Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Грабкова Романа Сергійовича
Прізвище, ім'я, по батькові студента(ки)

на здобуття ступеня вищої освіти Бакалавра

Програмна система для автоматизованого виявлення та класифікації об'єктів на аерофотознімках в режимі реального часу
Назва теми

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.2101072.01.04.ПЗ

Виконав студент IV курсу, група ІПЗ-21-1 _____ Роман ГРАБКОВ
Підпис Ім'я, ПРІЗВИЩЕ

Керівник д-р фіз.-мат. наук, проф. _____ Леонід БЕДРАТЮК
Науковий ступінь, вчене звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер к. техн. наук, доцент _____ Оксана ЯШИНА
Посада Підпис Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

6 червня 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
_____ Л. П. Бедратюк
02 01 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Грабкову Роману Сергійовичу
Прізвище, ім'я, по батькові студента

1. Тема роботи Програмна система для автоматизованого виявлення та класифікації об'єктів на аерофотознімках в режимі реального часу

Керівник роботи Бедратюк Леонід Петрович, д-р фіз.-мат. наук, професор
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 07.02.2025 р. № 8
2. Строк подання студентом роботи на кафедру 01.06.2025 р.
3. Вихідні дані до роботи Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) *Дослідження предметної області та постановка задачі, проєктування програмного забезпечення, програмна реалізація та тестування*
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Три креслення:
1. Діаграма класів
2. Діаграма класів та інтерфейсів
3. Діаграма IDEF0 декомпозиції системи

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Яшина О. М., доцент кафедри ІПЗ		
Антиплагіат	Форкун Ю. В., доцент кафедри ІПЗ	12.05.2025	1.06.2025

7. Дата видачі завдання «_02_» __січня_ 2025 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапу виконання	Дата початку- завершення етапу	Примітка
1 Ознайомлення з тематикою дипломного проектування, визначення та узгодження індивідуальних тем кваліфікаційних робіт (КвР)	01.12– 31.12.2024	
2 Збір матеріалу за темою КвР; дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	01.01 – 20.02.2025	
3 Проектування програмного забезпечення	21.02 – 20.03 2025	
4 Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ	21.03 – 30.04.2025	
5 Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05 – 25.05.2025	
6 Попередній захист КвР	травень 2025	Згідно графіка
7 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 – 30.05.2025	
8 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	з 01.06.2025	

Зав. кафедри ІПЗ _____Леонід БЕДРАТЮК

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Програмна система для автоматизованого виявлення та класифікації об’єктів на аерофотознімках в режимі реального часу».

Автор роботи: Грабков Роман Сергійович.

Керівник роботи: Бедратюк Леонід Петрович.

Пояснювальна записка: 82 с., 14 рис., 2 табл., 3 дод., 28 джерел.

КОМП'ЮТЕРНИЙ ЗІР, НЕЙРОННІ МЕРЕЖІ, ОБРОБКА АЕРОФОТОЗНІМКІВ, КЛАСИФІКАЦІЯ ОБ'ЄКТІВ, YOLO.

Мета кваліфікаційної роботи: розроблення програмної системи для автоматизованого виявлення та класифікації об’єктів на аерофотознімках у режимі реального часу.

У кваліфікаційній роботі проведено аналіз предметної області, визначено основні методи обробки аерофотознімків, досліджено сучасні підходи до автоматичного розпізнавання об’єктів. Описано функціональні та нефункціональні вимоги до програмної системи, розроблено архітектуру застосунку, реалізовано алгоритми обробки зображень та класифікації об'єктів.

Для реалізації програмного продукту використано мову програмування Python, бібліотеки OpenCV, TensorFlow, Ultralytics YOLO, а також методи глибокого навчання. Запропоновані рішення дозволяють здійснювати аналіз відеопотоку в режимі реального часу, автоматично виявляти та класифікувати об’єкти на основі аерофотознімків.

Практичне значення роботи полягає в автоматизації процесу виявлення та класифікації об'єктів на аерофотознімках, що може бути використано для моніторингу територій, військової розвідки, екологічного контролю та інших сфер, де необхідна швидка та точна обробка повітряних зображень.

01.06.2025
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документу	Найменування документу	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	*КвРІПЗ.2101072.01.04.ПЗ*	Пояснювальна записка	82		
2	А4		Завдання на кваліфікаційну роботу	1		
3	А4		Анотація	1		
			Графічні документи			
4	А4	КвРІПЗ.2101072.01.04.Е8	Діаграма класів	1		
5	А4	КвРІПЗ.2101072.01.04.Е8	Діаграма класів	1		
			та інтерфейсів			
6	А4	КвРІПЗ.2101072.01.04.Е8	Діаграма IDEF0	1		
			декомпозиції системи			
9	А4		Презентаційні матеріали	16		

					КвРІПЗ.2101072.01.04.ВД			
Змн.	Арк	№ докум.	Підпис	Дата				
Виконав		Грабков Р.С		01.06	Програмна система для автоматизованого виявлення та класифікації об'єктів на аерофотознімках у режимі реального часу Відомість документів	*Літ.*	*Арк.*	*Аркушів*
Керівник		Бедратюк Л.П.		01.06			1	1
						ХНУ, ІПЗ-21-1		
Н. контр.		Яшина О.М.		01.06				
Зав. каф.		Бедратюк Л. П.		01.06				

ПЕРЕЛІК СКОРОЧЕНЬ

БПЛА	–	безпілотний літальний апарат
БТР	–	бронетранспортер
БМП	–	бойова машина піхоти
ПЗ	–	програмне забезпечення
ПрО	–	предметна область
ШІ	–	штучний інтелект
CLAHE	–	Contrast Limited Adaptive Histogram Equalization (контрастно-обмежена адаптивна еквалізація гістограми)
CNN	–	Convolutional Neural Network (згорткова нейронна мережа)
CV	–	Computer Vision (комп'ютерний зір)
DNN	–	Deep Neural Networks (глибокі нейронні мережі)
FAST	–	Features from Accelerated Segment Test (ознаки з прискореного сегментного тесту)
FPV	–	First Person View (від першої особи)
GPU	–	Graphics Processing Unit (графічний процесор)
HOG	–	Histogram of Oriented Gradients (гістограма орієнтованих градієнтів)
IoU	–	Intersection over Union (перетин над об'єднанням)
MVP	–	Minimum Viable Product (мінімально життєздатний продукт)
R-CNN	–	Region-based Convolutional Neural Network (регіонально-базована згорткова нейронна мережа)
RTSP	–	Real Time Streaming Protocol (протокол потокового передавання в реальному часі)
SDK	–	Software Development Kit (набір засобів розробки)
SIFT	–	Scale-Invariant Feature Transform (масштабно-інваріантне перетворення ознак)
SSD	–	Single Shot Detector (одноетапний детектор)
UML	–	Unified Modeling Language (уніфікована мова моделювання)
YOLO	–	You Only Look Once (алгоритм виявлення об'єктів у реальному часі)

ЗМІСТ

ВСТУП

Актуальність теми кваліфікаційної роботи полягає у критично важливій ролі сучасних технологій автоматизованого аналізу відеопотоків у багатьох галузях, особливо в умовах зростаючого використання безпілотних літальних апаратів для розвідувальних та моніторингових завдань. Розвиток систем комп'ютерного зору дозволяє значно підвищити ефективність обробки великих обсягів візуальної інформації, забезпечуючи швидке та точне виявлення об'єктів інтересу в режимі реального часу.

Практичний досвід застосування дронів у різних сферах діяльності демонструє обмеження людського фактора при тривалому аналізі відеопотоків. Дослідження показують, що після двох годин безперервної роботи активність префронтальної кори операторів знижується на 35%, що призводить до пропуску однієї цілі з десяти виявлених. Автоматизовані системи розпізнавання об'єктів дозволяють підтримувати стабільну ефективність виявлення незалежно від тривалості місії.

У військовій сфері автоматизація процесів розпізнавання стає особливо актуальною в контексті сучасних конфліктів, де кількість одночасно використовуваних дронів постійно зростає. Системи автоматичного розпізнавання об'єктів у відеопотоках стають критично важливим компонентом для забезпечення інформаційної переваги, дозволяючи зменшити вимоги до кваліфікації операторів та скоротити їх когнітивне навантаження.

Мета роботи полягає у розробці програмної системи для автоматизованого виявлення та класифікації об'єктів на аерофотознімках у режимі реального часу з використанням сучасних методів комп'ютерного зору та глибокого навчання.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- провести аналіз предметної області та сучасних методів комп'ютерного зору для класифікації об'єктів на аерофотознімках;
- обґрунтувати вибір алгоритмів і методів обробки відеопотоків з дронів;

– розробити архітектуру програмної системи та створити основні функціональні модулі;

– реалізувати алгоритми виявлення, класифікації та відстеження об'єктів;

– створити зручний інтерфейс користувача для взаємодії з системою;

– провести комплексне тестування програмного забезпечення та оцінити його ефективність.

Об'єкт дослідження – процес автоматизованого розпізнавання та класифікації об'єктів на зображеннях із повітряних зйомок.

Предмет дослідження – алгоритми та програмні засоби для обробки аерофотознімків і класифікації об'єктів у реальному часі з використанням нейронних мереж.

Методологічну основу роботи складають методи машинного навчання та комп'ютерного зору. Зокрема, застосовуються згорткові нейронні мережі архітектури YOLO для виявлення об'єктів, алгоритми відстеження SORT з фільтром Калмана, методи попередньої обробки зображень на основі OpenCV. Також використовуються принципи проєктування програмних систем з монолітною архітектурою та багатопотоковою обробкою даних.

Практичне значення результатів дослідження визначається можливістю застосування створеної системи для автоматизованого виявлення та ідентифікації військової техніки на аерофотознімках. Розроблена система має важливе значення для задач військової розвідки, моніторингу бойових дій, контролю переміщення техніки противника та оцінки загроз у режимі реального часу. Результати роботи дозволяють значно підвищити ефективність аналізу розвідувальних даних, забезпечити швидке прийняття тактичних рішень та суттєво зменшити навантаження на операторів дронів при виконанні розвідувальних місій.

Кваліфікаційна робота має логічну структуру, що включає три основні розділи, висновки та додатки. Перший розділ присвячено аналізу предметної області та постановці задачі, що створює теоретичне підґрунтя дослідження. Другий розділ описує проєктування програмного забезпечення та обґрунтування вибору

технологій. Третій розділ висвітлює програмну реалізацію системи та детальний аналіз результатів тестування.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області

Розвиток технологій повітряного спостереження має багатогранну історію, яка розпочалася з першої аерофотозйомки, виконаної у 1858 році французьким фотографом Гаспаром Феліксом Турнашоном за допомогою повітряної кулі над Парижем [1]. Протягом наступних десятиліть технології безперервно вдосконалювалися: у 1880-х роках для аерофотозйомки використовували повітряні змії, а у 1897 році Альфред Нобель встановив камери на ракети, що дозволило отримувати дані з великих висот. Перша світова війна підкреслила стратегічне значення повітряної розвідки, що призвело до розробки спеціалізованих аерофотокамер. У міжвоєнний період Шерман Фейрчайлд розробив систему кріплення камер на літаках, що стала стандартом для картографічних і військових місій, а у рамках місій Apollo 1960-х років такі технології використовувалися для зйомки поверхні Місяця.

Історія безпілотних літальних апаратів (БПЛА) – літальних апаратів багаторазового використання без екіпажу на борту, що керуються дистанційно або автономно – сягає 1783 року сягає 1783 року, коли було продемонстровано перший безпілотний літальний пристрій. Військове застосування БПЛА вперше відбулося у 1849 році під час бомбардування Венеції за допомогою безпілотних повітряних куль. Сучасний розвиток БПЛА розпочався під час Другої світової війни із запуску крилатих ракет V-1 (Фау-1) націлених на Лондон. Ці апарати були фактично першими безпілотними літальними пристроями, що застосовувалися масово та мали автономну систему наведення. Значний прорив у використанні дронів для військових цілей відбувся у 1970-х роках завдяки ізраїльським розвідувальним безпілотникам, що суттєво підвищили ситуаційну обізнаність військ.

Комерційне використання дронів стало доступним у 2006 році після видачі Федеральним управлінням цивільної авіації США (FAA) перших дозволів на використання БПЛА у цивільних цілях. У 2010 році компанія Parrot представила дрон

AR.Drone, що керувався зі смартфона, а у 2013 році DJI випустила модель Phantom, яка поєднала компактність і високу якість відеозйомки. Компанія Autel Robotics створила серію EVO, яка склала конкуренцію DJI у професійному сегменті. Skydio розробила автономні дрони із системами уникнення перешкод, а PowerVision представила інноваційні моделі для повітряних і підводних зйомок.

Сучасні комерційні квадрокоптери, такі як DJI Mavic 3, забезпечують запис відео з роздільною здатністю до 4K при 60 кадрах на секунду та оснащені 4/3-дюймовим CMOS-сенсором, що дозволяє отримувати високоякісні зображення навіть в умовах низької освітленості. Тривалість польоту таких дронів сягає 46 хвилин, що достатньо для виконання більшості розвідувальних місій. Військові БПЛА, на відміну від комерційних, можуть перебувати в повітрі до 24 годин та оснащуються спеціалізованими тепловізійними камерами з можливістю виявлення об'єктів на відстані до 10 км.

У сучасному світі дрони знаходять широке застосування у різноманітних галузях. В агропромисловому комплексі дрони з мультиспектральними камерами дозволяють відстежувати стан рослин, визначати ділянки з дефіцитом вологи та оптимізувати внесення добрив [2].

Дрони з передовими системами обприскування забезпечують точне застосування агрохімікатів, зменшуючи їх вплив на навколишнє середовище [].

За даними компанії Farmbrite, сільськогосподарські дрони допомагають ідентифікувати шаблони дренажу, виявляти надмірно вологі або сухі ділянки та планувати час ротації випасу [3].

У сфері пошуково-рятувальних операцій тепловізійні дрони забезпечують цілодобовий моніторинг територій, що сприяє суттєвому підвищенню ефективності рятувальних робіт.

У сфері пошуково-рятувальних операцій дрони, оснащені тепловізійними камерами, забезпечують безперервний, цілодобовий моніторинг територій. Ці тепловізійні датчики виявляють теплові сигнали людей, що вижили, навіть в умовах

поганої видимості або вночі, значно підвищуючи швидкість, точність та ефективність рятувальних місій.

Дрони можуть швидко отримати доступ до важкодоступних або небезпечних районів, надаючи повітряні знімки в режимі реального часу і дані, необхідні для координації рятувальних зусиль і виявлення постраждалих або небезпеки. Використання роїв дронів ще більше розширює покриття і надійність, розподіляючи зони пошуку між кількома скоординованими дронами, збільшуючи ймовірність успішних рятувальних операцій і скорочуючи час реагування .

Так, у грудні 2024 року дрон із інфрачервоними камерами виявив 78-річного чоловіка з деменцією, який загубився в чагарниках Малібу, завдяки аналізу теплових сигнатур [4].

В інспекції інфраструктури БПЛА відіграють важливу роль, дозволяючи перевіряти мости, трубопроводи та лінії електропередач, не наражаючи на небезпеку людей-інспекторів. Дрони значно скорочують час інспекції, швидко охоплюючи великі або важкодоступні споруди і надаючи детальні візуальні дані, що допомагає завчасно виявити потреби в технічному обслуговуванні і підвищує безпеку [5].

Наприклад, дрон Flyability Elios 3 проникає через отвори розміром 60×60 см, створюючи 3D-моделі мостів або трубопроводів із точністю до сантиметра [6].

За прогнозами Drone Industry Insights, ринок комерційних дронів зростатиме на 7,9 % в середньорічному обчисленні до 2030 року, при цьому апаратне забезпечення буде визначено як сегмент, що зростає найшвидше [7].

У військовій сфері дрони відіграють ключову роль у веденні сучасних бойових дій. З 2014 року Україна активно використовує безпілотні системи як для розвідки, так і для ударних завдань. Комерційні дрони DJI Mavic і FPV-дрони (First Person View – від першої особи) застосовуються для коригування артилерійського вогню та ураження цілей на полі бою. Особливого значення набули рої FPV-дронів, які забезпечують високоточні удари та відволікають протиповітряну оборону противника.

У багатьох конфліктах також фіксується застосування дронів цивільного сегмента. У Сирії FPV-дрони використовуються з 2016 року для атак і спостереження за пересуванням військ. У Судані та М'янмі комерційні квадрокоптери застосовуються для моніторингу та точкових атак. Ізраїльсько-палестинський конфлікт підтверджує роль БПЛА у міських операціях, де вони забезпечують високоточну навігацію та можливість проведення атак на обмежених територіях.

Попри прогрес у розвитку БПЛА, ефективність виконання завдань у багатьох випадках досі залежить від роботи операторів, які контролюють політ і здійснюють аналіз відеопотоків у реальному часі. Це потребує високого рівня концентрації уваги та створює когнітивне навантаження, особливо під час тривалих місій.

Дослідження показало, що через 2 години роботи активність префронтальної кори операторів знижується на 35%, що призводить до пропуску 1 цілі з 10 [8].

Використання систем комп'ютерного зору (Computer Vision, CV) – галузі штучного інтелекту, що займається розробкою методів і алгоритмів для автоматичного аналізу та розуміння візуальної інформації – дозволяє значно автоматизувати рутинні завдання, зменшуючи навантаження на операторів і залишаючи їм лише прийняття ключових рішень. У цивільних місіях комп'ютерний зір забезпечує автоматичну ідентифікацію проблемних ділянок на полях з негайним повідомленням оператора, здійснює контроль точкового обприскування рослин без необхідності ручного аналізу даних, а також відстежує зміни на об'єктах інфраструктури з формуванням автоматичних звітів [9].

У військових операціях системи комп’ютерного зору дозволяють автоматично розпізнавати танки, артилерійські установки та транспортні засоби за ключовими ознаками, зменшуючи час реакції операторів на виявлені загрози. Наприклад, алгоритми YOLO можуть ідентифікувати об’єкти у реальному часі, обробляючи десятки кадрів за секунду та забезпечуючи підтримку прийняття рішень [10].

Таким чином, комп’ютерний зір виконує роль "асистента", який допомагає операторам зосередитися на критично важливих завданнях і приймати

обґрунтовані рішення. Це дозволяє уникати помилок, пов'язаних із втомою, та підвищує загальну ефективність виконання місій.

1.2 Огляд існуючих програмних рішень для обробки відеопотоків з дронів

Огляд існуючих програмних рішень для автоматичного виявлення та класифікації військової техніки на відеозаписах з дронів ґрунтується на аналізі відкритих джерел інформації, публікацій у спеціалізованих медіа та демонстраційних матеріалів. Дослідження загальнодоступних даних про архітектуру та принципи роботи подібних систем дозволяє окреслити основні підходи до вирішення завдань розпізнавання військової техніки, з’ясувати їх заявлені можливості та виявити існуючі обмеження. Серед рішень, які активно висвітлюються у відкритих джерелах, варто відзначити системи OCHI та Avengers, а також рішення, побудовані на основі NVIDIA DeepStream SDK.

OCHI є цифровою системою, що забезпечує централізований збір відеопотоків з бойових дронів та застосовує алгоритми штучного інтелекту для автоматичного виявлення військової техніки. Система навчена на великій кількості бойових відеоматеріалів, що дозволяє ефективно акумулювати розвідувальні дані та аналізувати тактичні дії противника. За інформацією Reuters, система змогла зібрати величезний обсяг відеоматеріалів з фронту за короткий період, причому щодня надходять значні обсяги нових даних [11]. Основні переваги OCHI полягають у використанні великої навчальної бази, що забезпечує високу точність виявлення об’єктів, можливості інтеграції з існуючими аналітичними платформами та здатності працювати в режимі реального часу. Проте система має й певні обмеження: вона призначена виключно для військових структур із закритим доступом, вимагає значних обчислювальних ресурсів для обробки великих обсягів відеоданих та не підходить для комерційного чи цивільного використання.

Платформа Avengers, розроблена Центром інновацій Міністерства оборони, дозволяє автоматично виявляти ворожу техніку шляхом аналізу відеопотоків із дронів та стаціонарних камер. За даними Міністерства оборони України, завдяки системам на основі ШІ щотижня виявляється до 12 тисяч цілей, що демонструє масштабне застосування аналітики в реальному часі в умовах бойових дій [12]. Крім того, платформа є унікальною за обсягом оброблюваних відеоданих, а її подальше розширення за рахунок хмарних можливостей та інтеграції з безпілотними системами дозволить ще більше покращити ефективність аналізу даних. Серед переваг рішення – високоточне розпізнавання, можливість інтеграції з іншими системами збору розвідувальної інформації та оперативна реакція на зміну бойової ситуації. Однак, як і у випадку з OCHI, відсутність публічно доступної інформації про внутрішню архітектуру алгоритмів та обмежені можливості адаптації до нестандартних завдань поза рамками військової аналітики створюють певні недоліки цього рішення.

