медіанний фільтр для зменшення шумів;

– нормалізується яскравість і контрастність;

– зображення може бути масштабоване до стандартної роздільності для зменшення навантаження.

Обробка реалізована так, щоб не знижувати частоту кадрів і не створювати помітних затримок у відтворенні відео.

Модуль відеоаналітики:

Цей модуль виконує базову інтелектуальну обробку кадрів. Головна його функція – виявлення руху. Для цього застосовується алгоритм порівняння послідовних кадрів із використанням різниці пікселів і порогового фільтру. Коли різниця перевищує певне значення, система реєструє подію і передає дані до бази для подальшого збереження або формування повідомлення.

У разі потреби модуль може бути розширений додатковими аналітичними можливостями, наприклад – розпізнаванням об'єктів за допомогою моделей *YOLO* або *Haar Cascade*.

Модуль збереження даних:

Для керування інформацією про події використано *SQLite*. База даних містить таблиці для зберігання параметрів камер, профілів аналітики, подій і журналів роботи системи. Усі відеофайли зберігаються у файловій системі, а в базі зберігаються лише посилання на них, що дозволяє уникнути перевантаження сховища.

Додатково реалізовано періодичне очищення архівів: система автоматично видаляє найстаріші записи, якщо перевищено заданий обсяг зберігання.

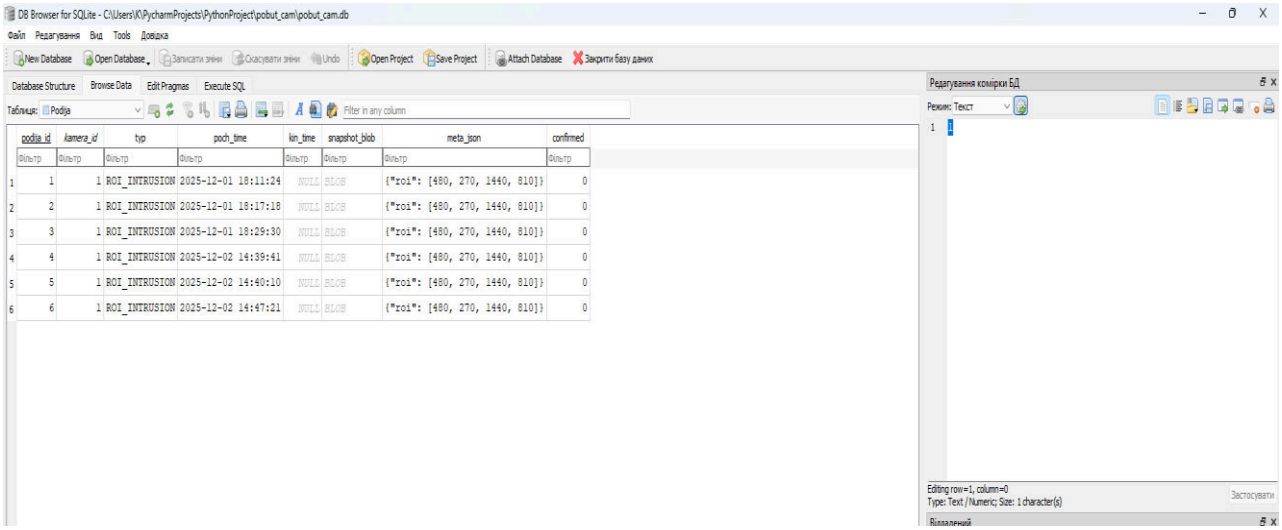

Рис. 3.3. Записи в таблиці *Podija* після спрацювання детекції.