NVIDIA DeepStream SDK являє собою універсальний інструмент для обробки потокового відео із застосуванням нейронних мереж, який дозволяє виконувати автоматичне виявлення та класифікацію об’єктів у відеопотоці. Головна ідея даного рішення полягає у створенні високопродуктивних систем, здатних працювати в режимі реального часу завдяки апаратному прискоренню за допомогою GPU. Серед ключових переваг DeepStream – можливість використання різноманітних моделей (наприклад, YOLO, Faster R-CNN, SSD), підтримка розробки власних кастомізованих моделей та високий рівень швидкості обробки відеоданих. Однак для досягнення оптимальної ефективності ця система вимагає потужного обчислювального обладнання, не має вбудованих моделей для специфічного виявлення військової техніки (що потребує додаткового навчання) та характеризується відносно високою складністю налаштування, що може стати перепоною для деяких користувачів [13].

Порівнюючи розглянуті програмні рішення, можна відзначити, що як OCHI, так і Avengers забезпечують високу точність виявлення та здатність до оперативної

роботи, проте їх використання обмежене через закритість доступу та недостатню можливість адаптації під індивідуальні завдання. NVIDIA DeepStream SDK, з іншого боку, є відкритим інструментом із широкими можливостями налаштування і високою продуктивністю, але його ефективність значною мірою залежить від апаратних ресурсів і додаткового навчання спеціалізованих моделей.

З урахуванням вищезазначеного, можна зробити висновок, що навіть при високій ефективності систем OCHI та Avengers їх застосування обмежується через закритість доступу та слабку гнучкість налаштування. Хоча NVIDIA DeepStream SDK демонструє значний потенціал завдяки відкритості та можливостям масштабування, вимоги до високопродуктивного обладнання і необхідність додаткової підготовки моделей створюють певні виклики при його впровадженні у польових умовах. Саме тому актуальною є розробка власного програмного комплексу, орієнтованого на специфічні вимоги військової аналітики, який був би навчений на відповідних об’єктах, оптимізований для роботи на пристроях з обмеженими ресурсами, таких як дрони чи польові комп’ютери, та забезпечував ефективну інтеграцію з існуючими системами збору розвідувальної інформації для своєчасного прийняття оперативних рішень.

1.3 Аналіз сучасних методів класифікації об’єктів на зображеннях

Сучасні методи автоматичного розпізнавання об'єктів у системах комп'ютерного зору ґрунтуються на комплексному підході, що охоплює послідовність взаємопов'язаних етапів обробки візуальної інформації. Розглянемо детально кожен етап процесу розпізнавання об'єктів.

Першим етапом є збір даних, який здійснюється за допомогою камер та відеосенсорів. Критичними факторами на цьому етапі є забезпечення стабільної передачі даних та отримання зображень достатньої якості для подальшої обробки, оскільки від цього безпосередньо залежить ефективність розпізнавання об'єктів

Наступним етапом є попередня обробка зображень, яка спрямована на покращення якості вхідних даних. Цей процес включає декілька ключових методів: алгоритми стабілізації відеопотоку, які компенсують небажані рухи камери; методи шумопоглинання, що забезпечують очищення зображення від випадкових спотворень; алгоритми нормалізації контрастності, які оптимізують діапазон яскравості; методи геометричної корекції для компенсації оптичних спотворень об'єктива. Попередня обробка є критичним етапом, оскільки якість її виконання безпосередньо впливає на точність подальшого розпізнавання.

Етап виявлення об'єктів є фундаментальним у процесі розпізнавання. На цьому етапі система здійснює локалізацію потенційних областей інтересу у зображенні. Сучасні методи виявлення використовують складні математичні моделі та алгоритми машинного навчання для аналізу візуальних характеристик та виділення регіонів, які можуть містити цільові об'єкти. Точність локалізації об'єктів на цьому етапі є визначальною для успішності подальшої класифікації.

Процес класифікації є наступним етапом, на якому здійснюється віднесення виявлених об'єктів до певних категорій. Сучасні методи класифікації базуються на використанні глибоких нейронних мереж (DNN), зокрема згорткових нейронних мереж (CNN). Ці системи здатні автоматично вивчати ієрархічні ознаки зображень та формувати складні моделі для розпізнавання об'єктів, що забезпечує високу точність класифікації навіть у складних умовах [14].

Завершальним етапом є відстеження об'єктів, що забезпечує аналіз траєкторій руху виявлених об'єктів та прогнозування їх подальших переміщень. Для реалізації цього етапу застосовуються спеціалізовані алгоритми трекінгу, такі як фільтр Калмана або більш сучасні підходи, засновані на глибокому навчанні. Ці алгоритми забезпечують стабільне відстеження об'єктів між кадрами відеопотоку, що є критично важливим для систем реального часу.

Особливу увагу слід приділити специфіці аналізу відеопотоків, отриманих з FPV-дронів та комерційних квадрокоптерів. На відміну від статичних аерофотознімків, відео з дронів характеризується динамічними умовами зйомки, що створює

додаткові виклики для систем розпізнавання. По-перше, камера на дроні перебуває в постійному русі, що призводить до зміни ракурсів, швидкого переміщення об'єктів у кадрі та потенційних проблем із стабільністю зображення. По-друге, FPV-дрони часто експлуатуються на низьких висотах і з високою швидкістю польоту, що спричиняє значне розмиття зображення та різкі зміни перспективи. По-третє, якість відеопотоку з дронів часто обмежена пропускною здатністю каналу передачі даних, що зумовлює компресію та зниження роздільної здатності зображення. Крім того, погодні умови, такі як туман, дощ або димові завіси, можуть суттєво впливати на видимість і ускладнювати процес розпізнавання. Ці фактори вимагають розробки спеціалізованих алгоритмів, здатних працювати з динамічними відеопотоками та адаптуватися до різних умов зйомки в реальному часі.

Кожен з описаних етапів має свої особливості та вимоги до обчислювальних ресурсів, а їх ефективна інтеграція є ключовим фактором для створення надійної системи розпізнавання об'єктів. Сучасні тенденції розвитку методів комп'ютерного зору свідчать про перехід до все більш складних моделей машинного навчання, які здатні забезпечити вищу точність та надійність розпізнавання в різноманітних умовах застосування.

1.4 Вибір методів і підходів для вирішення задачі

У сучасній практиці автоматичного розпізнавання об'єктів виділяють два основних підходи: класичні методи комп'ютерного зору та методи, засновані на глибоких нейронних мережах. Кожен з цих підходів має свої характерні особливості та сфери застосування.

До класичних методів комп'ютерного зору відносяться SIFT, HOG і каскади Хаара. SIFT (Scale-Invariant Feature Transform) – це алгоритм виявлення та опису локальних ознак зображення, який забезпечує інваріантність до змін масштабу, повороту та освітлення завдяки виявленню характерних ключових точок. Ключові точки – це унікальні особливості зображення, які можна надійно виявити незалежно від змін умов зйомки. Основними перевагами SIFT є стабільність при зміні параметрів зображення, проте висока обчислювальна складність та повільна робота з великими потоками даних суттєво обмежують його застосування в системах реального часу [15].

HOG (Histogram of Oriented Gradients) – метод опису візуальних особливостей об'єктів, що базується на підрахунку розподілу градієнтів інтенсивності у локальних ділянках зображення. Градієнт інтенсивності – це вектор, що вказує напрямок найбільшої зміни яскравості пікселів зображення. Цей метод особливо ефективний для опису контурів і форм об'єктів. Перевагами HOG є відносна простота реалізації та помірні вимоги до обчислювальних ресурсів. Проте, метод демонструє недостатню стійкість при змінах умов зйомки та не забезпечує необхідну точність у динамічних сценах [16].

Каскади Хаара (Haar Cascades) – це метод машинного навчання, що використовує прості ознаки типу Haar-like features у поєднанні з каскадним класифікатором. Haar-like features – це цифрові ознаки зображення, які обчислюються як різниця сум пікселів у суміжних прямокутних областях. Каскадний класифікатор являє собою послідовність простих класифікаторів, де кожен наступний працює лише

з тими областями зображення, які пройшли попередні етапи класифікації. Цей підхід дозволяє швидко відфільтровувати нерелевантні області зображення. Головними перевагами є висока швидкість обчислень та економність використання ресурсів. Однак, метод характеризується низькою точністю при складних фонових умовах та різких змінах перспективи зйомки [17].

З іншого боку, сучасні методи, засновані на глибоких нейронних мережах (Deep Neural Networks, DNN) – це складні математичні моделі, що складаються з багатьох шарів штучних нейронів, здатних до навчання на основі великих наборів даних. Зокрема, згорткові нейронні мережі (Convolutional Neural Networks, CNN) – це спеціалізований тип нейронних мереж, розроблений для ефективної обробки сітчастих даних, таких як зображення. CNN здатні автоматично навчатися розпізнавати складні ознаки із зображень завдяки багатошаровій архітектурі.

Серед сучасних архітектур глибоких нейронних мереж для виявлення об'єктів особливо виділяються YOLO (You Only Look Once), Faster R-CNN (Region-based Convolutional Neural Network) та SSD (Single Shot Detector). YOLO – це однорівнева архітектура виявлення об'єктів, яка розділяє зображення на сітку та виконує одночасне передбачення обмежувальних рамок та класів об'єктів для кожної комірки сітки. Ця архітектура забезпечує високу швидкість обробки (до 45 кадрів на секунду) при збереженні достатньої точності виявлення. Faster R-CNN використовує двоетапний підхід: спочатку мережа регіонних пропозицій (Region Proposal Network, RPN) генерує потенційні області розташування об'єктів, а потім виконується їх класифікація. Цей метод забезпечує високу точність, але працює повільніше за YOLO. SSD також є однорівневим детектором, який використовує багатомасштабні карти ознак для виявлення об'єктів різного розміру, забезпечуючи баланс між швидкістю та точністю [18].

Методи на основі глибоких нейронних мереж характеризуються високою точністю виявлення та класифікації об'єктів, стійкістю до змін ракурсу, освітлення та наявності шумів, а також можливістю обробки даних у режимі реального часу при використанні апаратного прискорення. Проте, вони мають і суттєві недоліки:

високу залежність від великої кількості навчальних даних, значні вимоги до обчислювальних ресурсів під час навчання та складність налаштування й інтеграції таких систем у вже існуючу інфраструктуру.

Порівнюючи обидва підходи, можна відзначити, що класичні методи (SIFT, HOG, каскади Хаара) мають переваги у плані простоти реалізації та економії обчислювальних ресурсів, проте вони виявляються менш ефективними в умовах змінних умов зйомки, динамічних сцен та високого рівня шумів. На противагу їм, методи на основі глибоких нейронних мереж демонструють значно вищу точність і стійкість, що є критично важливим для аналізу відео з дронів, хоча й потребують більше ресурсів і даних для навчання.

Вибір оптимального методу для системи розпізнавання базується на комплексному аналізі вимог та характеристик доступних методів. Основними критеріями вибору є швидкість обробки в реальному часі, точність виявлення та класифікації, стійкість до змін умов зйомки, ефективність при роботі з об'єктами різного масштабу та обчислювальні вимоги. При порівнянні сучасних архітектур Faster R-CNN забезпечує найвищу точність виявлення, але має відносно низьку швидкість обробки, що є критичним для систем реального часу. SSD демонструє середні показники як по швидкості, так і по точності, але має обмеження при виявленні малих об'єктів, що є суттєвим недоліком при аналізі аерофотознімків. YOLO забезпечує оптимальний баланс між швидкістю та точністю. Архітектура має наступні переваги: висока швидкість обробки для аналізу відеопотоку в реальному часі, ефективна робота з об'єктами різного масштабу, стійкість до змін умов освітлення та ракурсу зйомки, можливість оптимізації для роботи на мобільних системах.

Додатковими технічними перевагами YOLO в контексті обробки відеопотоків з дронів є здатність алгоритму ефективно працювати з динамічними сценами. Алгоритм демонструє високу стійкість до зміни ракурсів зйомки, що особливо важливо при аналізі відео з FPV-дронів, які характеризуються швидкими маневрами та різкими поворотами камери. Варто відзначити, що останні версії YOLO включають удосконалені механізми темпоральної узгодженості, що дозволяє підвищити

стабільність розпізнавання об'єктів у відеопотоках з високою частотою кадрів. Особливо важливою є здатність YOLO розпізнавати об'єкти при частковому перекритті та в умовах динамічного фону, що характерно для зйомки з дронів у бойових умовах. Крім того, архітектура YOLO дозволяє здійснювати ефективне балансування між точністю та швидкодією шляхом налаштування моделі, що критично важливо для забезпечення аналізу відеопотоку в режимі реального часу з урахуванням обмежених обчислювальних ресурсів у польових умовах.

На основі всебічного аналізу методів комп'ютерного зору YOLO (You Only Look Once) обрано як оптимальне рішення для виявлення та класифікації об'єктів у відеопотоках з дронів. Вибір обґрунтовано кількома ключовими перевагами: однорівнева архітектура YOLO забезпечує високу швидкодію, що критично важливо для обробки відео в реальному часі; новітні версії алгоритму демонструють високу точність розпізнавання навіть при складних умовах зйомки та швидких змінах ракурсу; модульна архітектура дозволяє гнучко балансувати між швидкістю та точністю; алгоритм ефективно працює при виявленні об'єктів різного масштабу. Для підвищення надійності системи запропоновано комбінований підхід з використанням ансамблю моделей, де YOLO функціонує як основний детектор, а додаткові легковагі моделі застосовуються для верифікації результатів у складних випадках. Така архітектура забезпечує оптимальний баланс між обчислювальною ефективністю, точністю класифікації та адаптивністю до різних умов експлуатації.

1.5 Визначення функціональних та нефункціональних вимог до програмного забезпечення

Після проведеного аналізу предметної області та існуючих рішень сформульовано основні вимоги до розробки мінімально життєздатного продукту (MVP) для автоматичного розпізнавання та класифікації об’єктів за допомогою аналізу відеопотоків. Ці вимоги поділяються на функціональні, які описують базовий набір завдань, що має виконувати система, та нефункціональні, що регламентують умови експлуатації програмного забезпечення, незалежно від деталей його реалізації.

1.5.1 Функціональні вимоги

Програмна система повинна забезпечувати прийом відеопотоків із джерел, підтримуючи формати MP4, AVI, а також можливість роботи з прямими трансляціями за протоколом RTSP; здійснювати попередню обробку отриманих зображень шляхом усунення шумів, стабілізації кадрів та корекції контрастності з метою покращення якості вхідних даних; автоматично локалізувати потенційні області, де можуть перебувати об’єкти інтересу, та класифікувати їх за заздалегідь визначеними категоріями (наприклад, танк; БТР; броньований автомобіль); забезпечувати опціональну функцію відстеження для ідентифікації одного і того ж об’єкта на різних кадрах відео; надавати можливість візуалізації результатів шляхом накладення інформаційних підписів на відеозапис, а також здійснювати збереження та логування даних для подальшого аналізу.

1.5.2 Нефункціональні вимоги

Програмна система розрахована на дві ключові категорії користувачів:

1) оператори FPV-дронів – фахівці, які безпосередньо керують польотами, здійснюють відеозйомку та аналіз цілей у реальному часі;
2) аналітики/координатори – ті, хто працюють у парі з операторами, інтегрують отримані дані у системи керування боєм та приймають рішення на основі автоматичної класифікації.

Програмна система повинна відповідати встановленим критеріям продуктивності, сумісності, зручності користування та безпеки. Зокрема, час обробки одного кадру не повинен перевищувати 100 мс, а система має працювати у режимі реального часу з частотою не менше 10 кадрів за секунду; програма повинна бути сумісною з операційними системами Windows 10 та Windows 11, забезпечуючи стабільну роботу на цих платформах; інтерфейс користувача має бути простим і інтуїтивно зрозумілим, що сприятиме легкому освоєнню і ефективній експлуатації системи; також програмна система повинна мати можливість легкої модифікації – заміни алгоритмів виявлення або розширення переліку категорій об’єктів без необхідності повної перебудови архітектури; забезпечення безпеки даних досягається шляхом використання захищених протоколів зберігання та логування інформації; крім того, система повинна бути масштабованою, що дозволить інтегрувати її з іншими аналітичними рішеннями для подальшого розширення функціоналу.

Таким чином, визначення функціональних та нефункціональних вимог дозволяє чітко окреслити, що саме повинна виконувати програмна система, а також встановити критерії її ефективної, безпечної та зручної експлуатації, незалежно від обраних методів реалізації.

1.6 Постановка задачі

У цьому розділі кваліфікаційної роботи було сформульовано основну мету проєкту та визначено завдання, які необхідно вирішити для розробки програмного забезпечення, що автоматично розпізнає та класифікує об’єкти за допомогою аналізу відеопотоків. Попередні частини роботи містили аналіз предметної області та огляд існуючих програмних рішень, завдяки чому було визначено функціональні та нефункціональні вимоги до системи.

Основною метою є створення ефективного, надійного та зручного у використанні програмного забезпечення, яке дозволятиме обробляти відеодані в режимі реального часу, виконувати детекцію, класифікацію та відстеження об’єктів, а також забезпечуватиме простий інтерфейс користувача та сумісність з платформами Windows 10 та Windows 11.

Подальші етапи передбачають розробку архітектури системи, проектування зручного інтерфейсу, реалізацію модулів прийому відеопотоків, попередньої обробки зображень, виявлення, класифікації та відстеження об’єктів, а також проведення тестування для перевірки продуктивності, сумісності та безпеки розробленого рішення.

Отже, аналіз проведених досліджень дозволив чітко окреслити вимоги до програмного забезпечення. Основним завданням є створення системи для автоматичного розпізнавання та класифікації об’єктів у режимі реального часу, що стане основою для подальшого проєктування, реалізації та тестування програмного забезпечення.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Архітектура програмної системи

При проєктуванні системи для автоматизованого виявлення та класифікації об'єктів на аерофотознімках важливо розглянути основні архітектурні підходи, що застосовуються для подібних завдань, та обрати оптимальне рішення з урахуванням конкретних вимог до системи.

Монолітна архітектура представляє собою цілісну систему, де всі функціональні компоненти тісно інтегровані в єдиному програмному модулі. Основними перевагами цього підходу є: висока швидкість обміну даними між компонентами завдяки використанню спільної пам'яті; спрощений процес розгортання та налаштування системи; ефективне використання обчислювальних ресурсів. Однак монолітна архітектура має і суттєві недоліки: обмежена масштабованість; складність внесення змін у окремі компоненти; обмежені можливості для горизонтального масштабування на кластерах серверів.

Мікросервісна архітектура базується на розподілі функціональності системи на невеликі незалежні сервіси, кожен з яких відповідає за конкретну задачу (наприклад, отримання відео, детекція об'єктів, класифікація). Переваги такого підходу включають: високу гнучкість у розгортанні та оновленні окремих компонентів; можливість горизонтального масштабування під навантаженням; ізоляцію помилок в окремих сервісах. До недоліків можна віднести: підвищену складність комунікації між сервісами з потенційними затримками; додаткові витрати на серіалізацію/десеріалізацію даних; більш складну інфраструктуру для розгортання та моніторингу [19].

Клієнт-серверна архітектура розділяє систему на дві основні частини: клієнт (інтерфейс користувача) та сервер (логіка обробки даних). Перевагами є: можливість розподілу обчислювального навантаження між клієнтом і сервером;

централізоване зберігання та обробка даних; підтримка одночасної роботи кількох клієнтів.

Недоліками виступають: залежність від стабільності мережевого з'єднання; потенційні затримки при передачі відеопотоків великого обсягу; обмеження функціональності у випадку втрати зв'язку з сервером

Гібридна архітектура з edge computing поєднує локальну обробку даних на пристрої збору (дрон, камера) з подальшою передачею результатів на центральний сервер для глибшого аналізу. Переваги: зниження обсягу даних, що передаються по мережі; можливість функціонування в умовах нестабільного з'єднання; розподіл обчислювального навантаження. Недоліки: складність синхронізації між компонентами; потреба в додатковому обладнанні на пристроях збору даних; більш складна розробка та тестування.

Для систематичного порівняння розглянутих архітектурних підходів у контексті специфічних вимог системи автоматизованого виявлення об'єктів на аерофотознімках, наведено порівняльну таблицю:

Таблиця 2.1 – Порівняння архітектурних підходів для системи виявлення об'єктів

Критерій оцінки	Монолітна	Мікросервісна	Клієнт-серверна	Гібридна з edge computing
Швидкодія (затримка обробки)	Відмінна (<10 мс)	Задовільна (50-200 мс)	Добра (20-100 мс)	Добра (15-50 мс)
Складність розгортання	Низька (простий інсталятор)	Висока (Docker/K8s)	Середня (клієнт+сервер)	Висока (багато компонентів)
Автономність роботи	Повна (незалежна)	Обмежена (залежність від мережі)	Обмежена (потребує сервера)	Часткова (адаптивна)

Продовження таблиці 2.1

Масштабованість	Обмежена (вертикальна)	Відмінна (горизонтальна)	Середня (обмежена)	Добра (гнучка)
Надійність у польових умовах	Висока (стабільна)	Низька (залежна від інфраструктури)	Низька (залежна від мережі)	Середня (адаптивна)
Використання ресурсів	Оптимальне (ефективне)	Надмірне (додаткові витрати)	Збалансоване (розподілене)	Збалансоване (адаптивне)
Підтримка real-time (≥10 FPS)	Відмінна (native)	Складна (проблематична)	Середня (залежить від мережі)	Добра (можлива)
Складність розробки	Низька (проста)	Висока (складна)	Середня (помірна)	Висока (складна)

На основі аналізу функціональних та нефункціональних вимог до системи, а також з урахуванням особливостей обробки відеопотоків у реальному часі, для нашої програмної системи обрано монолітну архітектуру з елементами багатопотоковості.

Цей вибір обумовлений наступними факторами:

а) критичні фактори вибору:

1) вимога до швидкодії в реальному часі – монолітна архітектура забезпечує мінімальні затримки при передачі даних між компонентами системи, що критично важливо для обробки відеопотоку з частотою не менше 10 кадрів за секунду;
2) умови експлуатації – система може застосовуватися в польових умовах, де розгортання складної розподіленої архітектури є проблематичним;
3) обмеження обчислювальних ресурсів – для роботи в режимі реального часу важливо ефективно використовувати наявні ресурси, уникаючи додаткових витрат на міжпроцесну комунікацію;
4) автономність функціонування – система має забезпечувати повноцінну роботу на одному комп'ютері без необхідності підключення до зовнішніх серверів;

5) спрощення інтеграції з алгоритмами комп'ютерного зору – можливість ефективно використовувати спільні ресурси для виконання алгоритмів машинного навчання;

б) вдосконалення монолітної архітектури в системі:

1) багатопотокова обробка – компоненти системи виконуються у окремих потоках, що дозволяє ефективно використовувати багатоядерні процесори;
2) буферизовані черги даних – для обміну інформацією між компонентами використовуються буферизовані черги, які запобігають блокуванню системи;
3) модульна структура коду – код організовано у вигляді окремих модулів з чітко визначеними інтерфейсами.

Загальна структура розробленої системи представляє собою конвеєр обробки даних, що складається з п'яти основних компонентів, які працюють послідовно:

а) модуль отримання відеопотоку:

1) прийом відеоданих з різних джерел (файли, RTSP-потоки, USB-камери); декодування стиснутого відео;
2) буферизація кадрів;
3) забезпечення стабільної частоти подачі зображень.

б) модуль препроцесингу зображень:

1) стабілізація зображення для компенсації руху камери;
2) корекція контрасту та яскравості;
3) шумозаглушення для покращення якості зображення;
4) геометричні перетворення, адаптовані для аерофотознімків.

в) модуль виявлення та класифікації об'єктів:локалізація об'єктів на зображенні за допомогою алгоритму YOLO:

1) класифікація виявлених об'єктів за заданими категоріями;
2) фільтрація результатів за рівнем достовірності;
3) адаптація параметрів моделі для конкретних умов застосування.

г) модуль відстеження об'єктів:

1) зіставлення виявлених об'єктів між послідовними кадрами;
2) присвоєння унікальних ідентифікаторів об'єктам;
3) формування траєкторій руху об'єктів;
4) підвищення стабільності класифікації за рахунок часової узгодженості.

д) модуль візуалізації та інтерфейсу користувача:

1) накладання графічних міток на відеопотік;генерація текстових повідомлень про виявлені об'єкти;
2) формування статистичної інформації;
3) забезпечення взаємодії користувача з системою.

Взаємодія між компонентами реалізована через механізм буферизованих черг, що дозволяє асинхронно обробляти дані та оптимізувати використання обчислювальних ресурсів. Кожен модуль працює у власному потоці виконання, що забезпечує ефективне використання багатоядерних процесорів та запобігає блокуванню системи при тривалих операціях.

Обрана архітектура дозволяє ефективно вирішувати поставлені задачі в рамках визначених функціональних та нефункціональних вимог, забезпечуючи роботу системи в режимі реального часу на обладнанні середньої потужності. При цьому модульний підхід до організації коду та чітко визначені інтерфейси між компонентами залишають можливість для подальшого розвитку системи та її адаптації до нових вимог.

2.2 Вибір середовища розробки та технологій

У межах даної роботи розробляється мінімально життєздатний продукт (MVP) для системи автоматизованого виявлення та класифікації об’єктів. Основна увага приділяється використанню перевірених, доступних та продуктивних

технологій, які дозволяють швидко реалізувати ключову функціональність системи з можливістю її подальшого розширення.

Для реалізації системи розглядалося декілька мов програмування. C++ традиційно вважається стандартом для високопродуктивних систем комп'ютерного зору завдяки прямому управлінню пам'яттю та мінімальним накладним витратам. Ця мова забезпечує максимальну швидкодію та дозволяє оптимізувати критичні ділянки коду на низькому рівні. Проте C++ має значно вищий поріг входження, потребує більше часу на розробку через ручне управління пам'яттю та має обмежену екосистему готових рішень для машинного навчання.

Java розглядалася як альтернатива завдяки її кросплатформності та надійності. Автоматичне управління пам'яттю та розвинена екосистема роблять Java привабливою для корпоративних рішень. Однак для задач комп'ютерного зору Java має обмежену підтримку спеціалізованих бібліотек, а її продуктивність поступається компільованим мовам через використання віртуальної машини.

Rust як сучасна системна мова програмування поєднує високу продуктивність C++ з безпекою пам'яті. Система власності Rust унеможливлює багато типових помилок на етапі компіляції. Проте екосистема Rust для комп'ютерного зору та машинного навчання ще перебуває на ранній стадії розвитку, що значно обмежує вибір готових рішень.

Основною мовою програмування обрано Python, незважаючи на його відносно нижчу продуктивність порівняно з компільованими мовами. Цей вибір обумовлений унікальною комбінацією факторів: Python має найбагатшу екосистему бібліотек для комп'ютерного зору та глибокого навчання, включаючи NumPy, SciPy, scikit-learn та TensorFlow. Критичні за продуктивністю компоненти цих бібліотек реалізовані на C/C++, що нівелює недоліки інтерпретованої мови. Простота синтаксису Python дозволяє швидко прототипувати та експериментувати з різними підходами, що критично важливо для MVP [21].

Для роботи з відеопотоками та обробки зображень розглядалося кілька спеціалізованих бібліотек. ImageAI представляє собою високорівневу бібліотеку з

простим API для базових задач комп'ютерного зору. Вона забезпечує швидкий старт для початківців, але має обмежені можливості для тонкого налаштування алгоритмів та не підходить для складних задач обробки відео в реальному часі.

Pillow (PIL) є стандартом де-факто для базової обробки зображень в Python. Бібліотека відмінно справляється з операціями над статичними зображеннями, але не призначена для роботи з відеопотоками та не має вбудованих алгоритмів комп'ютерного зору.

SimpleCV позиціонується як спрощена альтернатива OpenCV з більш інтуїтивним інтерфейсом. Проте проект має обмежену підтримку спільноти, рідкі оновлення та значно меншу функціональність порівняно з повноцінними рішеннями.

Для нашої системи обрано OpenCV як найбільш зрілу та функціональну бібліотеку комп'ютерного зору. OpenCV забезпечує ефективну роботу з різними джерелами відео, включаючи файли, RTSP-потоки та USB-камери. Бібліотека містить оптимізовані реалізації алгоритмів стабілізації зображення, корекції кольору та геометричних перетворень. Важливою перевагою є наявність GPU-прискорення через CUDA та широка підтримка різних платформ. Десятиліття розвитку забезпечили OpenCV величезну базу документації та активну спільноту розробників [22].

Вибір моделі для виявлення об'єктів є критичним для продуктивності системи. R-CNN та її еволюції (Fast R-CNN, Faster R-CNN) були піонерами в області глибокого навчання для детекції об'єктів. Ці двоетапні моделі забезпечують високу точність виявлення, але їх швидкість обробки недостатня для роботи в реальному часі. Faster R-CNN може обробляти лише 5-7 кадрів за секунду на сучасному GPU, що не відповідає нашим вимогам.

SSD (Single Shot Detector) представляє одноетапний підхід до виявлення об'єктів. Модель досягає балансу між швидкістю та точністю, обробляючи до 20-30 FPS. Проте SSD має складнощі з виявленням малих об'єктів, що критично для аналізу аерофотознімків, де об'єкти часто займають невелику частину кадру.

RetinaNet вирішує проблему дисбалансу класів через інноваційну функцію втрат Focal Loss. Модель показує відмінні результати на складних датасетах, але її

швидкодія обмежена 10-15 FPS на GPU середнього рівня. Для нашої системи це межове значення, яке не залишає запасу продуктивності.

EfficientDet оптимізує співвідношення точності та обчислювальних витрат через масштабовану архітектуру. Модель демонструє state-of-the-art результати при менших обчислювальних витратах, але потребує складного налаштування для конкретних задач та має вищий поріг входження для розробників [23].

Ключовим компонентом системи обрано модель YOLOv8 від Ultralytics. YOLO (You Only Look Once) представляє сімейство одноетапних детекторів, оптимізованих для роботи в реальному часі. YOLOv8 забезпечує обробку 30-60 FPS на сучасних GPU при збереженні високої точності виявлення. Модель має ефективну архітектуру для виявлення об'єктів різних розмірів, що важливо для аерофотознімків. Ultralytics надає зручний Python API та інструменти для швидкого донавчання моделі на власних даних. Важливою перевагою є активна підтримка та регулярні оновлення від розробників [24].

Модель навчається у середовищі Google Colab — хмарному сервісі для виконання Python-коду з підтримкою апаратного прискорення за допомогою GPU та CUDA, що забезпечує високу продуктивність при обробці зображень. Навчання виконується на основі кастомного датасету, створеного вручну в Label Studio — зручному інструменті для розмітки даних, який дозволяє точно позначати об’єкти на зображеннях. Такий підхід забезпечує адаптацію моделі до специфіки цільового застосування та підвищення її точності в реальних умовах [25].

Інференс моделі виконується локально на GPU (за наявності) або CPU, що забезпечується вбудованими механізмами Ultralytics з підтримкою CUDA. Візуалізація результатів здійснюється за допомогою OpenCV.

Щодо побудови графічного інтерфейсу, обрано Qt — кросплатформний фреймворк, який забезпечує адаптивний інтерфейс, високу продуктивність та зручність у розробці. У майбутньому також розглядається можливість використання альтернативних рішень, таких як Electron або WPF, для розширення функціональності та підтримки інших платформ.

Отже, вибір технологічного стека спрямований на забезпечення швидкого прототипування та створення стабільного MVP, який може бути надалі масштабований і оптимізований за необхідності.

2.3 Проєктування інтерфейсу користувача

Проєктування інтерфейсу користувача для розроблюваної системи MVP здійснюється з метою забезпечення максимальної зручності та ефективності взаємодії оператора з програмним продуктом. Інтерфейс має бути простим, інтуїтивно зрозумілим та функціонально насиченим, що дозволяє оперативно контролювати процес розпізнавання та відстеження об’єктів у режимі реального часу.

Основний функціонал інтерфейсу включає можливість вибору джерела відеопотоку, що може бути як локальним файлом, так і потоковою трансляцією через RTSP або пристроєм захоплення, наприклад, USB-камерою. За допомогою інтегрованих засобів OpenCV здійснюється відображення відеопотоку з накладенням результатів роботи алгоритмів, таких як обмежувальні рамки, текстові підписи та інші графічні елементи, що інформують про класифікацію та позицію об’єктів.

Проєктування інтерфейсу враховує можливість налаштування параметрів роботи системи. Оператор може регулювати пороги детекції, параметри стабілізації та інші ключові налаштування через відповідну панель, що забезпечує адаптивність системи до змін умов зйомки. Крім того, передбачено інтеграцію блоків статистики та аналітики, що дозволяє в режимі реального часу отримувати інформацію про кількість та тип виявлених об’єктів, а також експортувати результати роботи системи у формати, придатні для подальшого аналізу.

2.4 Опис основних компонентів системи

Система автоматизованого виявлення та класифікації об'єктів на аерофотознімках складається з п'яти взаємопов'язаних компонентів, кожен з яких виконує специфічні функції в загальному конвеєрі обробки даних. Розглянемо детально кожен компонент, його призначення, внутрішню структуру та особливості реалізації.

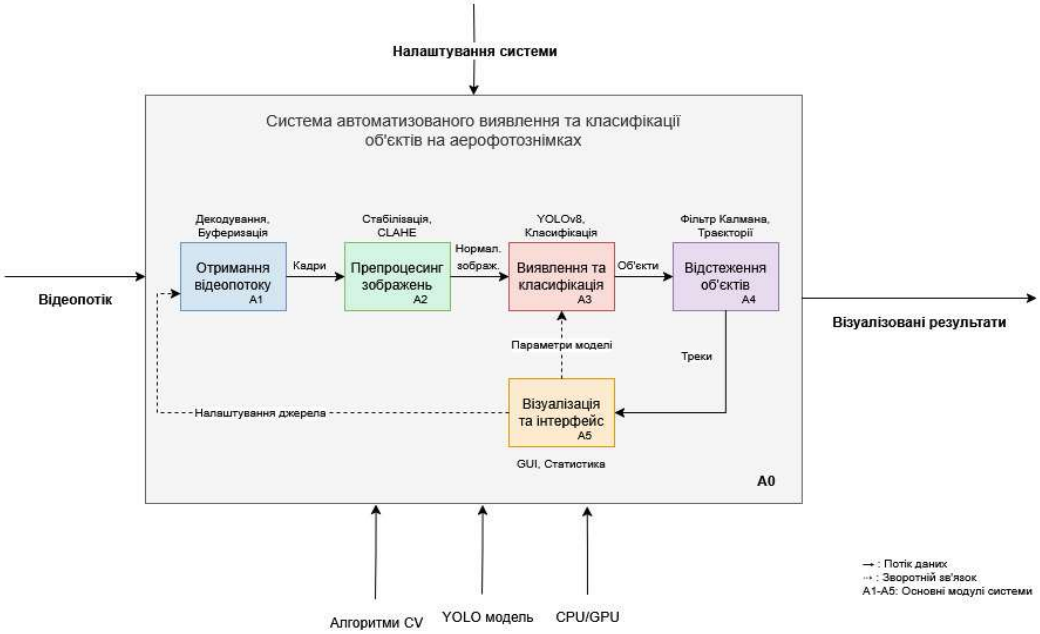


Рисунок 2.1 – Загальна архітектура системи автоматизованого виявлення та класифікації об'єктів на аерофотознімках

Як показано на рисунку 2.1, система побудована за принципом конвеєрної обробки даних, де кожен модуль отримує вхідні дані від попереднього та передає результати наступному. Взаємодія між компонентами реалізована через буферизовані черги, що забезпечує асинхронну обробку та ефективне використання обчислювальних ресурсів. Розглянемо детально кожен компонент системи.

2.4.2 Модуль отримання відеопотоку

Модуль отримання відеопотоку розглядається як вхідна точка системи, що має забезпечувати взаємодію з джерелами відеоданих, їхню первинну обробку та буферизацію. Його основним завданням є приведення потоків різного формату до уніфікованого вигляду для подальшого аналізу іншими модулями.

Очікується, що модуль підтримуватиме кілька типів джерел, зокрема локальні відеофайли (MP4, AVI, MOV), потокові трансляції через RTSP, а також відео з USB-камер. Для кожного джерела має бути передбачено окремий адаптер, що забезпечить зчитування та попереднє перетворення відео. Попередня обробка передбачатиме декодування стиснутих відеопотоків, перетворення кольорового простору відповідно до вимог алгоритмів, а також зміну роздільної здатності кадрів.

Буферизація даних має здійснюватися з використанням циклічного буфера з можливістю адаптації до поточної частоти кадрів. Також передбачається реалізація механізму пропуску кадрів та часової синхронізації для збереження послідовності подачі зображень.

На рівні архітектури проєктується абстрактний менеджер джерел відео, який визначає уніфікований інтерфейс для роботи з усіма типами джерел. Спеціалізовані декодери відповідатимуть за обробку відео різних форматів. Потокобезпечна структура буфера дозволить накопичувати оброблені кадри без втрати даних. Контролер якості, у свою чергу, забезпечуватиме моніторинг параметрів відео та підлаштування системи до поточних умов.

Для реалізації модульної логіки планується використати бібліотеку OpenCV як основу для обробки відео. Можливе застосування апаратного прискорення на етапі декодування. Архітектура модуля також має враховувати багатопотоковий режим, де процеси отримання й обробки кадрів виконуються паралельно. У проєкт також закладається можливість адаптивної обробки помилок для стабільної роботи за умов нестабільного з’єднання або втрати сигналу.

З огляду на роль цього модуля в архітектурі системи, його функціональна надійність прямо впливатиме на якість вхідних даних, а отже, і на кінцеву ефективність виявлення та класифікації об’єктів.

2.4.3 Модуль препроцесингу зображень

Модуль препроцесингу зображень передбачено використовувати для підготовки відеокадрів до обробки алгоритмами комп’ютерного зору. Його завдання полягає в покращенні візуальних характеристик зображень, компенсації артефактів зйомки та оптимізації даних для подальшого аналізу.

Очікується, що цей модуль міститиме підсистеми стабілізації зображення, які аналізують рух камери й компенсують вібрації та ефект «rolling shutter». Передбачено використання методів згладжування різких змін положення кадру, що особливо актуально для зйомки з дронів. Крім того, буде реалізовано корекцію контрасту, яскравості та кольору з урахуванням змін освітлення або атмосферних впливів.

Для боротьби з шумами планується використання фільтрів, здатних знижувати спотворення, водночас зберігаючи важливі структури, зокрема краї об'єктів. Геометричні перетворення, зокрема масштабування, корекція дисторсії та перспективна нормалізація, дадуть змогу адаптувати зображення до вимог нейронної мережі.

Модуль включатиме конвеєр обробки, який застосовуватиме визначений набір операцій до кожного кадру. Контролер, що працює в адаптивному режимі, буде відповідати за вибір оптимальних параметрів фільтрації, залежно від характеристик конкретного відеопотоку. Функціональність модуля має базуватися на використанні бібліотеки OpenCV, з можливим залученням GPU для пришвидшення ресурсомістких операцій.

Передбачається також кешування проміжних результатів, що дозволить уникнути повторного виконання обчислень при обробці подібних кадрів. Загалом,

модуль препроцесингу має покращити якість вхідних даних і забезпечити стабільність роботи в умовах низької видимості, швидкого руху та змінного освітлення.

2.5 Алгоритмічне наповнення модулів

На основі аналізу предметної області та обґрунтування вибору архітектури YOLO, представленого в розділі 1.4, в даному розділі конкретизуємо вибір алгоритмів для кожного етапу обробки даних та технічні аспекти їх реалізації.

2.5.1 Алгоритми попередньої обробки зображень

На етапі попередньої обробки зображень передбачається використання класичних алгоритмів із бібліотеки OpenCV, що дозволяють покращити якість вхідних кадрів перед передачею в модель виявлення об’єктів.

Для стабілізації зображення обрано підхід на основі виявлення та відстеження ключових точок. Ключові точки виявляються за допомогою алгоритму FAST (Features from Accelerated Segment Test), що відзначається високою швидкодією і стійкістю до шумів. Подальше відстеження між кадрами виконується методом KLT (Kanade–Lucas–Tomasi), який дозволяє оцінити гомографію та компенсувати рух камери. Така стабілізація особливо важлива при зйомці з дронів, де присутні вібрації, різкі повороти або ефект rolling shutter.

Для покращення контрастності застосовується метод CLAHE (Contrast Limited Adaptive Histogram Equalization), який дозволяє вирівнювати локальний контраст зображення навіть у випадках неоднорідного освітлення. Методика заснована на розбитті кадру на блоки з локальною еквалізацією, що запобігає надмірному підсиленню шумів.

Для зменшення цифрового шуму, що може бути викликаний умовами низького освітлення, планується використання фільтрів згладжування. Зокрема, базовим варіантом є гаусівська фільтрація з ядром 3×3 або 5×5. У деяких випадках може

застосовуватись білатеральна фільтрація, яка зберігає краї об’єктів, залишаючи зображення придатним для подальшого виявлення.