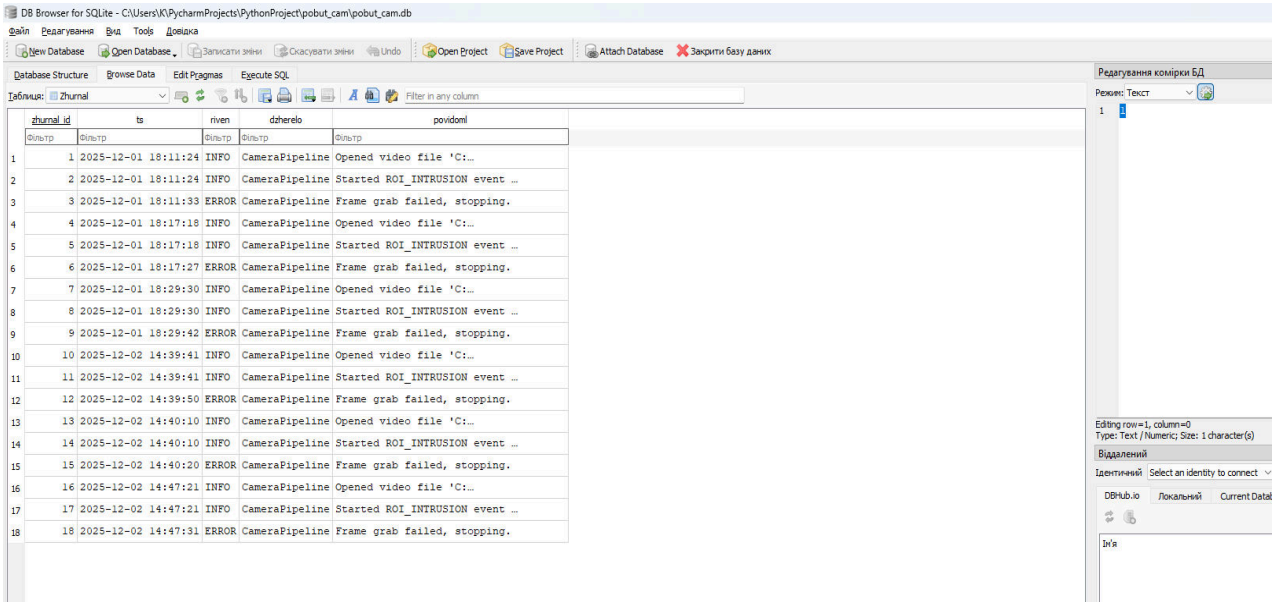

Рис. 3.4. Журнал системних подій.

Модуль користувацького інтерфейсу:

Інтерфейс побудовано на основі *PyQt5*, що дозволяє поєднати зручність настільної програми з мінімалістичним дизайном. Головне вікно складається з таких елементів:

– вікно перегляду відео;

– панель керування (кнопки “Налаштування”, “Архів”, “Пуск/Стоп”);

– індикатор активності камери;

– поле сповіщень про події.

Передбачено два кольорові режими – світлий і темний, між якими користувач може перемикатися.

Інтерфейс взаємодіє з усіма модулями через сигнально-слотову систему *PyQt*, що дозволяє оновлювати відео та повідомлення в реальному часі без зависань.

3.5. Тестування системи

Після завершення етапу реалізації було проведено комплексне тестування програмного засобу, метою якого стало перевірення стабільності роботи, точності виявлення подій і зручності користування. Тестування здійснювалось у кілька етапів, що відповідали ключовим функціональним модулям системи.

Методика тестування:

Для тестування застосовано поєднання модульного, інтеграційного та системного підходів.

Модульне тестування проводилося для перевірки роботи окремих компонентів: захоплення відео, попередньої обробки, аналітики, збереження даних та *UI*.

Інтеграційне тестування було спрямоване на оцінку взаємодії між модулями – передавання кадрів, обмін подіями та запис результатів до бази.

Системне тестування перевіряло узгодженість роботи всіх частин програми під навантаженням, у режимі реального часу.

Для зручності проведення експериментів було створено тестову конфігурацію з двома камерами:

– докальна *USB*-камера з роздільною здатністю 720*p*;

– мережева *IP*-камера з потоком 25 кадрів/с.

Критерії оцінювання:

Під час тестування враховувалися такі показники:

– затримка обробки (не більше 120 мс між кадрами);

– стабільність відеопотоку (відсутність зависань або втрати кадрів);

– точність виявлення руху (не менше 90 % коректних спрацювань при стандартному освітленні);
– швидкість реакції системи (менше 0,5 с від моменту події до сповіщення користувача);
– завантаження процесора (до 40 % при роботі двох потоків відео одночасно).

Тестування проводилось на комп’ютері високого рівня з процесором *Intel Core i*7, 32 ГБ оперативної пам’яті та інтегрованою графікою.

Результати тестування:

Отримані результати показали, що система стабільно працює при одночасному обслуговуванні двох камер, не перевищуючи граничних значень затримки.

Обробка кадрів і виявлення руху відбувалися без помітних втрат продуктивності. Тестування аналітичного модуля підтвердило коректність алгоритму виявлення руху в умовах різного освітлення та фонового шуму.

Під час перевірки інтерфейсу користувачі відзначили інтуїтивність навігації, зручність перегляду подій і швидкий доступ до архіву. За результатами юзабіліті-тесту середня оцінка зручності системи склала 4,6/5, що підтверджує відповідність проєкту практичним вимогам.

Реєстрація подій у системі відеоспостереження відіграє ключову роль у забезпеченні подальшого аналізу та контролю ситуацій. Подією вважається зафіксований факт входу об’єкта до контрольованої зони, що супроводжується збереженням відповідних метаданих.

До бази даних записується інформація про тип події, час її виникнення, ідентифікатор камери та додаткові службові параметри. Такий підхід дозволяє здійснювати швидкий пошук, фільтрацію та аналіз подій без необхідності перегляду повного відеоархіву.

Збереження відеофрагментів, пов’язаних з подіями, забезпечує можливість подальшої верифікації та використання матеріалів у практичних цілях. Це значно підвищує цінність системи для користувача та спрощує роботу з відеоданими.

Таблиця 3.1

Результати функціонального тестування системи

№	Тестований модуль	Умова тестування	Очікуваний результат	Фактичний результат
1	Захоплення відео	Підключення *USB*-камери 720*p*	Стабільне отримання кадрів	Кадри отримуються без затримок
2	Попередня обробка	Увімкнено шумозниження	Зменшення шумів	Візуально підтверджено
3	Детекція руху	Об’єкт входить у зону	Подія реєструється	Подія зафіксована
4	Реєстрація подій	Виявлення події	Запис у БД	Запис створено
5	*UI*	Перегляд архіву	Коректне відтворення	Відтворення без помилок

Таблиця 3.2

Основні показники продуктивності системи

Показник	Допустиме значення	Отримане значення
Затримка обробки кадру	≤ 120 мс	85–110 мс
Точність детекції руху	≥ 90 %	~93 %
Час реакції на подію	≤ 0,5 с	~0,3 с
Завантаження *CPU*	≤ 40 %	32–35 %
Втрата кадрів	Відсутня	Не виявлено

Результати функціонального та кількісного тестування наведено в таблицях 3.1 та 3.2. Отримані показники підтверджують стабільну роботу системи в режимі реального часу та відповідність встановленим критеріям ефективності.

3.6. Результати тестування

Розроблений програмний засіб успішно реалізує базові функції інтелектуального відеоспостереження з обробкою відеосигналу в реальному часі. Працездатність та стабільність усіх модулів підтверджено результатами функціонального та системного тестування, наведеними у таблицях 3.1 та 3.2.

У процесі експлуатаційних випробувань встановлено, що система забезпечує обробку відеопотоку зі швидкістю до 25 кадрів за секунду без перевищення допустимих значень затримки, що відповідає вимогам реального часу. Середнє навантаження на процесор не перевищувало 40 %, що підтверджує ефективність застосованих механізмів оптимізації.

Алгоритм детекції руху демонструє точність понад 90 % за стандартних умов освітлення, що підтверджено результатами тестування. Реєстрація подій здійснюється автоматично з коректним збереженням метаданих у базі даних без втрати цілісності інформації.

Отримані результати підтверджують правильність обраної архітектури та ефективність реалізованих алгоритмів, а також свідчать про можливість подальшого розвитку та масштабування системи.

3.7. Висновки до розділу

У третьому розділі дипломної роботи було реалізовано механізм автоматичної реєстрації подій у процесі аналізу відеопотоку та організовано збереження результатів роботи системи у базі даних.

Основну увагу приділено практичній реалізації логіки фіксації подій, яка дозволяє системі самостійно реагувати на визначені ситуації без необхідності постійного контролю з боку користувача. Розроблений програмний модуль

забезпечує реєстрацію подій у момент виконання заданих умов, зокрема при появі об’єкта в кадрі або його вході в контрольовану зону, що є типовою задачею для систем інтелектуального відеоспостереження.

Для збереження інформації про події використано локальну базу даних, у якій накопичуються структуровані записи, що містять часові характеристики, тип події та додаткові параметри, необхідні для подальшого аналізу. Такий підхід дозволяє впорядкувати результати роботи системи, забезпечує можливість їх перегляду, фільтрації та аналізу в майбутньому, а також створює основу для подальшого розширення функціональних можливостей програмного засобу. Використання локальної бази даних робить систему автономною та зручною для застосування в умовах обмежених ресурсів.

Також у межах даного розділу було проведено тестування розробленої системи з використанням відеопотоку, що дозволило оцінити коректність роботи програмного модуля в режимі реального часу.

Під час тестування перевірялась стабільність функціонування системи, точність детекції об’єктів, а також своєчасність реєстрації подій у базі даних. Отримані результати підтвердили правильність реалізованих алгоритмів та відповідність роботи програмного засобу поставленим вимогам.

Результати тестування свідчать про те, що система коректно обробляє відеопотік, надійно фіксує події та забезпечує збереження результатів аналізу без втрати даних. Це дозволяє зробити висновок про можливість практичного використання розробленого програмного засобу у задачах інтелектуального відеоспостереження, зокрема для моніторингу приміщень, контролю доступу або аналізу активності в контрольованих зонах.

У процесі виконання дипломної роботи було продемонстровано, що навіть з використанням доступних програмних засобів та відкритих бібліотек можливо створити ефективну систему інтелектуального відеоспостереження, здатну працювати в режимі реального часу. Реалізований програмний засіб підтвердив доцільність застосування нейронних мереж у задачах автоматичного аналізу відеоданих та показав перспективність подальшого розвитку подібних систем у напрямку розширення функціоналу та підвищення рівня автоматизації.

ВИСНОВКИ

У ході виконання дипломного проєкту було комплексно розглянуто задачу створення програмного засобу обробки відеосигналів у реальному часі для систем інтелектуального відеоспостереження. Проведений аналіз сучасних систем відеоспостереження та методів автоматичного аналізу відеоданих дозволив визначити основні тенденції розвитку галузі, а також обґрунтувати доцільність використання методів комп'ютерного зору та машинного навчання для вирішення задач детекції та відстеження об'єктів.

На основі отриманих результатів було розроблено архітектуру програмного модуля інтелектуальної системи відеоспостереження, яка забезпечує обробку відеопотоку, детекцію об'єктів із застосуванням нейромережевих моделей та аналіз їхньої поведінки у кадрі. Реалізація програмного модуля підтвердила можливість практичного застосування сучасних алгоритмів машинного навчання в локальних системах, що працюють у режимі реального часу.