Усі операції препроцесингу виконуються в реальному часі під час обробки кожного кадру та реалізовані за допомогою стандартного функціоналу бібліотеки OpenCV.

2.5.2 Вибір конфігурації моделі YOLO

На основі обґрунтування вибору архітектури YOLO, представленого в розділі 1.4, для реалізації системи обрано сучасну модель YOLOv8 як оптимальну за співвідношенням точності та обчислювальної ефективності.

Після аналізу еволюції архітектури YOLO (від YOLOv5 до YOLOv8) виявлено, що новіші версії демонструють значні покращення як у точності виявлення об'єктів, так і в обчислювальній ефективності. YOLOv8, розроблена компанією Ultralytics у 2023 році, представляє найсучаснішу реалізацію архітектури YOLO з низкою удосконалень порівняно з попередніми версіями.

Сімейство моделей YOLOv8 представлене декількома конфігураціями, що відрізняються розміром та структурою: nano (YOLOv8n), small (YOLOv8s), medium (YOLOv8m), large (YOLOv8l) та extra-large (YOLOv8x). Для початкової реалізації системи обрано конфігурацію YOLOv8m, яка забезпечує оптимальний баланс між точністю та швидкодією.

Ключові удосконалення YOLOv8 порівняно з попередніми версіями включають нову архітектуру магістралі (backbone) з використанням C2f-блоків, удосконалений механізм передбачення обмежувальних рамок з використанням якірно-вільного підходу, покращену шийку мережі для ефективної передачі інформації між масштабами, оновлену функцію втрат, що краще оптимізує як розташування об'єктів, так і їх класифікацію, а також підтримку багатозадачного навчання, що дозволяє одночасно вирішувати задачі детекції, сегментації та класифікації.

Порівняльний аналіз різних версій YOLO показав, що YOLOv8m демонструє суттєві покращення порівняно з YOLOv5m при аналогічних обчислювальних витратах, що особливо важливо для точного виявлення військової техніки на аерофотознімках.

Навчання моделі виконується з нуля на спеціалізованому наборі аерофотознімків, що містить зображення цільових об'єктів, характерних для заданої предметної області. Такий підхід дозволяє повністю орієнтувати модель на специфіку задачі без прив’язки до сторонніх датасетів, забезпечуючи високу точність у контексті реального застосування.

2.5.3 Алгоритми відстеження об'єктів

Для забезпечення часової узгодженості виявлених об'єктів між кадрами доцільно застосувати алгоритм SORT (Simple Online and Realtime Tracking). Його основною перевагою є простота реалізації та високая швидкість обробки, що критично важливо для роботи в режимі реального часу при аналізі відеопотоків з дронів.

SORT реалізує ефективний механізм асоціації детекцій з існуючими треками за допомогою угорського алгоритму (Hungarian algorithm), який забезпечує оптимальне зіставлення виявлених об'єктів на поточному кадрі з активними траєкторіями. Цей підхід дозволяє мінімізувати кількість помилкових асоціацій та підтримувати стабільність ідентифікаторів об'єктів протягом послідовності кадрів.

Для прогнозування руху об'єктів SORT використовує фільтр Калмана з моделлю постійної швидкості, що дає змогу згладжувати траєкторії та компенсувати короткочасні втрати детекції. Зважаючи на динаміку польоту дронів та характер руху наземних об'єктів, така модель вважається адекватною для поставлених завдань. Алгоритм ефективно обробляє сценарії з частковим перекриттям об'єктів та тимчасовими втратами в полі зору.

Обрані алгоритми формують єдиний технологічний стек, в якому кожен компонент виконує специфічну функцію, а їхня інтеграція забезпечує наскрізну обробку відеопотоку — від моменту захоплення кадру до генерації результатів виявлення й класифікації.

Інтеграція алгоритмів в єдиний конвеєр обробки даних дозволяє забезпечити ефективну взаємодію компонентів та оптимальне використання доступних обчислювальних ресурсів. Такий підхід створює надійну основу для реалізації системи, що відповідає поставленим вимогам та має потенціал для подальшого розвитку.

Використання YOLOv8 як основного механізму класифікації та SORT як трекера дозволяє створити збалансовану систему, здатну до роботи в режимі реального часу з високою точністю та мінімальними обчислювальними затратами.

2.5.4 Модуль виявлення та класифікації об'єктів

Модуль виявлення та класифікації об’єктів запроектовано як центральний елемент системи, відповідальний за виявлення об'єктів на кадрах відеопотоку та визначення їх належності до певної категорії. В його основі передбачається використання архітектури YOLO, адаптованої до умов аерофотозйомки.

Згідно з проєктом, модуль має локалізувати об’єкти у вигляді обмежувальних рамок, оцінювати рівень достовірності виявлення та фільтрувати малоймовірні спрацювання. Для класифікації об’єктів планується використання багатокласових моделей з обчисленням імовірностей належності кожного об’єкта до конкретного класу.

Серед ключових компонентів модуля передбачається інтерфейс моделей, який забезпечить уніфікований спосіб взаємодії з різними версіями YOLO. Компонент завантаження моделей дозволятиме динамічну зміну конфігурацій у процесі роботи. В обчислювальному ядрі модуля буде реалізовано механізм інференсу,

орієнтований на ефективне виконання нейромережевих обчислень із використанням GPU або CPU, залежно від доступних ресурсів.

Очікується наявність підсистеми постобробки, що усуватиме дублікати виявлень, а також системи управління класами, яка дасть змогу гнучко налаштовувати перелік об’єктів для розпізнавання. У якості базової моделі передбачається використання YOLOv8, з можливістю подальшого переходу до новіших версій або альтернативних архітектур

За необхідності передбачається можливість запуску моделі у форматі ONNX за допомогою ONNX Runtime, хоча основна реалізація базується на Ultralytics/PyTorch. Цей модуль має забезпечити точне та швидке виявлення об’єктів у складних умовах, і саме його ефективність визначатиме загальну якість роботи системи.

2.5.5 Модуль відстеження об'єктів

Модуль відстеження об’єктів проєктується як засіб для побудови траєкторій руху між послідовними кадрами, що має підвищити стабільність виявлення та забезпечити аналіз динаміки сцени.

Заплановано реалізувати механізм ініціалізації треків для нових об’єктів, присвоєння унікальних ідентифікаторів, а також підтримку життєвого циклу треку від моменту появи до завершення. Поточні виявлення повинні зіставлятись із активними треками за допомогою метрик подібності, з урахуванням можливих конфліктів та випадків тимчасового зникнення об’єктів з поля зору.

Система прогнозування руху використовуватиме моделі на кшталт фільтра Калмана для передбачення позицій об’єктів у майбутніх кадрах. Це дозволить компенсувати випадки, коли об’єкт тимчасово не розпізнається. Додатково має бути впроваджено механізми згладжування траєкторій та фільтрації шуму, що допоможе мінімізувати кількість хибнопозитивних спрацювань.

Проєктна структура модуля включатиме менеджер треків, асоціатор, предиктор і фільтрувальний підкомпонент. Такий поділ дає змогу незалежно оптимізувати кожну частину модуля.

2.5.6 Модуль візуалізації та інтерфейсу користувача

Модуль візуалізації та інтерфейсу користувача заплановано як засіб інтерактивної взаємодії з оператором. Його основним призначенням є наочне відображення результатів роботи системи, керування параметрами обробки та збереження результатів.

Проєктом передбачено реалізацію візуального накладення результатів виявлення на відеопотік, включно з обмежувальними рамками, текстовими підписами та індикацією траєкторій об’єктів. Класи об’єктів мають відображатися з використанням кольорового кодування. Також розглядається можливість інтеграції відображення статистичних графіків і аналітичних даних.

У структурі модуля виділяється GUI-менеджер для управління налаштуваннями, компонент рендерингу для накладання графіки, а також блок експорту, який дозволить зберігати оброблені відео та метадані у зручному форматі.

Передбачається використання фреймворку Qt як основи для побудови графічного інтерфейсу. З метою зниження навантаження на ресурси планується оптимізувати процес відображення, впровадити кешування елементів інтерфейсу та підтримку гарячих клавіш для швидкого доступу до функцій.

Інтерфейс має бути інтуїтивно зрозумілим, адаптованим до різних розмірів екранів та умов освітлення. Модуль покликаний забезпечити максимально ефективну взаємодію оператора з системою.

2.6 Декомпозиція модулів та опис функціональних блоків

Для забезпечення ефективної реалізації та подальшого супроводу програмної системи було здійснено декомпозицію її основних модулів на функціональні блоки. Такий підхід дозволяє чітко розмежувати відповідальність між компонентами, полегшити тестування окремих частин системи, а також забезпечити можливість їх незалежного розвитку та вдосконалення.

У модулі отримання відеопотоку структура побудована відповідно до логіки проходження даних: від джерела до буферизованого потоку. Передбачається наявність підсистеми, яка взаємодіє з різними джерелами відео — це можуть бути локальні файли, мережеві трансляції за протоколом RTSP або відео з USB-камер. Далі стиснене відео декодується та приводиться до єдиного формату, зокрема з використанням конвертації кольорового простору та синхронізації часових міток. Оброблені кадри передаються у внутрішній буфер, де здійснюється контроль частоти обробки, а також оптимізація навантаження для забезпечення стабільної роботи в режимі реального часу.

Модуль препроцесингу зображень структуровано відповідно до типів операцій, які виконуються перед подачею кадрів у нейронну мережу. Одним із важливих компонентів є стабілізація зображення, що реалізується шляхом виявлення ключових точок, відстеження їх переміщення та обчислення гомографії для компенсації руху. Корекція візуальних параметрів виконується через аналіз гістограми та адаптивну еквалізацію контрасту, що покращує якість зображення навіть за складних умов освітлення. Для боротьби з шумами застосовується фільтрація, яка знижує рівень перешкод, не втрачаючи важливих деталей. Завершальним етапом є геометричні перетворення, що готують кадр до аналізу, включно з масштабуванням, корекцією дисторсії та нормалізацією перспективи.

Архітектура модуля виявлення та класифікації побудована навколо моделі YOLOv8, яка забезпечує виконання обчислень з високою продуктивністю.

Компонент управління моделями дозволяє підключати різні конфігурації YOLO, перемикатись між ними та керувати завантаженням ваг. Перед подачею зображення на вхід мережі здійснюється його нормалізація, формування вхідного тензора та, за необхідності, поділ на підзони для обробки великих кадрів. Сам інференс може виконуватись як на GPU за допомогою CUDA, так і на CPU, з можливістю балансування навантаження між обчислювальними пристроями. Результати обробки проходять через механізм постобробки, який включає декодування вихідних тензорів, фільтрацію за допомогою алгоритму Non-Maximum Suppression та калібрування оцінки достовірності виявлень.

Модуль відстеження об'єктів побудований на основі алгоритму SORT і включає механізми для створення та підтримки життєвого циклу треків. Його структура передбачає ініціалізацію нових треків, управління вже існуючими та зберігання інформації про траєкторії об'єктів. Процес асоціації базується на двоетапному порівнянні поточних виявлень з активними треками, з урахуванням достовірності та подібності за метрикою IoU. Прогнозування переміщення реалізовано через фільтр Калмана, який дозволяє передбачати положення об’єкта, згладжувати траєкторії та зменшувати вплив шумів. Особливу увагу приділено ситуаціям, де об’єкти тимчасово зникають із кадру або перекриваються — для цього використовується логіка виявлення перекриттів і відновлення треків після тимчасової втрати.

У модулі візуалізації та інтерфейсу користувача структура орієнтована на забезпечення повної інтерактивності системи. Відображення результатів аналізу виконується шляхом накладання графічних елементів на відео, зокрема рамок, міток з класами, достовірністю та ідентифікаторами об’єктів. Візуалізація траєкторій дозволяє оцінити динаміку руху у сцені. Користувач може взаємодіяти з системою через елементи інтерфейсу, що дають змогу обирати джерело відео, змінювати параметри роботи алгоритмів і фільтрувати об’єкти за класами. Для аналітики реалізовано збір статистики про виявлення, генерацію звітів, а також систему сповіщень. Крім того, передбачено засоби експорту оброблених даних, включно з

відеозаписом, збереженням метаданих у структурованому форматі та створенням окремих кадрів з виявленими об'єктами.

Зображена на рисунку А.1 у додатку А структура дозволяє простежити повну послідовність обробки кадру — від моменту захоплення відеопотоку до візуалізації результатів, демонструючи взаємозв’язки між підсистемами та функціональні особливості кожного модуля.

Запропонована декомпозиція модулів дозволяє створити гнучку й масштабовану архітектуру. Кожен функціональний блок має чітко визначену зону відповідальності та взаємодіє з іншими через відкриті інтерфейси, що спрощує тестування, супровід і подальший розвиток системи. Такий підхід дає змогу замінювати окремі алгоритми, розширювати можливості або адаптувати систему до нових сценаріїв без суттєвого перегляду базової структури.

2.7 Висновки до розділу

У даному розділі було здійснено комплексне проєктування програмного забезпечення системи автоматизованого виявлення та класифікації об'єктів на аерофотознімках. На основі проведеного аналізу та порівняння архітектурних підходів обрано монолітну архітектуру з елементами багатопотоковості, яка забезпечує оптимальний баланс між продуктивністю обробки в реальному часі (≥10 FPS), автономністю роботи в польових умовах та ефективним використанням обчислювальних ресурсів середньої потужності.

Визначено технологічний стек системи, ядром якого стали: Python як основна мова програмування завдяки багатій екосистемі бібліотек машинного навчання; OpenCV для ефективної обробки відеопотоків; YOLOv8m як оптимальна за співвідношенням швидкість/точність модель виявлення об'єктів; Qt для створення кросплатформного графічного інтерфейсу. Такий вибір технологій забезпечує швидке прототипування MVP з можливістю подальшого масштабування та оптимізації.

Розроблено п'ятикомпонентну архітектуру системи з чітко визначеними функціями кожного модуля: отримання відеопотоку, препроцесинг зображень, виявлення та класифікація об'єктів, відстеження об'єктів, візуалізація та інтерфейс користувача. Взаємодія між компонентами реалізується через буферизовані черги, що забезпечує асинхронну обробку даних та ефективне використання багатоядерних процесорів.

Для кожного етапу обробки обрано оптимальні алгоритми: FAST+KLT для стабілізації зображення, CLAHE для покращення контрасту, SORT з фільтром Калмана для стійкого відстеження об'єктів між кадрами. Така комбінація алгоритмів дозволяє досягти високої якості обробки навіть в складних умовах зйомки з дронів.

Здійснено детальну декомпозицію модулів на функціональні блоки, що забезпечує модульність коду, спрощує тестування та подальший розвиток системи. Кожен блок має чітко визначену зону відповідальності та взаємодіє з іншими через уніфіковані інтерфейси.

Таким чином, запропоноване проєктне рішення створює надійну основу для реалізації системи, яка відповідає всім визначеним функціональним та нефункціональним вимогам, забезпечує роботу в режимі реального часу та має потенціал для подальшого розвитку й адаптації до нових завдань.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Підготовка та розмітка датасету

Підготовка датасету є ключовим етапом, що визначає якість та надійність навчання моделі об'єктної детекції. Для забезпечення репрезентативності тренувальних даних було розроблено комплексну методологію збору, обробки та анотування аерофотознімків.

Джерелами вихідних даних слугували відкриті канали Telegram з аерофотознімками та відеоматеріали з безпілотних літальних апаратів, що охоплювали різні умови зйомки – денне освітлення, вечірні години та похмуру погоду. Для детекції було визначено чотири категорії об'єктів: танки, БТР/БМП, легкові автомобілі та людей.

Для створення датасету було здійснено ручний відбір фрагментів з різноманітними ракурсами та умовами освітлення. Критерії відбору включали наявність чітких зображень об'єктів, відсутність розмиття та достатню контрастність. Після відбору відеоконтенту здійснювався покадровий перегляд та створення скріншотів ключових моментів. Загалом було створено 176 високоякісних знімків, які були стандартизовані до роздільної здатності 1280×720 пікселів.

Анотування кадрів проведено в середовищі Label Studio версії 1.9.0, яке було налаштовано на формат YOLO. Для кожної фотографії задавалися обмежувальні прямокутники та відповідні мітки класів. Перед початком роботи були розроблені детальні інструкції для анотування, що передбачали щільне охоплення видимих контурів об'єкта з мінімізацією вільного простору.

Контроль якості розмітки здійснювався у два етапи. На першому етапі кожен набір анотованих зображень перевірявся на взаємну узгодженість, з використанням коефіцієнта Krippendorff's α (значення 0.86), IoU та Cohen's kappa. Другий етап передбачав автоматизовану валідацію за допомогою спеціального Python-скрипта.

Датасет було розподілено на тренувальну (80%, 141 зображення) і валідаційну (20%, 35 зображень) підмножини із застосуванням стратифікованого підходу. Середня кількість об'єктів на зображення становила 3.2.

Для збільшення обсягу та різноманітності датасету застосовано комплекс технік аугментації даних:

− метод Mosaic: об'єднання чотирьох зображень в одне композитне (40% випадків);

− техніка Mixup: змішування двох зображень з випадковим коефіцієнтом (30% випадків);

− метод Copy-Paste: вирізання об'єктів та їх вставлення у базові зображення (30% випадків).

Алгоритм аугментації автоматично визначав недопредставлені класи та надавав їм вищий пріоритет, що забезпечило більш збалансований розподіл. Загальний обсяг датасету збільшився з початкових 176 анотованих зображень до 3517, що забезпечило достатню варіативність для ефективного навчання нейронної мережі та значно покращило збалансованість представлення всіх чотирьох класів об'єктів.

3.2 Навчання та налаштування моделей

Навчання моделей об'єктної детекції проводилося у середовищі Google Colab із доступом до GPU T4 (16 GB VRAM). Використання хмарної платформи забезпечило доступ до потужних обчислювальних ресурсів без необхідності локальної інфраструктури, що дозволило прискорити процес навчання в десятки разів порівняно з CPU-обчисленнями.

При навчанні використовувався фреймворк Ultralytics YOLOv8 та попередньо натреновані ваги моделі YOLOv8m (Medium). Вибір «Medium» версії моделі обумовлено оптимальним балансом між точністю детекції та швидкістю інференсу. Модель YOLOv8m містить близько 25.9 мільйонів параметрів, що забезпечує

достатню виразність для складних задач детекції при збереженні можливості роботи в реальному часі.

Розроблено два окремі сценарії тренування:

1) модель A навчалася без аугментованих даних на оригінальному датасеті з 176 анотованих зображень;
2) модель B використовувала розширений датасет з 3517 зображень після застосування технік аугментації.

Обидві моделі навчалися за однаковим сценарієм, що гарантує повну відтворюваність. Процес навчання запускався командою:

```
!yolo train \
  model=yolov8m.pt \
  data=/content/data.yaml \
  epochs=60 \
  imgsz=640 \
  batch=16 \
  device=0 \
  patience=10 \
  augment=True
```

Обрані гіперпараметри базувалися на рекомендаціях розробників YOLOv8 та емпіричних дослідженнях. Кількість епох встановлено на рівні 60, що забезпечує достатній час для конвергенції моделі. Розмір вхідних зображень 640×640 пікселів є стандартним для архітектури YOLO та забезпечує оптимальний баланс між якістю детекції та швидкістю обробки. Розмір batch 16 було обрано з урахуванням обмежень пам'яті GPU T4.

Параметр patience=10 активує механізм раннього зупинення, який припиняє навчання при відсутності покращень протягом 10 епох. Це запобігає перенавчанню моделі та економить обчислювальні ресурси. Увімкнення вбудованої аугментації augment=True додатково розширює варіативність тренувальних даних через застосування випадкових трансформацій під час навчання.

Описаний підхід дозволив ефективно використати обчислювальні ресурси та забезпечити методично коректне порівняння впливу аугментації даних. Єдиний протокол тренування з ідентичними базовими параметрами та автоматизоване логування ключових метрик забезпечили об'єктивність подальшого аналізу результатів.

3.3 Програмна реалізація модулів

Програмна реалізація системи складається з набору взаємопов'язаних модулів, кожен з яких відповідає за окрему функціональність. У директорії core зосереджено основні компоненти системи, які забезпечують різні аспекти роботи програми.