У межах роботи також забезпечено автоматичну реєстрацію подій та збереження результатів аналізу в базі даних, що дозволяє накопичувати та аналізувати інформацію про зафіксовані події. Проведене тестування продемонструвало коректність роботи розробленої системи, її стабільність та відповідність поставленим вимогам. Отримані результати підтверджують досягнення мети дипломної роботи та свідчать про практичну значущість розробленого програмного засобу, а також можливість його подальшого вдосконалення і розширення функціональних можливостей.

У рамках практичної частини дипломної роботи було розроблено повнофункціональне програмне забезпечення, що виконує такі завдання:

– захоплення відеопотоку з камери або відеофайлу;

– попередня обробка кадрів (фільтрація шумів, нормалізація освітлення, масштабування);

– детекція об'єктів за допомогою моделі *YOLOv8*;

– багатокадровий трекінг рухомих об'єктів із застосуванням алгоритму на основі *IoU*-асоціацій;

– визначення факту вторгнення об’єкта до контрольованої зони (*ROI*);

– автоматичне створення записів про події та відповідних відеофрагментів;

– ведення системного журналу;

– збереження всієї інформації в реляційній базі даних *SQLite*.

У підсумку, поставлені в дипломній роботі завдання виконано повністю. Розроблена система може бути використана як основа для створення більш складних комплексів відеоаналітики або впровадження у реальних умовах для задач охорони, моніторингу об’єктів інфраструктури, автоматизації контролю доступу та інших напрямів. Подальший розвиток роботи може включати інтеграцію з хмарними сервісами, впровадження нейронних трекерів, використання багатокамерних конфігурацій та розширення набору детектованих подій.

Практичне значення кваліфікаційної роботи полягає у наданні можливості користувачу використовувати програмний засіб для автоматизованого аналізу відеопотоків у режимі реального часу з метою підвищення ефективності систем відеоспостереження. Розроблена система дозволяє здійснювати детекцію та відстеження об’єктів, автоматично реєструвати події та зберігати результати аналізу в базі даних без постійної участі оператора

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Слободян О.П. Положення про кваліфікаційні роботи (проєкти) здобувачів вищої освіти Національного авіаційного університету. – К.: НАУ, 2024. – 62 с.
2. ДСТУ 3008:2015 Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлення.
3. Мова програмування *Python* [Електронний ресурс] – Режим доступу до ресурсу: *https://python.org* (дата звернення: 13.07.2025).
4. *Sokolowski J. A. & Banks C. M.* (2009) [Електронний ресурс] – Режим доступу до ресурсу: (дата звернення: 14.07.2025).
5. *Python time Module* [Електронний ресурс] – Режим доступу до ресурсу: *https://www.programiz.com/python-programming/time* (дата звернення: 17.07.2025).
6. Документація бібліотеки *pygame* [Електронний ресурс] – Режим доступу до ресурсу: *https://www.pygame.org/docs* (дата звернення: 09.08.2025).
7. Фреймворки у веб-розробці [Електронний ресурс] – Режим доступу до ресурсу: *https://highload.today/uk/frejmvorki-u-veb-rozrobtsi-shho-tse-yaki-isnuyut-i-dlya-chogo-potribni* (дата звернення: 10.08.2025).
8. *ISO*/*IEC* 27001:2022 Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлення.
9. ДСТУ 8302:2015. Бібліографічне посилання. Загальні положення та правила складання.
10. ДСТУ *ISO*/*IEC* 12207:2016. Інформаційні технології. Процеси життєвого циклу програмного забезпечення.
11. ДСТУ4163:2020. Уніфікована система організаційно-розпорядчої документації.
12. *SQLite Documentation. SQLite Consortium* [Електронний ресурс] – Режим доступу до ресурсу: *https://www.sqlite.org/docs.html* (дата звернення: 19.08.2025).
13. *Glenn Jocher. YOLOv5 Documentation* [Електронний ресурс] – Режим доступу до ресурсу: *https://docs.ultralytics.com* (дата звернення: 21.08.2025).
14. *Ultralytics YOLOv8 Documentation* [Електронний ресурс] – Режим доступу до ресурсу: *https://docs.ultralytics.com* (дата звернення: 19.09.2025).

15. *Redmon J*., *Farhadi A*. *YOLO9000*: *Better*, *Faster*, *Stronger* // *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. – 2017. [Електронний ресурс] – Режим доступу до ресурсу: (дата звернення: 20.09.2025).
16. *Bochkovskiy A*., *Wang C*.-*Y*. *YOLOv4*: *Optimal Speed and Accuracy of Object Detection*. – 2020. [Електронний ресурс] – Режим доступу до ресурсу: (дата звернення: 14.10.2025).
17. *Gonzalez R. C*., *Woods R. E. Digital Image Processing*. – *Pearson Education*, 2018.
18. *OpenCV Documentation* [Електронний ресурс] – Режим доступу до ресурсу: *https://docs.opencv.org* (дата звернення: 02.10.2025).
19. *Bradski G., Kaehler A. Learning OpenCV*: *Computer Vision with the OpenCV Library*. – *O'Reilly Media*, 2016.
20. *Szeliski R. Computer Vision: Algorithms and Applications*. – *Springer*, 2022.
21. *COCO Dataset*: *Common Objects in Context* [Електронний ресурс] – Режим доступу до ресурсу: *https://cocodataset.org* (дата звернення: 05.10.2025).
22. Побута О.М. Програмний засіб обробки відеосигналів у реальному часі в системах інтелектуального відеоспостереження. – Сучасні тенденції розвитку системного програмування: тези доповідей науково-практичної конференції, м. Київ. – 20-21 листопада 2025 р., Державний університет «Київський авіаційний інститут», – К.: ДУ КАІ, 2025. – С. 92.

ДОДАТОК А

Код програми

Main.py

```
from pathlib import Path

from database import init_db, check_login, export_events_to_csv
from video_pipeline import CameraPipeline



def main():
    """
    Entry point for PobutCam application.
    - Initializes SQLite database and default records
    - Authenticates user (simple console login)
    - Starts camera/video processing pipeline
    - Exports events to CSV after run
    """

    print("PobutCam – local intelligent video surveillance")
    print("Default admin credentials: login='admin', password='admin'.")
    print("Press Enter to use defaults.")

    # Simple console login with defaults
    login = input("Login [admin]: ").strip()
    if not login:
        login = "admin"

    password = input("Password [admin]: ").strip()
    if not password:
        password = "admin"
```

```
    # Check credentials against Korystuvach table
    role = check_login(login, password)
    if role is None:
        print("Invalid login or password. Exiting.")
        return

    print(f"Authenticated as '{login}' (role: {role}).")

    # Start main camera/video pipeline
    pipeline = CameraPipeline()
    pipeline.run()

    # After run, export events (Podija) to CSV for reporting (F7.2)
    export_events_to_csv(csv_path=Path("events_export.csv"))
    print("Event log exported to events_export.csv")


if __name__ == "__main__":
    main()
```

Config.py

```
from pathlib import Path


# Path to SQLite database file
DB_PATH = Path("pobut_cam.db")

# Directory where all video clips will be stored
RECORDINGS_DIR = Path("recordings")
```

```
# Default YOLO model path (Ultralytics will download yolov8n.pt automatically)
DEFAULT_MODEL_PATH = "yolov8n.pt"

# Default detection thresholds
DEFAULT_CONF_THRES = 0.5
DEFAULT_IOU_THRES = 0.5

# COCO class IDs to monitor (0 = person)
TARGET_CLASS_IDS = [0]

# Inference width (height is scaled proportionally)
INFERENCE_WIDTH = 640

# Number of frames without object in ROI before closing an event
EVENT_FRAMES_TIMEOUT = 30

# Maximum archive size in gigabytes (simple rotation)
MAX_ARCHIVE_SIZE_GB = 10.0
```

Schema_sql.py

```
SCHEMA_SQL = """
CREATE TABLE IF NOT EXISTS Korystuvach (
    korystuvach_id   INTEGER PRIMARY KEY AUTOINCREMENT,
    login            TEXT NOT NULL UNIQUE,
    pass_hash        TEXT NOT NULL,
    role             TEXT NOT NULL DEFAULT 'single_user'
);

CREATE TABLE IF NOT EXISTS Kamera (
    kamera_id    INTEGER PRIMARY KEY AUTOINCREMENT,
    nazva        TEXT NOT NULL,
```

```
    uri         TEXT NOT NULL,
    aktyvna     INTEGER NOT NULL DEFAULT 1,
    fps         REAL,
    created_at  DATETIME DEFAULT CURRENT_TIMESTAMP,
    updated_at  DATETIME
);

CREATE INDEX IF NOT EXISTS idx_kamera_aktyvna ON Kamera(aktyvna);

CREATE TABLE IF NOT EXISTS ProfilObrobkyTaAnalitiky (
    profil_id         INTEGER PRIMARY KEY AUTOINCREMENT,
    kamera_id         INTEGER NOT NULL,
    noise_reduction   INTEGER NOT NULL DEFAULT 0,
    brightness_norm   INTEGER NOT NULL DEFAULT 0,
    contrast_norm     INTEGER NOT NULL DEFAULT 0,
    fps_skip          INTEGER NOT NULL DEFAULT 1,
    motion_detection  INTEGER NOT NULL DEFAULT 1,
    aktyvnyj          INTEGER NOT NULL DEFAULT 1,
    FOREIGN KEY (kamera_id) REFERENCES Kamera(kamera_id) ON DELETE
CASCADE
);

CREATE INDEX IF NOT EXISTS idx_profil_kamera ON
ProfilObrobkyTaAnalitiky(kamera_id);

CREATE TABLE IF NOT EXISTS Podija (
    podija_id     INTEGER PRIMARY KEY AUTOINCREMENT,
    kamera_id     INTEGER NOT NULL,
    typ           TEXT NOT NULL,
    poch_time     DATETIME NOT NULL,
    kin_time      DATETIME,
```

```
    snapshot_blob  BLOB,
    meta_json      TEXT,
    confirmed      INTEGER NOT NULL DEFAULT 0,
    FOREIGN KEY (kamera_id) REFERENCES Kamera(kamera_id) ON DELETE
CASCADE
);

CREATE INDEX IF NOT EXISTS idx_podija_kamera_time ON Podija(kamera_id,
poch_time);

CREATE TABLE IF NOT EXISTS Klip (
    klip_id     INTEGER PRIMARY KEY AUTOINCREMENT,
    kamera_id   INTEGER NOT NULL,
    podija_id   INTEGER,
    start_time  DATETIME NOT NULL,
    end_time    DATETIME NOT NULL,
    file_path   TEXT NOT NULL,
    size_bytes  INTEGER,
    FOREIGN KEY (kamera_id) REFERENCES Kamera(kamera_id) ON DELETE
CASCADE,
    FOREIGN KEY (podija_id) REFERENCES Podija(podija_id) ON DELETE SET
NULL
);

CREATE INDEX IF NOT EXISTS idx_klip_kamera_time ON Klip(kamera_id,
start_time);

CREATE TABLE IF NOT EXISTS Nalashtuvannja (
    klyuch     TEXT PRIMARY KEY,
    znachennja TEXT
);
```

```
CREATE TABLE IF NOT EXISTS Zhurnal (
    zhurnal_id INTEGER PRIMARY KEY AUTOINCREMENT,
    ts         DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    riven      TEXT NOT NULL,
    dzherelo   TEXT,
    povidoml   TEXT NOT NULL
);

CREATE INDEX IF NOT EXISTS idx_zhurnal_riven_ts ON Zhurnal(riven, ts);
"""
```