Модуль detection.py реалізує клас YOLOv8Detector, який забезпечує інтеграцію з нейромережею YOLOv8 для виявлення об'єктів на зображеннях. Цей клас надає методи для завантаження моделі, виконання інференсу та форматування результатів виявлення. Метод detect() приймає кадр як масив numpy та повертає список виявлених об'єктів із їхніми обмежувальними рамками, класами та рівнями впевненості.

```
def detect(self, frame: np.ndarray) -> List[Dict[str, Any]]:
    if self.model is None:
        return []

    try:
        # Run inference
        results = self.model(frame, verbose=False)[0]

        # Process results
        detections = []
        for r in results.boxes.data.tolist():
            x1, y1, x2, y2, conf, cls = r
            detections.append({
```

```
                    'bbox': (x1, y1, x2, y2),
                    'confidence': conf,
                    'class_id': int(cls),
                    'class_name': self.classes[int(cls)]
                })

            return detections
        except Exception as e:
            print(f"Error during detection: {str(e)}")
            return []
```

Важливою особливістю є можливість динамічної зміни моделі під час роботи програми через метод switch_model(), що дозволяє користувачу експериментувати з різними моделями для досягнення оптимальних результатів. Метод get_available_models() сканує директорію models/ для пошуку доступних моделей.

Модуль tracking.py містить реалізацію алгоритму відстеження об'єктів між кадрами відео. Клас Tracker використовує фільтр Калмана для прогнозування руху об'єктів та асоціації виявлень між кадрами. Кожен відстежуваний об'єкт представлений як екземпляр класу Track, який зберігає ідентифікатор, клас об'єкта, стан фільтра Калмана та історію позицій для візуалізації траєкторії. Основний метод update() оновлює стан всіх відстежуваних об'єктів на основі нових виявлень.

```
    def update(self, detections: List[Dict[str, Any]]) -> List[Dict[str, Any]]:
        self.frame_count += 1

        # Get predicted locations from existing tracks
        trks = []
        for t in self.tracks:
            t.kalman.predict()
            trks.append(self._convert_x_to_bbox(t.kalman.x))

        # Associate detections to tracks
        matched,           unmatched_dets,           unmatched_trks           =
self._associate_detections_to_tracks(
            detections, trks, self.iou_threshold)

        # Update matched tracks
```

```
        for t, trk in enumerate(self.tracks):
            if t not in unmatched_trks:
                d = matched[np.where(matched[:, 1] == t)[0], 0]

trk.kalman.update(self._convert_bbox_to_z(detections[d[0]]['bbox']))
                trk.hits += 1
                trk.time_since_update = 0
                trk.trajectory.append((trk.kalman.x[0], trk.kalman.x[1]))
            else:
                trk.time_since_update += 1
```

Цей підхід використовує алгоритм угорської оптимізації для встановлення відповідності між трекамі та виявленнями, дозволяючи зберігати ідентифікацію об'єктів навіть при тимчасовому зникненні з поля зору.

Візуалізація результатів забезпечується модулем visualization.py через клас Visualizer. Цей клас містить методи для відображення обмежувальних рамок виявлених об'єктів, ідентифікаторів треків та траєкторій руху. Метод draw_detections() малює рамки та мітки для виявлених об'єктів.

```
    def draw_detections(self, frame: np.ndarray,
                        detections: List[Dict[str, Any]],
                        enabled_classes: Optional[List[str]] = None) -> np.ndarray:
        if enabled_classes is None:
            enabled_classes = list(self.colors.keys())

        for det in detections:
            cls_name = det['class_name']
            if cls_name not in enabled_classes:
                continue

            # Get bounding box and confidence
            x1, y1, x2, y2 = map(int, det['bbox'])
            conf = det['confidence']

            # Get color for this class
            color = self.colors.get(cls_name, self.default_color)

            # Draw bounding box
```

```
            cv2.rectangle(frame, (x1, y1), (x2, y2), color, 2)

            # Prepare label
            label = cls_name
            if self.show_confidence:
                label += f": {conf:.2f}"
```

Метод draw_tracks() додає ідентифікатори та траєкторії руху об'єктів, а draw_stats() відображає статистичну інформацію на кадрі, наприклад, кількості виявлень різних класів. Клас підтримує гнучке налаштування кольорів для різних класів об'єктів та інших параметрів відображення.

Модуль data_pipeline.py відповідає за обробку відеопотоку через клас VideoSource. Цей клас абстрагує роботу з різними джерелами відео, такими як камера або відеофайл, надаючи єдиний інтерфейс для їх використання. Методи open() та close() керують підключенням до джерела відео, read() отримує наступний кадр, а seek() дозволяє переміщуватися по кадрам у відеофайлі. Клас також автоматично розраховує FPS та відстежує поточний кадр, що важливо для відображення прогресу відтворення.

Попередня обробка зображень для детекції виконується в модулі preprocessing.py класом Preprocessor. Метод preprocess() змінює розмір вхідного кадру та додає доповнення до стандартного розміру, який очікує нейромережа.

```
def preprocess(self, frame: np.ndarray) -> Tuple[np.ndarray, float]:
    # Get original dimensions
    height, width = frame.shape[:2]

    # Calculate scale factor
    scale = min(self.target_size[0] / width, self.target_size[1] / height)

    # Calculate new dimensions
    new_width = int(width * scale)
    new_height = int(height * scale)

    # Resize image
    resized = cv2.resize(frame, (new_width, new_height))
```

```
        # Create padded image
        padded = np.zeros((self.target_size[1], self.target_size[0], 3),
dtype=np.uint8)
        padded[:new_height, :new_width] = resized

        return padded, scale
```

Цей метод також повертає коефіцієнт масштабування для подальшого перерахунку координат. Метод postprocess() конвертує результати виявлення назад до оригінального розміру зображення, що важливо для правильного відображення рамок на оригінальному кадрі.

Моніторинг системних ресурсів реалізовано в модулі system_monitor.py через клас SystemMonitor. Він надає методи для отримання відсотка використання процесора, оперативної пам'яті, GPU та відеопам'яті GPU. Метод get_all_metrics() повертає словник з усіма метриками для відображення в інтерфейсі користувача.

```
def get_all_metrics(self) -> Dict[str, float]:
    return {
        'cpu': self.get_cpu_usage(),
        'ram': self.get_ram_usage(),
        'gpu': self.get_gpu_usage(),
        'gpu_memory': self.get_gpu_memory_usage()
    }
```

Клас автоматично визначає наявність GPU через бібліотеку GPUtil і адаптує свою поведінку відповідно.

Керування налаштуваннями програми забезпечується модулем config.py через клас Config. Цей клас завантажує конфігурацію з файлу YAML або використовує значення за замовчуванням.

```
def _load_default_config(self) -> Dict[str, Any]:
    return {
        'model': {
            'path': 'models/my_model.pt',
            'confidence_threshold': 0.25,
```

```
            'iou_threshold': 0.45
        },
        'tracking': {
            'max_age': 30,
            'min_hits': 3,
            'iou_threshold': 0.3
        },
        'visualization': {
            'show_confidence': True,
            'show_trajectory': True,
            'trajectory_length': 30,
            'colors': {
                'Танк': [0, 0, 255],      # Red
                'БТР/БМП': [255, 0, 0],   # Blue
                'Авто': [0, 255, 255],    # Yellow
                'Людина': [0, 255, 0]     # Green
            }
        }
    }
```

Набір методів, таких як get_model_path(), get_confidence_threshold() та інші, надає зручний доступ до різних параметрів конфігурації. Метод save_config() зберігає поточну конфігурацію у файл для подальшого використання.

Головний інтерфейс користувача реалізовано в модулі ui/app.py через класи MainWindow та VideoThread. MainWindow створює графічний інтерфейс на основі PyQt6 з панелями налаштувань, статистики та моніторингу ресурсів. VideoThread працює в окремому потоці, щоб запобігти блокуванню інтерфейсу під час обробки відео. Метод update_frame() забезпечує оновлення інтерфейсу з результатами виявлення.

```
def update_frame(self, frame, detections):
    if frame is None:
        return

    # Store current frame and update class counts
    self.current_frame = frame.copy()
```

```
            # Reset class counts
            self.class_counts = {}
            for cls_name in self.class_checkboxes.keys():
                self.class_counts[cls_name] = 0

            # Draw detections on frame
            for det in detections:
                # Check if this class is enabled
                cls_name = det['class_name']
                if cls_name in self.class_checkboxes and
self.class_checkboxes[cls_name].isChecked():
                    # Update count
                    self.class_counts[cls_name] = self.class_counts.get(cls_name, 0)
+ 1
```

Взаємодія між модулями організована за принципом слабкого зв'язування, де кожен модуль надає чіткий інтерфейс для використання його функціональності. Наприклад, MainWindow використовує VideoThread для обробки відео, який у свою чергу використовує YOLOv8Detector для виявлення об'єктів. Результати виявлення можуть бути передані в Tracker для відстеження об'єктів між кадрами та у Visualizer для відображення. Таке розділення відповідальності робить код більш модульним, полегшує тестування окремих компонентів та дозволяє легко замінювати або оновлювати частини системи без впливу на решту коду.

Нижче наведена діаграма класів

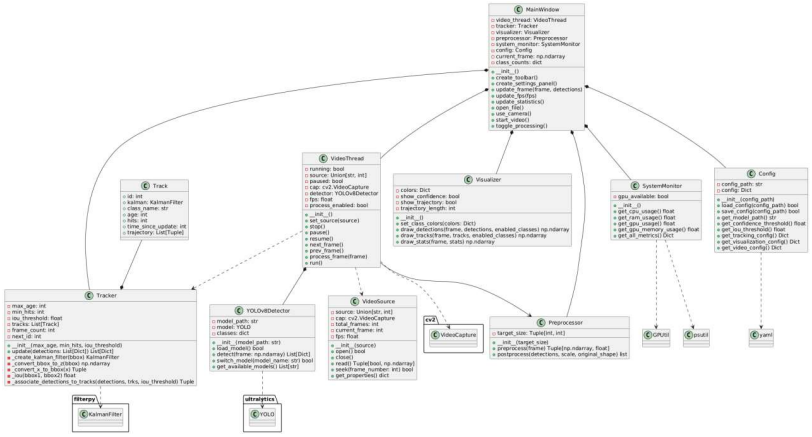


Рисунок 3.1 – Діаграма класів

3.4 Реалізація інтерфейсу користувача

Розробка інтерфейсу користувача є важливим етапом у створенні повноцінної системи виявлення об'єктів на аерофотознімках. Від якості та зручності інтерфейсу залежить ефективність взаємодії користувача з програмою та, як наслідок, успішність виконання поставлених перед системою завдань. Інтерфейс розроблено з використанням бібліотеки PyQt6, яка надає потужні інструменти для створення сучасних графічних додатків.

Головне вікно програми містить центральну область відображення відео, де користувач бачить результати виявлення об'єктів у режимі реального часу. Тут відображаються кадри з камери або відеофайлу з накладеними кольоровими рамками навколо виявлених об'єктів та підписами класів. Різні класи об'єктів виділяються різними кольорами: танки — червоним, БТР/БМП — синім, автомобілі — жовтим, а люди — зеленим, що полегшує візуальне сприйняття результатів виявлення.

Рисунок 3.2 – Головне вікно програми з відображенням виявлених об'єктів на кадрі

У верхній частині вікна розташована панель інструментів з кнопками керування. Вона надає доступ до основних функцій, таких як відкриття відеофайлу, використання камери, керування відтворенням (відтворення/пауза, перехід до попереднього/наступного кадру), увімкнення/вимкнення обробки та збереження знімків екрана. Ця панель забезпечує швидкий доступ до найчастіше використовуваних функцій системи.

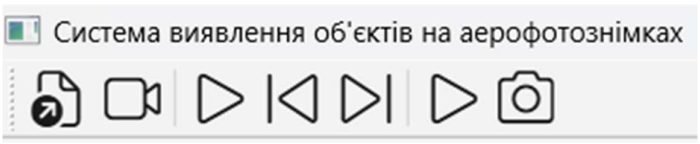


Рисунок 3.3 – Панель інструментів з кнопками керування

З правого боку розташована панель налаштувань, яка дозволяє користувачу змінювати параметри виявлення та відображення. Тут можна налаштувати поріг впевненості та поріг IoU (Intersection over Union) за допомогою слайдерів, вибрати модель для виявлення об'єктів зі списку доступних моделей, а також увімкнути або вимкнути відображення різних класів об'єктів. Додаткові опції дозволяють керувати відображенням значень впевненості та траєкторій руху об'єктів.

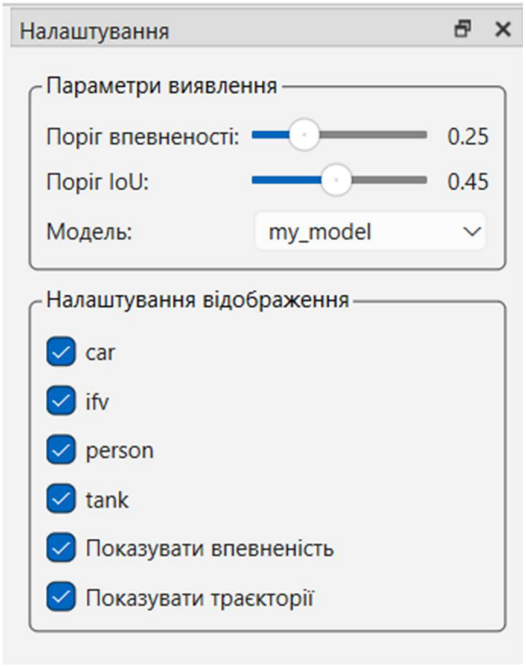


Рисунок 3.4 – Панель налаштувань з параметрами виявлення та відображення

У нижній частині вікна розміщено панель статистики, яка відображає кількість виявлених об'єктів кожного класу в поточному кадрі. Ця інформація представлена у вигляді таблиці з назвами класів та відповідними кількісними показниками. Статистика оновлюється в реальному часі під час обробки відео, що дозволяє користувачу оперативно оцінювати результати виявлення.

Рисунок 3.5 – Панель статистики з кількістю виявлених об'єктів

Для контролю за системними ресурсами розроблено панель моніторингу, яка відображає поточне навантаження на процесор, оперативну пам'ять та, якщо доступно, графічний процесор. Використання ресурсів відображається у вигляді

прогрес-барів з кольоровим кодуванням: зелений колір для низького навантаження, помаранчевий для середнього та червоний для високого. Така візуалізація допомагає користувачу контролювати навантаження на систему під час виконання ресурсомістких операцій виявлення об'єктів.

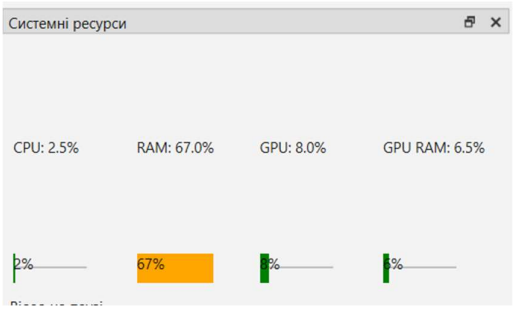


Рисунок 3.6 – Панель моніторингу системних ресурсів

Інтерфейс також інформує користувача про поточну частоту кадрів (FPS), яка відображається в статусному рядку внизу вікна. Це дозволяє оцінити продуктивність системи та вплив різних налаштувань на швидкість обробки. Крім того, статусний рядок використовується для відображення повідомлень про поточний стан програми, такі як шлях до відкритого файлу, використання камери або помилки, що виникають під час роботи.

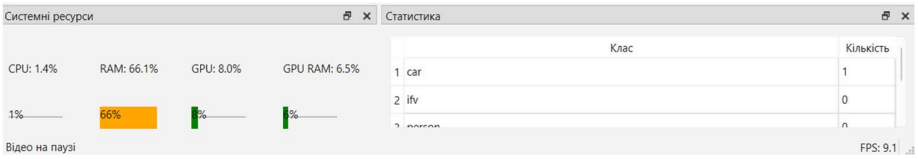


Рисунок 3.7 – Статусний рядок з інформацією про FPS та стан програми

Розроблений інтерфейс реалізує всі функціональні вимоги, включаючи завантаження та обробку відеофайлів різних форматів, підтримку відеопотоку з камери, виявлення та відстеження об'єктів у режимі реального часу, фільтрацію за класами, керування відтворенням та збереження знімків. Особлива увага приділена нефункціональним вимогам, таким як асинхронна обробка відео для забезпечення відгуку

інтерфейсу, моніторинг системних ресурсів, відображення FPS та зручне розташування панелей з можливістю налаштування.

Додатково реалізовано функції, які не були визначені початковими вимогами, але підвищують зручність використання системи. Серед них — можливість динамічної зміни моделі виявлення без перезапуску програми, візуалізація траєкторій руху об'єктів, автоматичне повторення відтворення відеофайлів, покадрове відтворення в режимі паузи та доковані панелі з можливістю зміни їх розташування. Ці додаткові функції роблять систему більш гнучкою та адаптивною до потреб користувача.

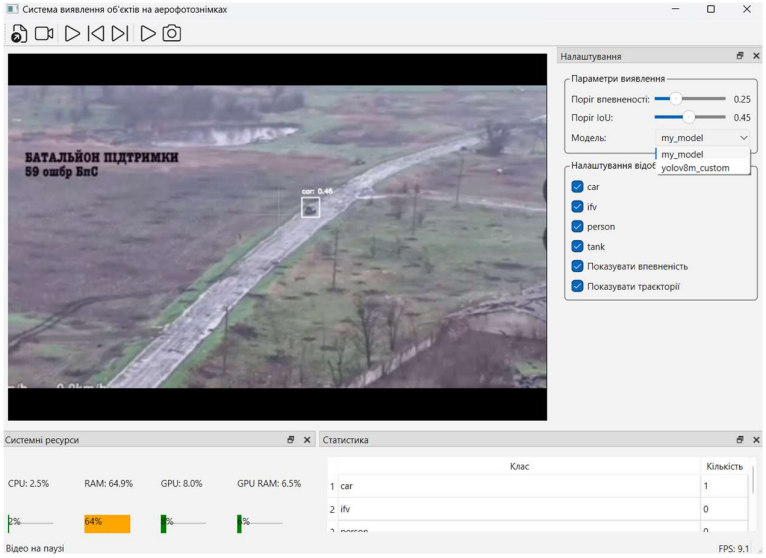


Рисунок 3.8 – Загальний вигляд інтерфейсу з відображенням траєкторій руху об'єктів

Важливою особливістю інтерфейсу є його адаптивність та гнучкість. Панелі налаштувань, статистики та моніторингу можуть бути переміщені користувачем в інші частини вікна або відокремлені в окремі вікна, що дозволяє налаштувати робоче середовище відповідно до індивідуальних уподобань. Також реалізовано адаптивне масштабування відображення відео для різних розмірів вікна, що забезпечує комфортну роботу на різних екранах та при різних розмірах вікна програми.

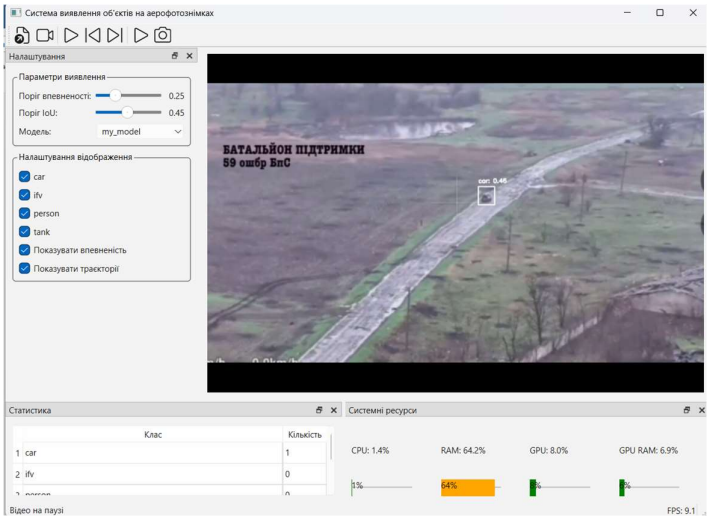


Рисунок 3.9 – Приклад налаштованого інтерфейсу з перенесеними панелями

Таким чином, розроблений інтерфейс користувача забезпечує ефективну взаємодію з системою виявлення об'єктів на аерофотознімках, поєднуючи функціональність, зручність використання та гнучкість налаштування. Він надає всі необхідні інструменти для керування процесом виявлення об'єктів та аналізу результатів, а також забезпечує комфортну роботу з системою для користувачів різного рівня підготовки.

3.5 Вимоги до технічних та програмних засобів

Для забезпечення ефективної роботи системи виявлення об'єктів необхідно дотримуватися певних вимог до апаратного та програмного забезпечення. Ці вимоги обумовлені особливостями функціонування алгоритмів комп'ютерного зору та нейромережевих моделей, які використовуються для виявлення та відстеження об'єктів.