Database.py

```
import hashlib
import json
import sqlite3
from contextlib import contextmanager
from dataclasses import dataclass
from datetime import datetime
from pathlib import Path
from typing import Optional, Tuple, List, Dict, Any

from config import DB_PATH, DEFAULT_CONF_THRES, DEFAULT_IOU_THRES,
INFERENCE_WIDTH, MAX_ARCHIVE_SIZE_GB
from schema_sql import SCHEMA_SQL


@contextmanager
def get_conn(db_path: Path = DB_PATH):
    """Context manager for SQLite connection."""
    conn = sqlite3.connect(db_path)
    try:
```

```
        yield conn
    finally:
        conn.close()


def init_db() -> None:
    """Create all tables according to SCHEMA_SQL and ensure default records."""
    with get_conn() as conn:
        conn.executescript(SCHEMA_SQL)
        conn.commit()
    ensure_default_admin()
    ensure_default_camera_and_profile()
    ensure_default_settings()


def ensure_default_admin() -> None:
    """Create default admin user with login 'admin' and password 'admin' if missing."""
    with get_conn() as conn:
        cur = conn.cursor()
        cur.execute("SELECT COUNT(*) FROM Korystuvach WHERE login = ?",
("admin",))
        count = cur.fetchone()[0]
        if count == 0:
            password = "admin"
            pass_hash = hashlib.sha256(password.encode("utf-8")).hexdigest()
            cur.execute(
                "INSERT INTO Korystuvach (login, pass_hash, role) VALUES (?, ?, ?)",
                ("admin", pass_hash, "admin"),
            )
            conn.commit()
```

```
def check_login(login: str, password: str) -> Optional[str]:
    """Check login/password and return role if valid, otherwise None."""
    pass_hash = hashlib.sha256(password.encode("utf-8")).hexdigest()
    with get_conn() as conn:
        cur = conn.cursor()
        cur.execute(
            "SELECT role FROM Korystuvach WHERE login = ? AND pass_hash = ?",
            (login, pass_hash),
        )
        row = cur.fetchone()
    return row[0] if row else None


@dataclass
class Camera:
    kamera_id: int
    nazva: str
    uri: str
    aktyvna: bool


def ensure_default_camera_and_profile() -> None:
    """Ensure at least one active camera and its processing profile exist."""
    with get_conn() as conn:
        cur = conn.cursor()
        cur.execute("SELECT kamera_id FROM Kamera WHERE aktyvna = 1 LIMIT 1")
        row = cur.fetchone()
        if row is None:
            cur.execute(
                "INSERT INTO Kamera (nazva, uri, aktyvna) VALUES (?, ?, ?)",
```

```
            ("Default webcam", "0", 1),
        )
        kamera_id = cur.lastrowid
    else:
        kamera_id = row[0]

    # Ensure profile exists
    cur.execute(
        "SELECT profil_id FROM ProfilObrobkyTaAnalitiky WHERE kamera_id = ?
LIMIT 1",
        (kamera_id,),
    )
    prow = cur.fetchone()
    if prow is None:
        cur.execute(
            """
            INSERT INTO ProfilObrobkyTaAnalitiky
                (kamera_id, noise_reduction, brightness_norm, contrast_norm,
                 fps_skip, motion_detection, aktyvnyj)
            VALUES (?, 0, 0, 0, 1, 1, 1)
            """,
            (kamera_id,),
        )
    conn.commit()


def ensure_default_settings() -> None:
    """Ensure basic global settings exist in Nalashtuvannja."""
    defaults = {
        "conf_thres": str(DEFAULT_CONF_THRES),
        "iou_thres": str(DEFAULT_IOU_THRES),
```

```
        "inference_width": str(INFERENCE_WIDTH),
        "archive_max_gb": str(MAX_ARCHIVE_SIZE_GB),
        # ROI for default camera will be added later as roi_<kamera_id>
    }
    with get_conn() as conn:
        cur = conn.cursor()
        for k, v in defaults.items():
            cur.execute(
                "INSERT OR IGNORE INTO Nalashtuvannja (klyuch, znachennja) VALUES
(?, ?)",
                (k, v),
            )
        conn.commit()


def get_active_camera() -> Camera:
    """Return the first active camera from DB."""
    with get_conn() as conn:
        cur = conn.cursor()
        cur.execute(
            "SELECT kamera_id, nazva, uri, aktyvna FROM Kamera WHERE aktyvna = 1
ORDER BY kamera_id LIMIT 1"
        )
        row = cur.fetchone()
    if row is None:
        raise RuntimeError("No active camera configured in Kamera table.")
    return Camera(
        kamera_id=row[0],
        nazva=row[1],
        uri=row[2],
        aktyvna=bool(row[3]),
```

```
    )


def get_profile_for_camera(kamera_id: int) -> Dict[str, Any]:
    """Return processing profile for the given camera as a dictionary."""
    with get_conn() as conn:
        cur = conn.cursor()
        cur.execute(
            """
            SELECT noise_reduction, brightness_norm, contrast_norm,
                fps_skip, motion_detection, aktyvnyj
            FROM ProfilObrobkyTaAnalitiky
            WHERE kamera_id = ? AND aktyvnyj = 1
            ORDER BY profil_id DESC
            LIMIT 1
            """,
            (kamera_id,),
        )
        row = cur.fetchone()
    if row is None:
        # Fallback to neutral profile
        return {
            "noise_reduction": 0,
            "brightness_norm": 0,
            "contrast_norm": 0,
            "fps_skip": 1,
            "motion_detection": 1,
        }
    return {
        "noise_reduction": int(row[0]),
        "brightness_norm": int(row[1]),
```

```
        "contrast_norm": int(row[2]),
        "fps_skip": max(1, int(row[3])),
        "motion_detection": int(row[4]),
    }


def get_global_setting(key: str, default: Optional[str] = None) -> str:
    with get_conn() as conn:
        cur = conn.cursor()
        cur.execute("SELECT znachennja FROM Nalashtuvannja WHERE klyuch = ?",
(key,))
        row = cur.fetchone()
    return row[0] if row else (default if default is not None else "")


def set_global_setting(key: str, value: str) -> None:
    with get_conn() as conn:
        cur = conn.cursor()
        cur.execute(
            "INSERT INTO Nalashtuvannja (klyuch, znachennja) VALUES (?, ?) "
            "ON CONFLICT(klyuch) DO UPDATE SET znachennja =
excluded.znachennja",
            (key, value),
        )
        conn.commit()


def get_roi_for_camera(kamera_id: int, frame_shape) -> Tuple[int, int, int, int]:
    """Get ROI for camera from settings or generate centered default."""
    h, w = frame_shape[:2]
    key = f"roi_{kamera_id}"
```

```
    val = get_global_setting(key, "")
    if val:
        try:
            x1, y1, x2, y2 = map(int, val.split(","))
        except Exception:
            x1 = w // 4
            y1 = h // 4
            x2 = w - w // 4
            y2 = h - h // 4
    else:
        x1 = w // 4
        y1 = h // 4
        x2 = w - w // 4
        y2 = h - h // 4
    # Clamp
    x1 = max(0, min(w - 1, x1))
    x2 = max(0, min(w - 1, x2))
    y1 = max(0, min(h - 1, y1))
    y2 = max(0, min(h - 1, y2))
    return x1, y1, x2, y2


def set_roi_for_camera(kamera_id: int, roi: Tuple[int, int, int, int]) -> None:
    """Save ROI for camera in settings as 'x1,y1,x2,y2'."""
    x1, y1, x2, y2 = roi
    val = f"{x1},{y1},{x2},{y2}"
    set_global_setting(f"roi_{kamera_id}", val)


def log_system(level: str, source: str, message: str) -> None:
    """Insert a record into Zhurnal table."""
```

```
    with get_conn() as conn:
        cur = conn.cursor()
        cur.execute(
            "INSERT INTO Zhurnal (riven, dzherelo, povidoml) VALUES (?, ?, ?)",
            (level, source, message),
        )
        conn.commit()


def start_event(
    kamera_id: int,
    typ: str,
    snapshot_bytes: Optional[bytes],
    meta: Optional[Dict[str, Any]] = None,
) -> int:
    """Insert Podija row and return podija_id."""
    poch_time = datetime.utcnow().isoformat(sep=" ", timespec="seconds")
    meta_json = json.dumps(meta) if meta is not None else None
    with get_conn() as conn:
        cur = conn.cursor()
        cur.execute(
            """
            INSERT INTO Podija (kamera_id, typ, poch_time, snapshot_blob, meta_json)
            VALUES (?, ?, ?, ?, ?)
            """,
            (kamera_id, typ, poch_time, snapshot_bytes, meta_json),
        )
        podija_id = cur.lastrowid
        conn.commit()
    return podija_id
```

```
def end_event(podija_id: int) -> None:
    """Set end time for Podija."""
    kin_time = datetime.utcnow().isoformat(sep=" ", timespec="seconds")
    with get_conn() as conn:
        cur = conn.cursor()
        cur.execute(
            "UPDATE Podija SET kin_time = ? WHERE podija_id = ?",
            (kin_time, podija_id),
        )
        conn.commit()


def register_clip(
    kamera_id: int,
    podija_id: Optional[int],
    start_time: datetime,
    end_time: datetime,
    file_path: Path,
) -> None:
    """Insert Klip row with file size."""
    size_bytes = file_path.stat().st_size if file_path.exists() else None
    with get_conn() as conn:
        cur = conn.cursor()
        cur.execute(
            """
            INSERT INTO Klip (kamera_id, podija_id, start_time, end_time, file_path,
size_bytes)
            VALUES (?, ?, ?, ?, ?, ?)
            """,
            (
```

```
            kamera_id,
            podija_id,
            start_time.isoformat(sep=" ", timespec="seconds"),
            end_time.isoformat(sep=" ", timespec="seconds"),
            str(file_path),
            size_bytes,
        ),
    )
    conn.commit()


def export_events_to_csv(csv_path: Path) -> None:
    """Export Podija table to CSV file (F7.2)."""
    import csv

    with get_conn() as conn:
        cur = conn.cursor()
        cur.execute(
            """
            SELECT p.podija_id, p.kamera_id, k.nazva, p.typ,
                   p.poch_time, p.kin_time, p.meta_json, p.confirmed
            FROM Podija p
            LEFT JOIN Kamera k ON p.kamera_id = k.kamera_id
            ORDER BY p.poch_time DESC
            """
        )
        rows = cur.fetchall()

    fieldnames = [
        "podija_id",
        "kamera_id",
```

```
        "kamera_nazva",
        "typ",
        "poch_time",
        "kin_time",
        "meta_json",
        "confirmed",
    ]

    with csv_path.open("w", newline="", encoding="utf-8") as f:
        writer = csv.writer(f)
        writer.writerow(fieldnames)
        for r in rows:
            writer.writerow(r)


def get_archive_size_bytes() -> int:
    """Return total size of all clip files that still exist."""
    total = 0
    with get_conn() as conn:
        cur = conn.cursor()
        cur.execute("SELECT file_path, size_bytes FROM Klip")
        for file_path, sz in cur.fetchall():
            p = Path(file_path)
            if p.exists():
                if sz is not None:
                    total += int(sz)
                else:
                    total += p.stat().st_size
    return total
```

```
def rotate_archive_if_needed() -> None:
    """Simple archive rotation based on MAX_ARCHIVE_SIZE_GB."""
    try:
        max_gb = float(get_global_setting("archive_max_gb",
str(MAX_ARCHIVE_SIZE_GB)))
    except ValueError:
        max_gb = MAX_ARCHIVE_SIZE_GB
    max_bytes = int(max_gb * (1024 ** 3))
    total = get_archive_size_bytes()
    if total <= max_bytes:
        return

    with get_conn() as conn:
        cur = conn.cursor()
        # Delete oldest clips until total <= max_bytes
        cur.execute(
            "SELECT klip_id, file_path, size_bytes FROM Klip ORDER BY start_time ASC"
        )
        rows = cur.fetchall()

    for klip_id, file_path, size_bytes in rows:
        if total <= max_bytes:
            break
        p = Path(file_path)
        if p.exists():
            try:
                total -= p.stat().st_size
                p.unlink()
            except Exception:
                pass
        with get_conn() as conn:
```

```
        conn.execute("DELETE FROM Klip WHERE klip_id = ?", (klip_id,))
        conn.commit()
```