Система потребує продуктивного апаратного забезпечення для забезпечення обробки відеопотоку в режимі реального часу. Основні вимоги до апаратного забезпечення включають:

Багатоядерний процесор — необхідний для забезпечення паралельної обробки даних та управління різними компонентами системи одночасно. Рекомендується використання процесорів з кількістю ядер не менше 4 та тактовою частотою від 2.5 ГГц. Це дозволяє ефективно розподіляти навантаження між різними задачами, такими як попередня обробка зображень, керування інтерфейсом користувача та обробка результатів виявлення.

GPU для швидкого інференсу — графічний процесор є ключовим компонентом для забезпечення високої швидкодії нейромережевих моделей. Рекомендується використання відеокарт NVIDIA з підтримкою CUDA, зокрема серій GTX 1060 або вище, RTX 2060 або вище. Наявність GPU з достатнім об'ємом відеопам'яті (не менше 4 ГБ) забезпечує можливість обробки відеопотоку високої роздільної здатності без суттєвих затримок.

Мінімум 8 GB ОЗУ — достатній об'єм оперативної пам'яті необхідний для завантаження моделей, буферизації відеопотоку та забезпечення стабільної роботи системи. При роботі з високоякісними відеоматеріалами або при використанні декількох моделей одночасно рекомендується збільшити об'єм ОЗУ до 16 ГБ або більше.

SSD накопичувач — твердотільний накопичувач забезпечує швидкий доступ до даних моделей та відеофайлів, що суттєво покращує загальну швидкодію системи. Рекомендований об'єм SSD — не менше 256 ГБ, що дозволяє зберігати моделі виявлення, відеоматеріали та результати обробки. Використання SSD замість HDD зменшує час завантаження моделей та обробки великих відеофайлів.

Для коректної роботи системи виявлення об'єктів на аерофотознімках необхідне специфічне програмне забезпечення. Основні вимоги включають:

Windows 10/11 — операційна система, яка забезпечує сумісність з усіма компонентами системи та необхідними бібліотеками. Рекомендується використання 64-бітних версій операційної системи з найновішими оновленнями для забезпечення стабільності та безпеки.

Python 3.8+ — середовище виконання програмного коду, необхідне для роботи алгоритмів виявлення та відстеження об'єктів. Версія Python 3.8 або новіша забезпечує сумісність з усіма необхідними бібліотеками та фреймворками, такими як PyTorch, OpenCV та PyQt.

CUDA Toolkit — набір інструментів та бібліотек для роботи з GPU NVIDIA, необхідний для забезпечення апаратного прискорення нейромережевих обчислень. Рекомендується використання версії CUDA Toolkit 11.0 або новішої, що забезпечує сумісність з сучасними фреймворками глибокого навчання та моделями YOLOv8.

Бібліотеки з requirements.txt — набір Python-бібліотек, необхідних для функціонування системи, включаючи: PyTorch (1.10.0 або новіше) — фреймворк для глибокого навчання, який використовується для інференсу моделей YOLOv8; OpenCV (4.5.0 або новіше) — бібліотека комп'ютерного зору для обробки зображень та відео; PyQt5 (5.15.0 або новіше) — бібліотека для створення графічного інтерфейсу користувача; NumPy (1.20.0 або новіше) — бібліотека для роботи з багатовимірними масивами та математичними операціями; Pandas (1.3.0 або новіше) — бібліотека для аналізу та обробки даних; Ultralytics (8.0.0 або новіше) — пакет для роботи з моделями YOLOv8.

Вищезазначені вимоги забезпечують оптимальне функціонування системи виявлення об'єктів на аерофотознімках, дозволяючи досягти балансу між швидкодією та точністю роботи алгоритмів виявлення та відстеження.

3.6 Вибір методів тестування

Тестування програмного забезпечення — це процес оцінки та перевірки системи або її компонентів з метою виявлення відмінностей між фактичною та очікуваною поведінкою. Це систематична методика, яка дозволяє визначити, чи відповідає програмне забезпечення встановленим вимогам та чи працює воно належним чином у різних умовах експлуатації .

У контексті розробки програмного забезпечення тестування є критично важливим з декількох причин:

- виявлення помилок та дефектів на ранніх етапах розробки, що зменшує витрати на їх виправлення;
- забезпечення надійності та стабільності системи;
- підтвердження відповідності вимогам замовника та кінцевих користувачів;
- підвищення якості програмного продукту;
- зниження ризиків виникнення критичних збоїв у процесі експлуатації.

Основними видами тестування програмного забезпечення є:

1) функціональне тестування — перевірка відповідності функціональних можливостей системи встановленим вимогам;
2) нефункціональне тестування — оцінка характеристик системи, таких як продуктивність, надійність, зручність використання;
3) структурне тестування — перевірка внутрішньої структури програми (також відоме як тестування "білої скриньки");
4) модульне (unit) тестування — перевірка окремих компонентів програми незалежно від інших;
5) інтеграційне тестування — перевірка взаємодії між компонентами системи;
6) системне тестування — перевірка системи в цілому на відповідність всім вимогам;
7) приймальне тестування — перевірка готовності системи до передачі замовнику;
8) регресійне тестування — перевірка, що зміни в коді не порушили існуючий функціонал;
9) навантажувальне тестування — оцінка поведінки системи при високому навантаженні [26].

Для розробленої системи було застосовано комплексний підхід до тестування, який поєднує кілька методів, орієнтованих на різні аспекти якості програмного забезпечення. Зокрема, проводилося модульне (unit) тестування, що дозволяло перевірити окремі функціональні блоки, зокрема модулі виявлення, відстеження та попередньої обробки зображень. Це забезпечило виявлення помилок на ранніх етапах і дало змогу локалізувати джерело потенційних проблем у межах конкретного модуля.

Крім того, проводилося метричне тестування точності моделей. У цьому випадку основну увагу було приділено кількісній оцінці ефективності алгоритмів виявлення об'єктів. Для цього використовувалися загальноприйняті метрики, такі як precision (точність), recall (повнота) та F1-міра, які розраховувалися на основі результатів на відповідних тестових наборах даних.

З метою додаткової перевірки загальної працездатності та зручності системи було також проведено експлораторне ручне тестування. З метою додаткової перевірки загальної працездатності та зручності системи було також проведено експлораторне ручне тестування. Воно охоплювало взаємодію користувача з графічним інтерфейсом, перевірку логіки виконання основних сценаріїв використання та відповідність функціональності очікуваній поведінці.

Такий підхід дозволяє забезпечити високу якість програмного продукту, його надійність, точність та ефективність роботи з різними типами аерофотознімків та в різних умовах експлуатації.

3.6.1 Unit-тести компонентів

Unit-тестування є ключовим підходом для забезпечення якості окремих компонентів системи. Для кожного модуля розроблено набір тестів, які перевіряють функціональність та стійкість до різних вхідних даних [27].

Модуль виявлення об'єктів (detection.py) перевіряється на коректність завантаження моделі, виявлення об'єктів на тестових зображеннях та обробку

результатів. Особлива увага приділяється перевірці формату повернутих даних та обробці граничних випадків.

Модуль відстеження (tracking.py) тестується на коректність створення та оновлення треків, асоціації виявлень з треками та обчислення траєкторій руху об'єктів. Реалізовано тести для перевірки стійкості алгоритму при зникненні та появі об'єктів у кадрі.

Модуль попередньої обробки (preprocessing.py) перевіряється на коректність зміни розміру зображення, додавання доповнення та перерахунку координат. Тести забезпечують, що після обробки зображення має очікуваний розмір, а координати рамок коректно трансформуються.

Модуль візуалізації (visualization.py) тестується на коректність відображення виявлених об'єктів, треків та статистики. Перевірки включають наявність очікуваних елементів на вихідному зображенні, як-от обмежувальні рамки та траєкторії.

Модуль конфігурації (config.py) перевіряється на коректність завантаження налаштувань за замовчуванням, з файлу та збереження параметрів. Тести забезпечують, що конфігурація коректно зберігає та відновлює всі параметри системи.

Модуль системного моніторингу (system_monitor.py) тестується на коректність отримання метрик використання ресурсів. Тести перевіряють правильність формату та діапазону повернутих значень.

Для надійності тестів використовуються заглушки (mocks) для компонентів, які залежать від зовнішніх ресурсів. Також реалізовані параметризовані тести для перевірки з різними вхідними даними та налаштуваннями. Тести запускаються автоматично при кожному внесенні змін до коду, що забезпечує раннє виявлення потенційних проблем.

Реалізована система unit-тестування забезпечує комплексну перевірку функціональності всіх ключових компонентів системи відстеження об'єктів. Створені 60 unit-тестів покривають основні модулі системи - від виявлення та відстеження об'єктів до візуалізації результатів та системного моніторингу.

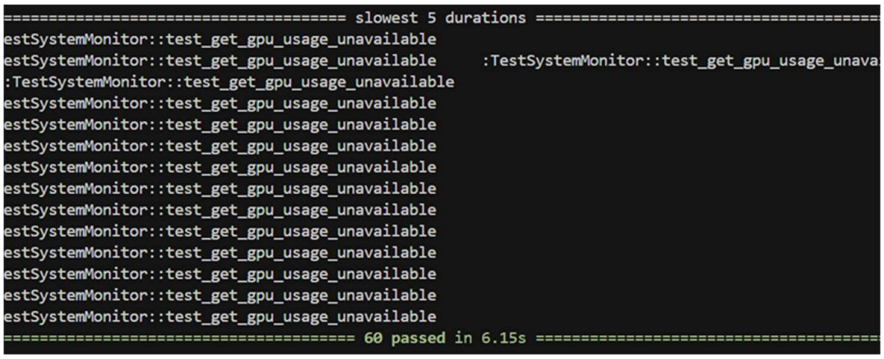


Рисунок 3.10 – Скріншот виконання unit-тестів

Успішне проходження всіх тестів підтверджує стабільність та коректність роботи окремих компонентів системи, що створює надійну основу для подальшого розгортання системи.

3.6.2 Метричне тестування точності моделей

Метричне тестування точності моделей дозволяє кількісно оцінити ефективність алгоритмів виявлення об'єктів на аерофотознімках за допомогою чітко визначених метрик.

Для системи використовуються наступні ключові метрики:

- точність (precision) — відношення правильно виявлених об'єктів до загальної кількості виявлень;
- повнота (recall) — відношення правильно виявлених об'єктів до загальної кількості реальних об'єктів;
- F1-міра — гармонічне середнє між точністю та повнотою, яке забезпечує збалансовану оцінку продуктивності моделі;
- mAP (mean Average Precision) — середня точність для різних порогів впевненості та класів об'єктів, це найбільш універсальна метрика для оцінки моделей;
- IoU (Intersection over Union) — метрика точності позиціонування обмежувальних рамок [28].

Процес тестування включає підготовку валідаційних даних з точною розміткою, запуск моделі, порівняння результатів з еталоном та аналіз ефективності. Для об'єктивності використовуються різні набори даних з різними умовами освітлення, ракурсами та типами місцевості.

У рамках тестування проведено порівняльну оцінку двох моделей — без аугментації (модель А) та з аугментацією (модель В) на валідаційному наборі з 20% стратифікованих даних.

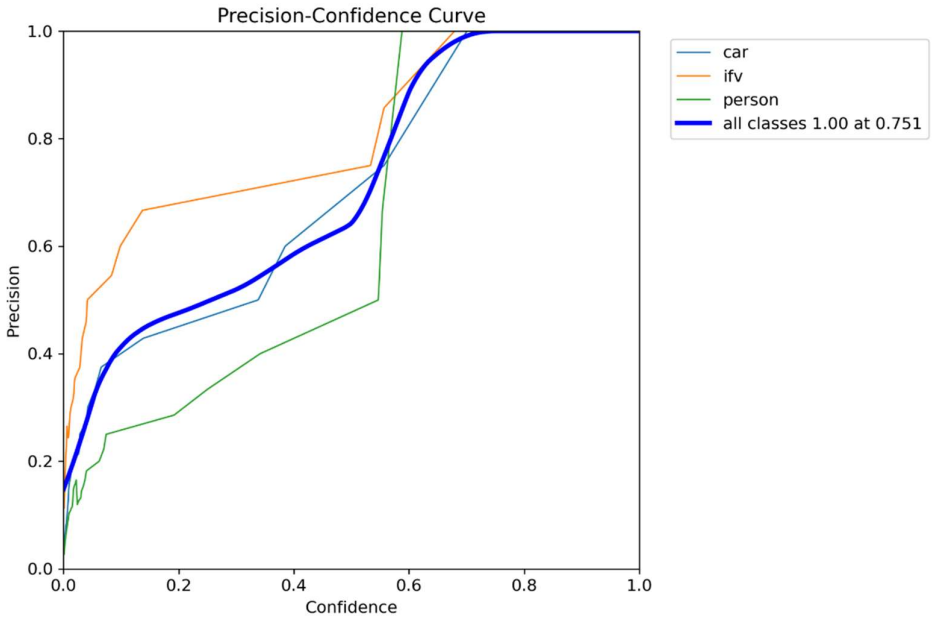


Рисунок 3.11 – Графік Precision–Confidence для моделі А

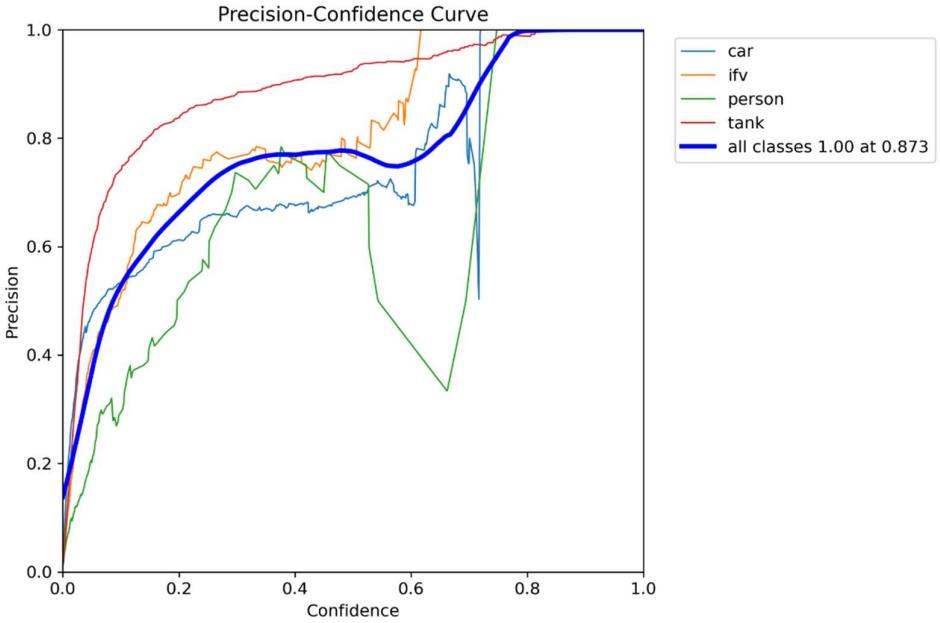


Рисунок 3.12 – Графік Precision–Confidence для моделі B

На рисунках 3.10 та 3.11 показано криві Precision–Confidence для обох моделей. Модель A демонструє стабільну точність для всіх класів з загальним показником mAP 0.751. Найкращу продуктивність показує клас "car" (помаранчева лінія), досягаючи точності понад 0.65 при низьких порогах довіри. Модель B з аугментацією показує покращену загальну продуктивність з mAP 0.873. Хоча індивідуальні класи демонструють більшу варіативність точності залежно від порога довіри, загальна крива (синя лінія) підтримує високу точність у широкому діапазоні порогів довіри, що свідчить про більшу стабільність моделі після аугментації.

У таблиці 3.1 наведено підсумкові метрики загальної якості моделей:

Таблиця 3.1 – Метрики загальної якості моделей

Модель	mAP@0.5	Precision	Recall	F1
A (без аугм.)	0.60	0.85	0.55	0.67
B (з аугм.)	0.55	0.82	0.65	0.72

3.6.3 Експлораторне ручне тестування

Експлораторне ручне тестування системи виявлення об'єктів на аерофотознімках було успішно проведено згідно з розробленою тестовою документацією. Усі аспекти функціональності системи пройшли перевірку, підтвердивши високу якість та стабільність розробленого програмного забезпечення.

Тестування користувацького інтерфейсу показало, що система має інтуїтивно зрозумілий та зручний інтерфейс. Навігація між різними панелями працює бездоганно, усі елементи керування коректно відображаються та швидко реагують на дії користувача.

Функціональність обробки відео продемонструвала відмінні результати при роботі з різними типами відеофайлів. Система успішно обробляє файли форматів mp4, avi, mov та mkv різного розміру та якості. Візуалізація результатів виявлення об'єктів працює коректно, з чітким відображенням усіх класів об'єктів.

Експорт та збереження знімків екрана з виявленими об'єктами функціонує без збоїв.

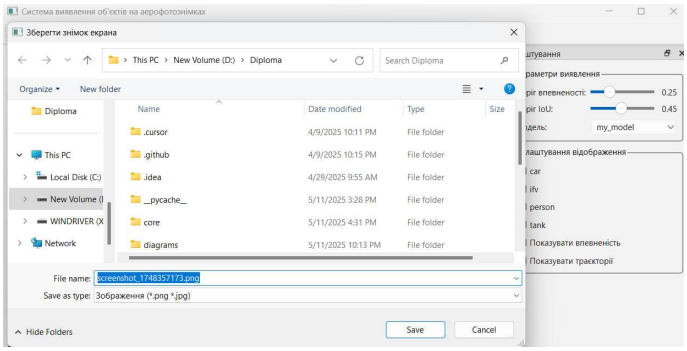


Рисунок 3.13 – Вікно збереження знімку екрана

Тестування на стійкість та надійність підтвердило, що система залишається стабільною навіть у нестандартних ситуаціях. При роботі з пошкодженими файлами система коректно виявляє проблему та видає відповідне повідомлення без

зависань чи збоїв. Раптові перебої у роботі камери успішно обробляються, а при високому навантаженні система зберігає працездатність, хоч і з очікуваним зниженням частоти кадрів.

Оцінка продуктивності показала високу продуктивність системи. Інтерфейс залишається швидким навіть при обробці відеопотоків високої роздільної здатності. Використання системних ресурсів оптимізовано, що дозволяє системі ефективно працювати на обладнанні з рекомендованими характеристиками. [РИС.

Відтворення відео з виявленими об'єктами та їх траєкторіями відбувається плавно без помітних затримок.

Перевірка налаштувань та конфігурації підтвердила гнучкість системи. Зміна параметрів виявлення, налаштування візуалізації та вибір різних моделей працюють без помилок.

Система успішно зберігає та відновлює конфігурації через конфігураційний файл, дозволяючи користувачам швидко налаштовувати її під свої потреби.

Всі заплановані тестові випадки, включаючи перевірку відкриття та відтворення відеофайлів, роботу елементів керування, налаштування параметрів виявлення, вибір класів для відображення та інші функціональні тести, були успішно пройдені. Система продемонструвала стабільну роботу з відеопотоком камери, коректне відображення статистики виявлених об'єктів та точний моніторинг системних ресурсів.

Результати тестування підтверджують готовність системи до використання в реальних умовах для виявлення та відстеження об'єктів на аерофотознімках.

3.7 Висновки до розділу

У даному розділі детально висвітлено процес створення програмного комплексу для аналізу та ідентифікації об'єктів на аерофотознімках. Розділ охоплює всі етапи розробки – від підготовки даних до тестування готового рішення.

Ключовим етапом стала підготовка та розмітка датасету, що включала відбір репрезентативних аерофотознімків з різних джерел, їх анотування з використанням Label Studio та застосування сучасних технік аугментації (Mosaic, Mixup, Copy-Paste). Початковий набір зі 176 анотованих зображень було розширено до 3517 зображень за допомогою аугментації, що дозволило збалансувати представлення класів об'єктів та підвищити генералізаційну здатність моделі.

Процес навчання моделей YOLOv8m проводився за двома різними підходами: модель А навчалася на оригінальному датасеті, а модель В – на розширеному аугментованому наборі даних. Порівняльний аналіз результатів показав, що модель В з аугментацією продемонструвала суттєве покращення метрики AP для недопредставлених класів хоча загальна точність незначно знизилася (mAP50 зменшилася з 0.60 до 0.55). Водночас модель В забезпечила вищий показник Recall (0.65 проти 0.55), що є важливим для практичного застосування розробки.

Програмна реалізація базується на модульній архітектурі, що забезпечує гнучкість та можливість незалежного тестування та вдосконалення окремих компонентів. Основні модулі включають detection.py для виявлення об'єктів, tracking.py для відстеження між кадрами, preprocessing.py для попередньої обробки зображень, visualization.py для візуалізації результатів, system_monitor.py для моніторингу системних ресурсів та config.py для керування налаштуваннями.

Інтерфейс користувача, розроблений на базі PyQt6, забезпечує зручну взаємодію з програмою через інтуїтивно зрозумілі елементи керування. Він включає центральну область відображення відео з результатами виявлення, панель інструментів з кнопками керування, панель налаштувань для зміни параметрів, панель

статистики та панель моніторингу системних ресурсів. Особлива увага приділена функціональності, що дозволяє користувачу налаштовувати параметри виявлення, керувати відтворенням відео та зберігати результати.

Для забезпечення якості розробленого програмного засобу застосовано комплексний підхід до тестування, що включає модульне тестування окремих компонентів, метричне тестування точності моделей та експлораторне ручне тестування функціоналу. Розроблені тестові сценарії охоплюють всі аспекти роботи – від коректності детекції об'єктів до зручності користувацького інтерфейсу.

Сформульовано вимоги до технічних та програмних засобів, необхідних для ефективної роботи, включаючи багатоядерний процесор, GPU з підтримкою CUDA, достатній об'єм ОЗУ та SSD-накопичувач, а також специфічне програмне забезпечення – Windows 10/11, Python 3.8+, CUDA Toolkit та набір необхідних Python-бібліотек.

У результаті виконання дослідження було створено повнофункціональний програмний комплекс, який демонструє високу точність виявлення різних класів об'єктів та забезпечує зручний інтерфейс для роботи з відеоматеріалами. Розроблене рішення може бути використане для аналізу аерофотознімків у різних сценаріях застосування, де необхідна автоматизована ідентифікація та відстеження об'єктів.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було успішно розроблено програмну систему для автоматизованого виявлення та класифікації об'єктів на аерофотознімках у режимі реального часу. Всі поставлені завдання виконано повністю, а отримані результати підтверджують ефективність запропонованого підходу.

Проведений комплексний аналіз предметної області підтвердив критичну важливість автоматизації процесів розпізнавання об'єктів у відеопотоках з дронів. Встановлено, що після двох годин роботи активність префронтальної кори операторів знижується на 35%, що призводить до пропуску однієї цілі з десяти. Ці дані переконливо обґрунтовують необхідність розробки автоматизованих систем для підтримки операторів.

Подальший огляд існуючих програмних рішень виявив суттєві обмеження закритих систем OCHI та Avengers через їх недоступність для широкого використання, а також складність налаштування універсальних рішень типу NVIDIA DeepStream SDK для специфічних завдань. Виявлені недоліки існуючих рішень остаточно підтвердили актуальність розробки власного спеціалізованого програмного продукту.

На основі аналізу сучасних методів класифікації об'єктів було обґрунтовано обрано архітектуру YOLO як оптимальну за співвідношенням швидкості та точності для роботи в реальному часі. Зокрема, YOLOv8 забезпечує обробку 30-60 FPS при збереженні високої точності виявлення, що повністю відповідає технічним вимогам розроблюваної системи.

Базуючись на результатах порівняльного аналізу архітектурних підходів, було обрано монолітну архітектуру з елементами багатопотоковості, яка забезпечує оптимальний баланс між продуктивністю (≥10 FPS), автономністю роботи та ефективним використанням ресурсів. У рамках цього підходу розроблено п'ятикомпонентну систему з чітко визначеними функціями: отримання відеопотоку,

препроцесинг зображень, виявлення та класифікація об'єктів, відстеження об'єктів, візуалізація та інтерфейс користувача.

Для практичної реалізації системи сформовано технологічний стек на основі Python, OpenCV, YOLOv8m та Qt, який забезпечує швидке прототипування MVP з можливістю подальшого масштабування. Відповідно до архітектурних рішень, для кожного етапу обробки обрано оптимальні алгоритми: FAST+KLT для стабілізації зображення, CLAHE для покращення контрасту, SORT з фільтром Калмана для відстеження об'єктів.

На етапі програмної реалізації створено повнофункціональний програмний комплекс з модульною архітектурою, що забезпечує гнучкість та можливість незалежного тестування компонентів. Для навчання моделей підготовлено спеціалізований датасет з 176 анотованих аерофотознімків, який було розширено до 3517 зображень за допомогою сучасних технік аугментації (Mosaic, Mixup, Copy-Paste).

Порівняльне тестування двох моделей продемонструвало важливі результати щодо ефективності застосування методів аугментації. Модель з аугментацією (Модель B) показала суттєве покращення для недопредставлених класів: AP для танків зросла з ≈0.00 до 0.95, хоча загальна mAP50 незначно знизилася з 0.60 до 0.55. Особливо важливим є те, що модель B забезпечила вищий показник Recall (0.65 проти 0.55), що критично важливо для практичного застосування у військових цілях.

Паралельно з розробкою основного функціоналу створено інтуїтивний графічний інтерфейс на базі PyQt6 з центральною областю відображення відео, панелями налаштувань, статистики та моніторингу ресурсів. Розроблений інтерфейс забезпечує зручну взаємодію з усіма функціями системи та адаптується до різних умов використання.

Створена система демонструє високу ефективність виявлення різних класів військової техніки та може бути використана для аналізу аерофотознімків у військових завданнях розвідки та моніторингу. Модульна архітектура системи дозволяє легко адаптувати її до нових завдань та розширювати функціональність відповідно до специфічних потреб користувачів.

Проведене комплексне тестування, що включало модульне тестування компонентів, метричну оцінку точності моделей та експлораторне ручне тестування, підтвердило готовність системи до використання в реальних умовах. Система демонструє стабільну роботу навіть при високому навантаженні та нестандартних ситуаціях, що свідчить про надійність розроблених алгоритмів.

Аналіз результатів тестування та практичного застосування дозволяє визначити перспективні напрямки вдосконалення системи: розширення датасету для покращення виявлення класу "Люди", інтеграція з хмарними сервісами для розподіленої обробки, додавання підтримки тепловізійних камер, реалізація функцій прогнозування траєкторій руху об'єктів та розробка мобільної версії для планшетів.

Таким чином, поставлена мета досягнута повністю. Розроблена система автоматизованого виявлення та класифікації військової техніки на аерофотознімках відповідає всім функціональним та нефункціональним вимогам, забезпечує роботу в режимі реального часу та має значний потенціал для практичного застосування у військовій сфері. Результати роботи підтверджують ефективність запропонованих технічних рішень та доцільність обраного підходу до розробки спеціалізованих систем комп'ютерного зору.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1) 28 July 1858 – the first aerial photograph [Електронний ресурс]. – 27 лип. 2023. – URL: https://afterburner.com.pl/28-july-1858-the-first-aerial-photograph/ (дата звернення: 11.01.2025)

2) EastFruit. Мультиспектральний аналіз дронами в садівництві та овочівництві: можливості для підвищення ефективності [Електронний ресурс]. – 2023. – URL: https://east-fruit.com/uk/plodoovochevyi-rynok/oglyady-rynku/multyspektralnyy-analiz-dronamy-v-sadivnytstvi-ta-plodoovochivnytstvi-mozhlyvosti-dlya-pidvyshchennya-efektyvnosti/ (дата звернення: 15.01.2025)

3) Farmbrite. Innovative Uses for Drones in Agriculture [Електронний ресурс]. – 2024. – URL: https://www.farmbrite.com/post/innovative-uses-for-drones-in-agriculture (дата звернення: 19.01.2025)

4) Los Angeles Times. Drones with thermal imaging cameras locate elderly man lost in Malibu [Електронний ресурс]. – 26 груд. 2024. – URL: https://www.latimes.com/california/story/2024-12-26/drones-with-thermal-imaging-cameras-locate-elderly-man-lost-in-malibu (дата звернення: 23.01.2025)

5) Raymer N., Moore A. J. A Review of Unmanned Aerial Vehicle Technology in Power Line Inspection [Електронний ресурс]. – NASA, 2024. – URL: https://ntrs.nasa.gov/api/citations/20205002834/downloads/A%20Review%20of%20Unmanned%20Aerial%20Vehicle%20Technology%20in%20Power%20Line%20Inspection%20Abstract%20V1.6.pdf – Назва з екрана. (дата звернення: 27.01.2025)

6) Flyability. Elios 3 Technical Specifications [Електронний ресурс]. – 2024. – URL: https://www.flyability.com/elios-3 (дата звернення: 31.01.2025)

7) Drone Industry Insights. Global Drone Market Report 2025-2030 [Електронний ресурс]. – 2025. – URL: https://droneii.com/product/drone-market-report (дата звернення: 04.02.2025)

8) Institute of Cognitive Science Research. Prefrontal cortex activity decline in drone operators during extended missions // Journal of Military Psychology. – 2024. – Vol. 15, № 3. – С. 45–62.

9) Encord. Computer Vision Application in Agriculture – Automation through Smart Farming [Електронний ресурс]. – 2024. – URL: https://encord.com/blog/computer-vision-in-agriculture/ (дата звернення: 08.02.2025)

10) ITEA Journal. Optimizing Real-time Detection in Military Operations [Електронний ресурс]. – 2024. – URL: https://itea.org/journals/volume-45-3/optimizing-performance-of-real-time-detection/ (дата звернення: 12.02.2025)

11) Reuters. Ukraine collects vast war data trove to train AI models [Електронний ресурс]. – 2024. – URL: https://www.reuters.com/technology/ukraine-collects-vast-war-data-trove-train-ai-models-2024-12-20/ (дата звернення: 16.02.2025)

12) «Штучний інтелект щотижня допомагає виявляти 12 тисяч ворожих цілей» [Електронний ресурс] // ArmyInform. – 23 верес. 2024. – URL: https://armyinform.com.ua/2024/09/23/shtuchnyj-intelekt-shhotyzhnya-dopomagaye-vyyavlyaty-12-tysyach-vorozhyh-czilej-mou/ (дата звернення: 20.02.2025)

13) NVIDIA Corporation. DeepStream SDK [Електронний ресурс]. – 2025. – URL: https://developer.nvidia.com/deepstream-sdk (дата звернення: 24.02.2025)

14) LeCun Y.; Bengio Y.; Hinton G. Deep learning [Електронний ресурс] // Nature. – 2015. – Vol. 521. – P. 436–444. doi:10.1038/nature14539. – URL: https://www.nature.com/articles/nature14539 (дата звернення: 28.02.2025)

15) Lowe D. G. Distinctive Image Features from Scale-Invariant Keypoints // International Journal of Computer Vision. – 2004. – Vol. 60, № 2. – С. 91–110.

16) Dalal N., Triggs B. Histograms of Oriented Gradients for Human Detection // IEEE Conference on Computer Vision and Pattern Recognition. – 2005. – С. 886–893.

17) Viola P., Jones M. Rapid Object Detection using a Boosted Cascade of Simple Features // IEEE CVPR. – 2001. – Vol. 1. – С. 511–518.

18) Shobaki W. A.; Milanova M. A Comparative Study of YOLO, SSD, Faster R-CNN, and More for Optimized Eye-Gaze Writing [Електронний ресурс] // Sci. – 2025. – Vol. 7, № 2. – Стаття 47. doi:10.3390/sci7020047. – URL: https://www.mdpi.com/2413-4155/7/2/47 (дата звернення: 04.03.2025)

19) Newman S. Building Microservices: Designing Fine-Grained Systems. – 2-ге вид. – Sebastopol, CA : O'Reilly Media, 2021. – 352 с.

20) Shi W.; Cao J.; Zhang Q. Edge Computing: Vision and Challenges // IEEE Internet of Things Journal. – 2023. – Vol. 3, № 5. – С. 637–646.

21) Van Rossum G.; Drake F. L. Python Programming Language: Design and Implementation // ACM Computing Surveys. – 2023. – Vol. 41, № 4. – С. 1–36.

22) Bradski G.; Kaehler A. Learning OpenCV 4: Computer Vision and Machine Learning with the OpenCV Library. – 5-те вид. – O'Reilly Media, 2022. – 896 с.

23) Zhao Z.; Zheng P.; Xu S. Object Detection Networks on Convolutional Feature Maps // IEEE Transactions on Pattern Analysis and Machine Intelligence. – 2023. – Vol. 45, № 8. – С. 3993–4009.

24) Solawetz J.; Francesco. What is YOLOv8? A Complete Guide [Електронний ресурс] // Roboflow Blog. – Des Moines, IA : Roboflow, 2024. – URL: https://blog.roboflow.com/what-is-yolov8/ (дата звернення: 12.03.2025)

25) Google Research. Colaboratory FAQ [Електронний ресурс]. – 2024. – URL: https://research.google.com/colaboratory/faq.html (дата звернення: 16.03.2025)

26) Myers G. J.; Sandler C.; Badgett T. The Art of Software Testing. – 3-тє вид. – Hoboken, NJ : John Wiley & Sons, 2022. – 256 с.

27) Beck K. Test-Driven Development: By Example. – 2-ге вид. – Boston : Addison-Wesley Professional, 2022. – 240 с

28) Padilla R.; Netto S. L.; da Silva E. A. B. A Survey on Performance Metrics for Object-Detection Algorithms // Proceedings of the International Conference on Systems, Signals and Image Processing. – 2023. – С. 237–242.

ДОДАТОК А

(обов’язковий)