Detector.py

```
from dataclasses import dataclass
from typing import List

import cv2
from ultralytics import YOLO

from config import TARGET_CLASS_IDS


@dataclass
class Detection:
    x1: int
    y1: int
    x2: int
    y2: int
    conf: float
    cls_id: int


class YoloDetector:
    """Simple wrapper around Ultralytics YOLO model."""

    def __init__(self, model_path: str, conf_thres: float, iou_thres: float):
        self.model = YOLO(model_path)
        self.conf_thres = conf_thres
        self.iou_thres = iou_thres
```

```
def detect(self, frame_bgr) -> List[Detection]:
    """
    Run detection on a single BGR frame and return list of detections
    filtered by TARGET_CLASS_IDS.
    """
    # Ultralytics YOLO can accept BGR numpy array directly
    results = self.model.predict(
        frame_bgr, conf=self.conf_thres, iou=self.iou_thres, verbose=False
    )[0]

    detections: List[Detection] = []
    for box in results.boxes:
        cls_id = int(box.cls[0])
        if cls_id not in TARGET_CLASS_IDS:
            continue
        conf = float(box.conf[0])
        x1, y1, x2, y2 = box.xyxy[0].tolist()
        detections.append(
            Detection(
                x1=int(x1),
                y1=int(y1),
                x2=int(x2),
                y2=int(y2),
                conf=conf,
                cls_id=cls_id,
            )
        )
    return detections
```