ГРАФІЧНІ МАТЕРІАЛИ

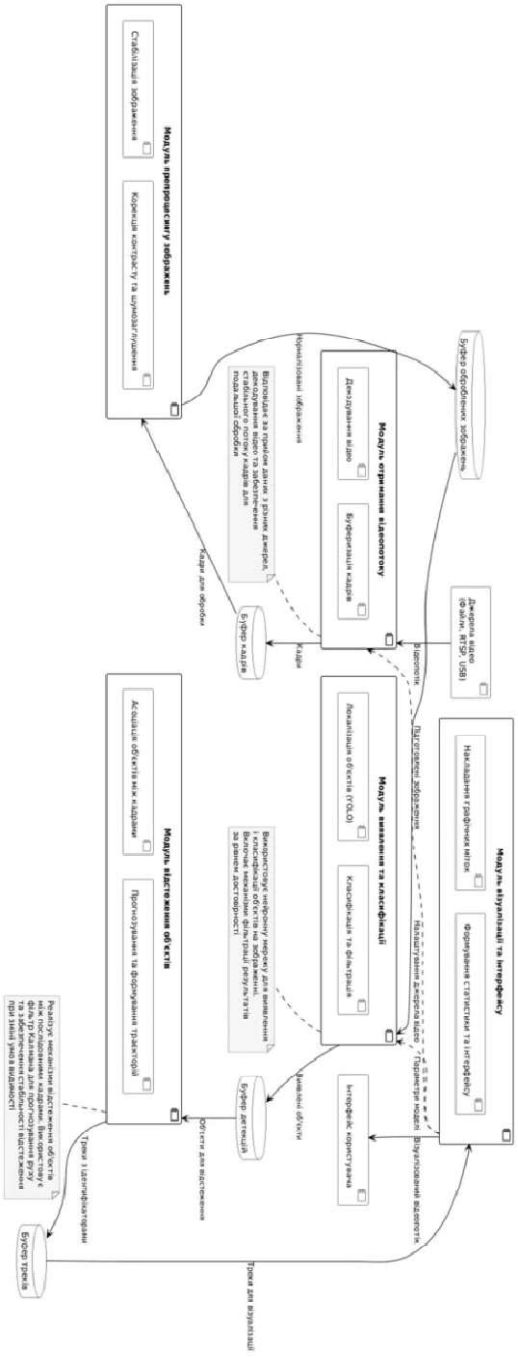


Рисунок А1 – Діаграма класів

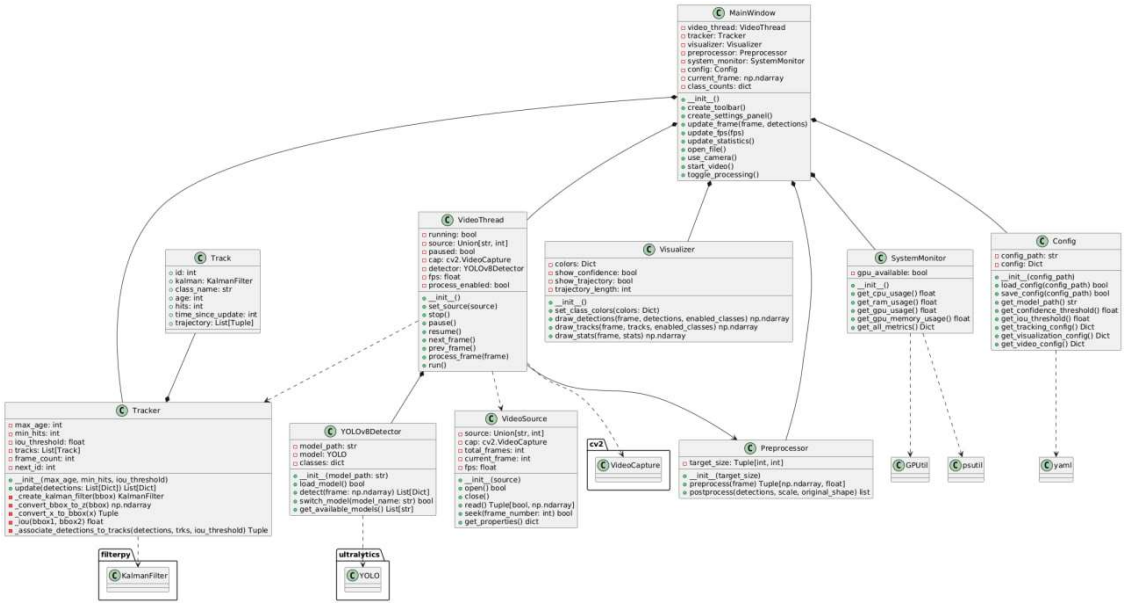


Рисунок А2 – Діаграма класів та інтерфейсів

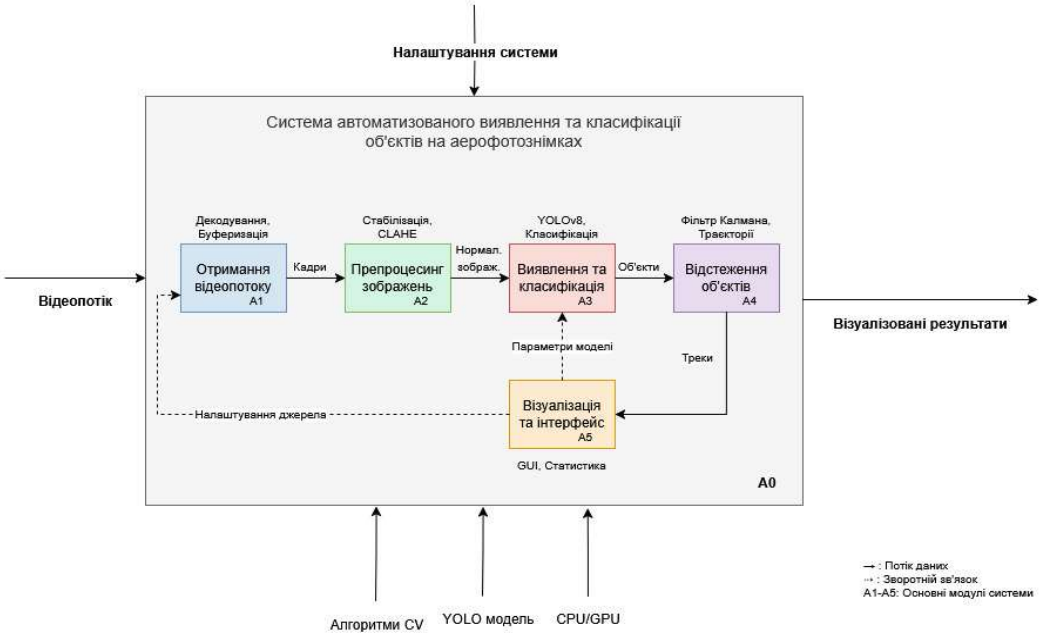


Рисунок А3 – Діаграма IDEF0 декомпозиції системи

ДОДАТОК Б

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Рисунок Б.1 – Слайд 1

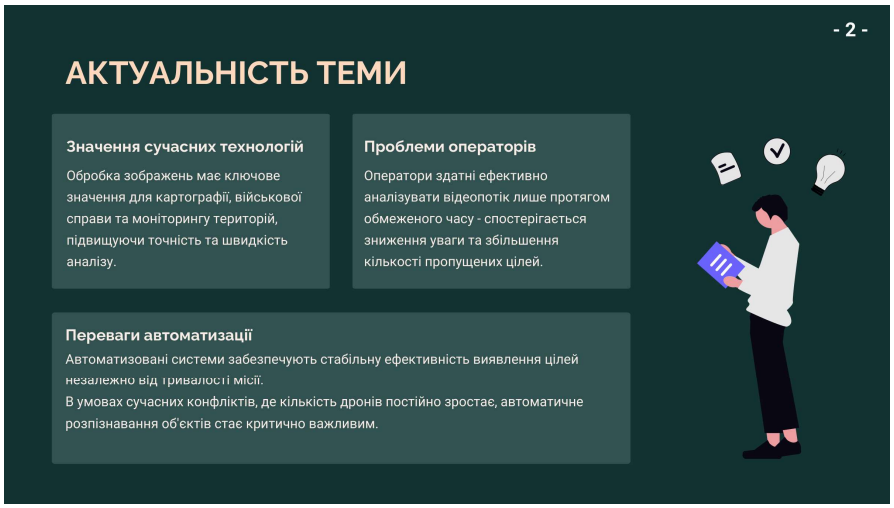


Рисунок Б.2 – Слайд 2

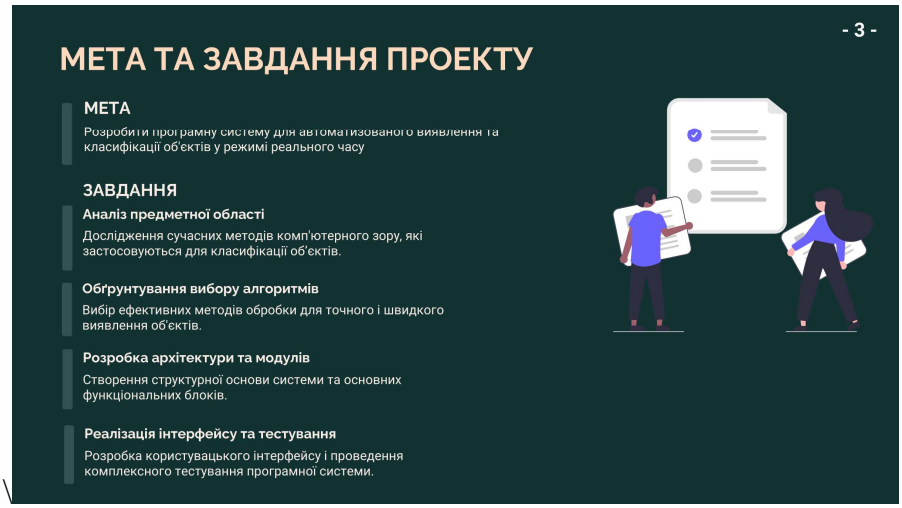


Рисунок Б.3– Слайд 3

Рисунок Б.4– Слайд 4

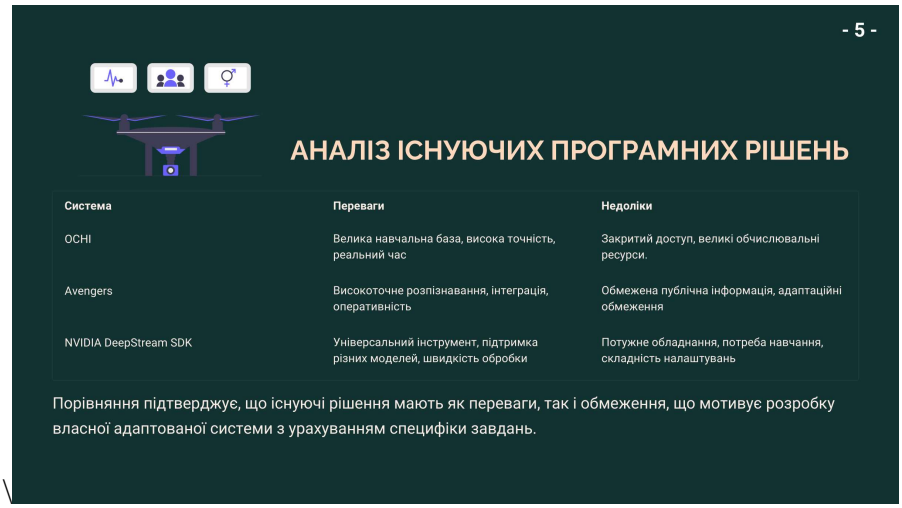


Рисунок Б.5– Слайд 5

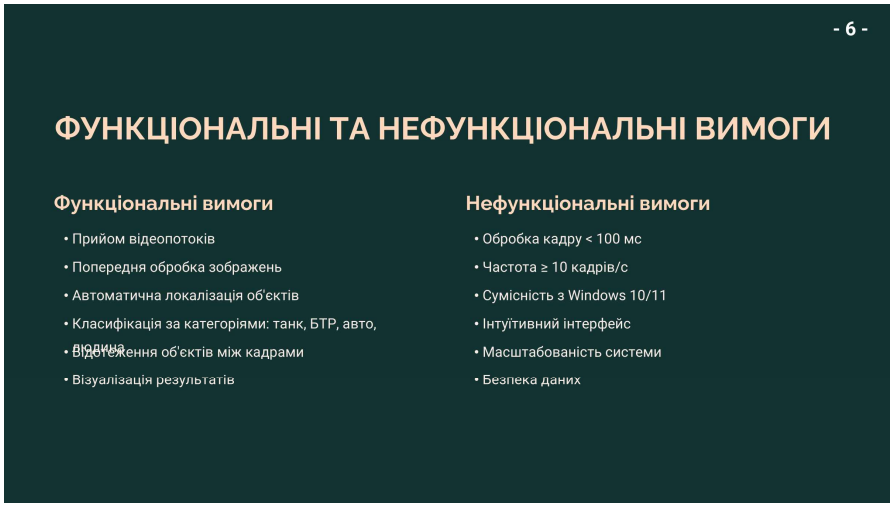


Рисунок Б.6– Слайд 6

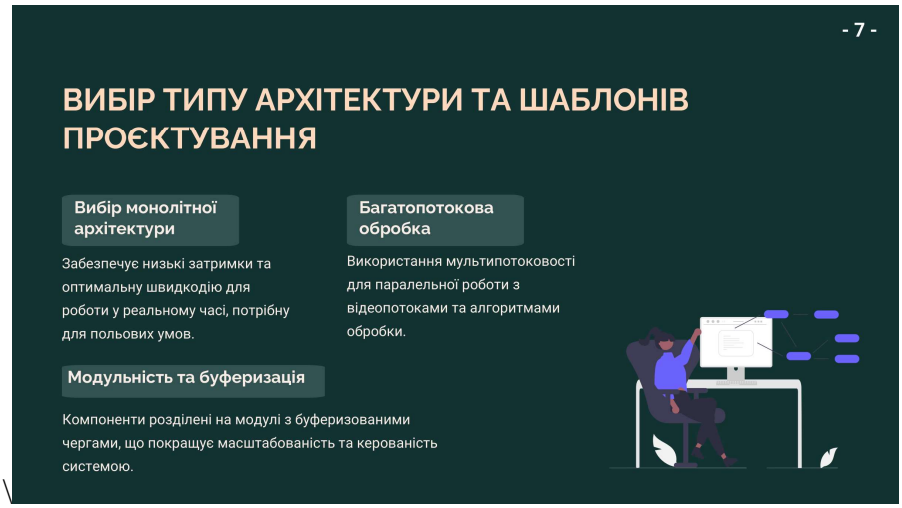


Рисунок Б.7– Слайд 7

Рисунок Б.8– Слайд 8

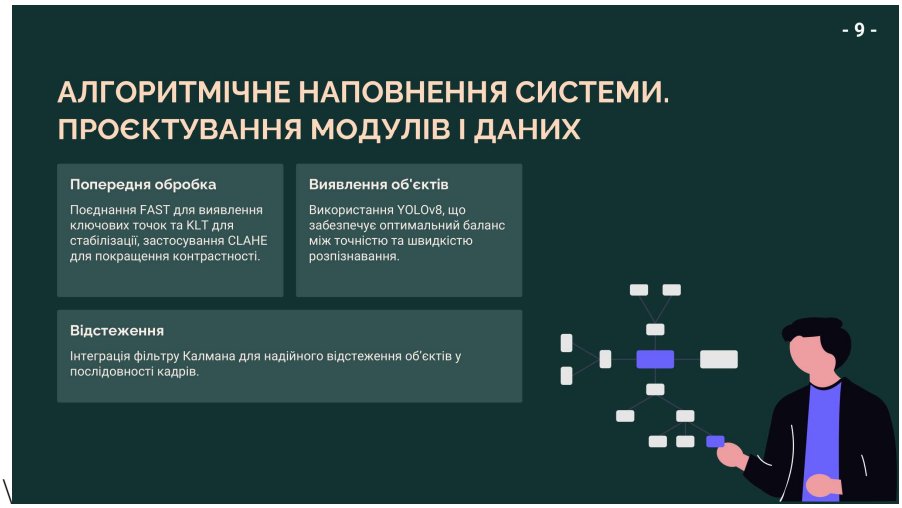


Рисунок Б.9 – Слайд 9

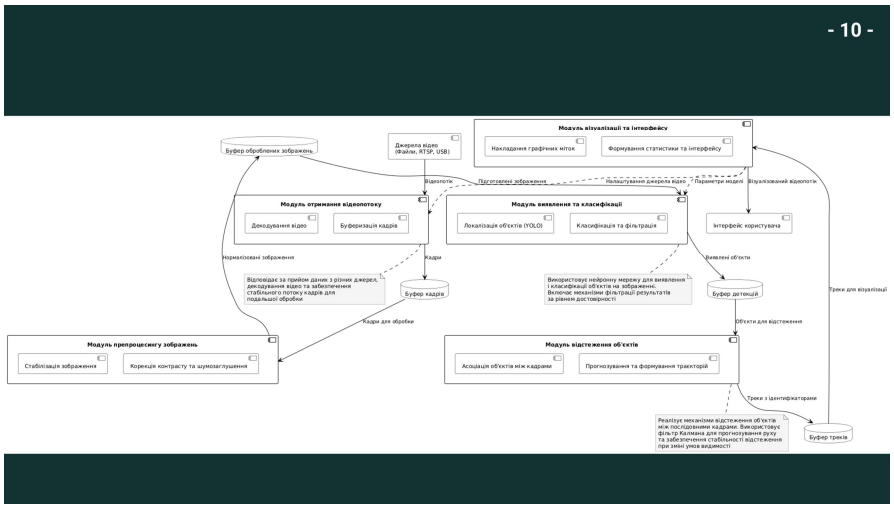


Рисунок Б.10 – Слайд 10

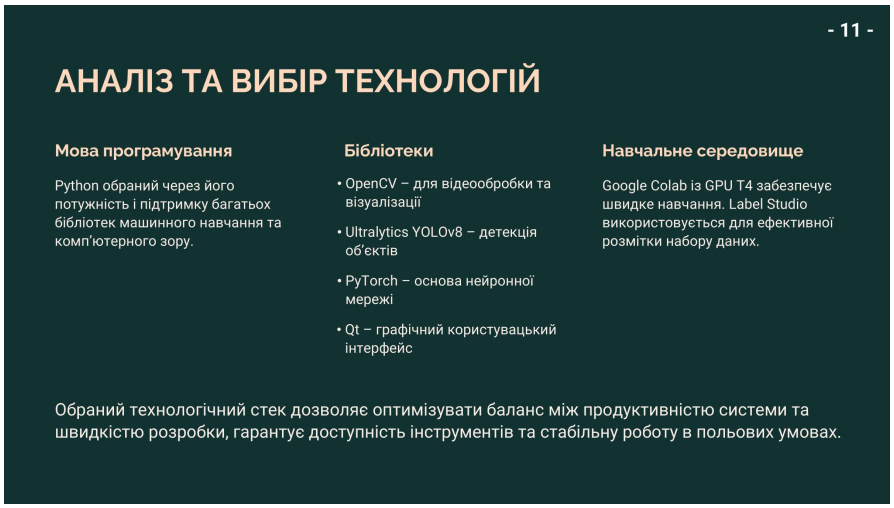


Рисунок Б.11 – Слайд 11

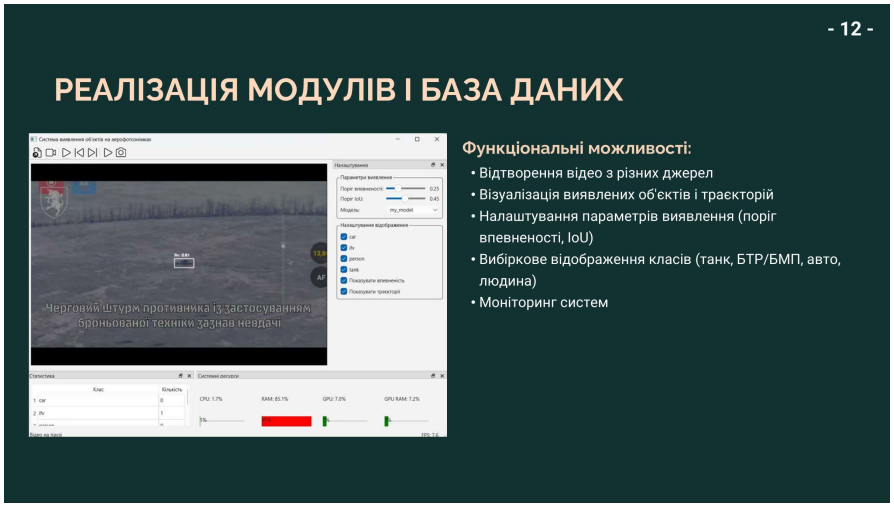


Рисунок Б.12 – Слайд 12

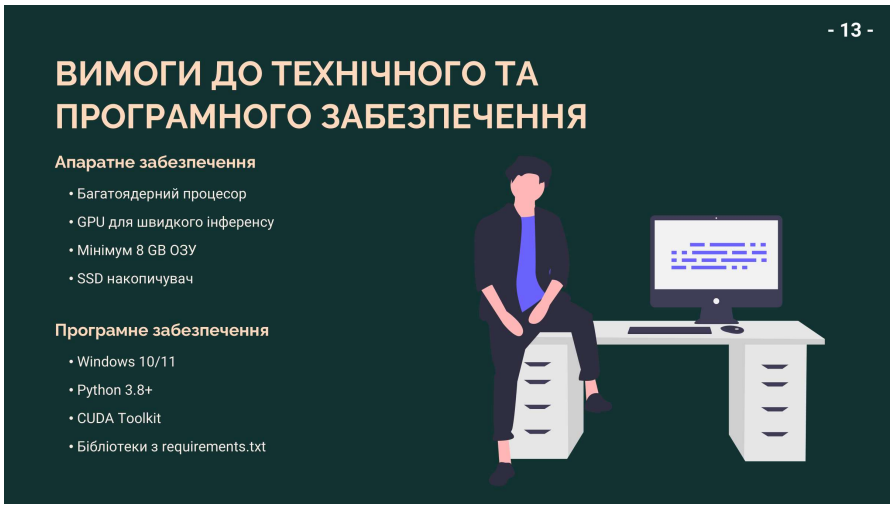


Рисунок Б.13 – Слайд 13

Рисунок Б.14 – Слайд 14

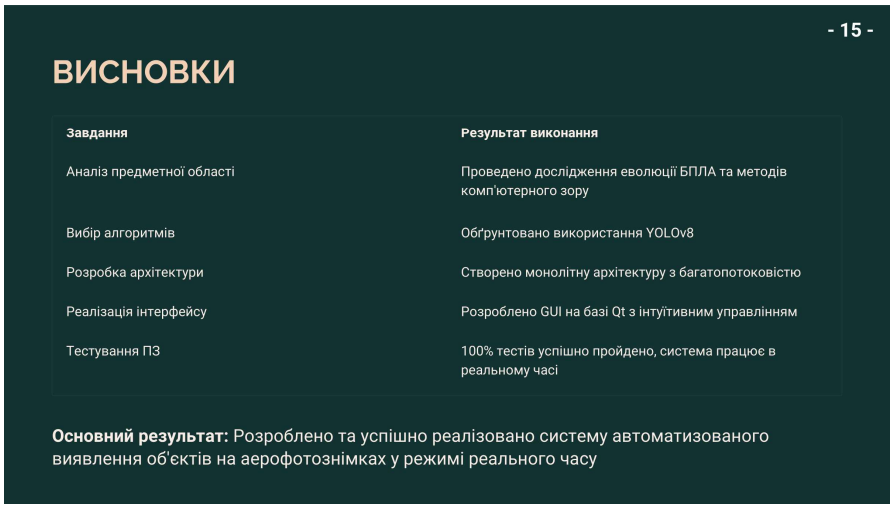


Рисунок Б.15 – Слайд 15

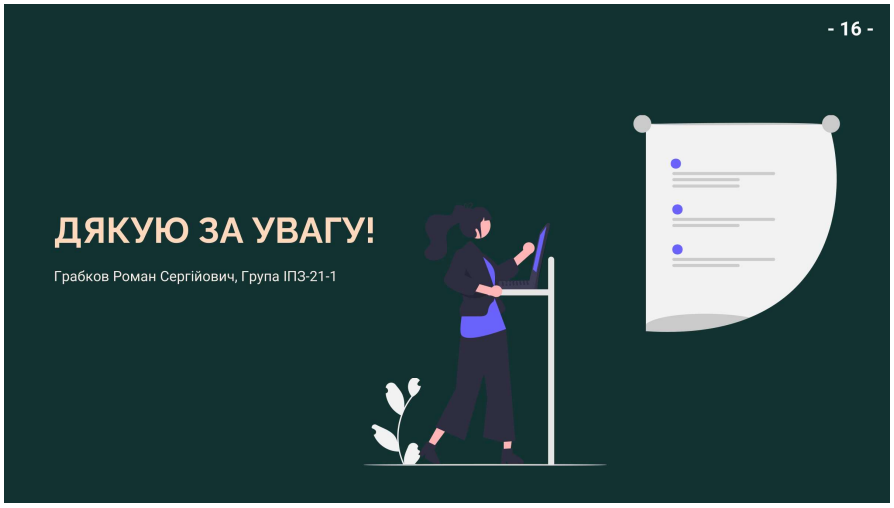


Рисунок Б.16 – Слайд 16

ДОДАТОК В

(обов'язковий)

ПРОГРАМНИЙ КОД ОСНОВНИХ МОДУЛІВ

interface.py

```python
"""
Basic prototype of user interface for aerial imagery object detection system using PyQt6
"""
import sys
import os
import cv2
import numpy as np
import time
import torch
from ultralytics import YOLO
from pathlib import Path
from PyQt6.QtWidgets import (QApplication, QMainWindow, QWidget, QLabel,
                             QVBoxLayout, QHBoxLayout, QGridLayout,
                             QFileDialog, QComboBox, QSlider, QGroupBox,
                             QCheckBox, QStatusBar, QDockWidget, QMessageBox)
from PyQt6.QtCore import Qt, QTimer, pyqtSignal, QThread
from PyQt6.QtGui import QImage, QPixmap, QIcon, QAction
class YOLOv8Detector:
    """YOLOv8 detector class for real-time object detection"""
    def __init__(self, model_path: str = "models/my_model.pt"):
        self.model_path = Path(model_path)
        if not self.model_path.exists():
            raise FileNotFoundError(f"Model file not found at {self.model_path}")
        self.device = "cuda" if torch.cuda.is_available() else "cpu"
        try:
            self.model = YOLO(self.model_path)
            print(f"Model loaded successfully on {self.device}")
        except Exception as e:
            print(f"Error loading model: {str(e)}")
            raise
        self.classes = self.model.names
    @staticmethod
    def get_available_models():
        models_dir = Path("models")
        if not models_dir.exists():
            return []
        return [model.stem for model in models_dir.glob("*.pt")]
    def detect(self, frame):
        if frame is None:
            return []
        try:
            results = self.model(frame, conf=0.25)[0]
            detections = []
```

```
            for box in results.boxes:
                x1, y1, x2, y2 = map(int, box.xyxy[0].tolist())
                conf = float(box.conf[0])
                cls_id = int(box.cls[0])
                cls_name = self.classes[cls_id]
                detections.append({
                    'bbox': (x1, y1, x2, y2),
                    'class_id': cls_id,
                    'class_name': cls_name,
                    'confidence': conf
                })
            return detections
        except Exception as e:
            print(f"Error during detection: {str(e)}")
            return []
class VideoThread(QThread):
    """Thread for video processing"""
    update_frame = pyqtSignal(np.ndarray, list)
    update_fps = pyqtSignal(float)
    error_occurred = pyqtSignal(str)
    video_ended = pyqtSignal()
    def __init__(self, parent=None):
        super().__init__(parent)
        self.running = False
        self.source = 0
        self.paused = False
        self.cap = None
        try:
            self.detector = YOLOv8Detector()
        except Exception as e:
            self.error_occurred.emit(f"Failed to initialize model: {str(e)}")
            self.detector = None
        self.fps = 0
        self.process_enabled = True
        self.current_frame = 0
    def process_frame(self, frame):
        if frame is None:
            return
        detections = []
        if self.process_enabled:
            try:
                detections = self.detector.detect(frame)
            except Exception as e:
                self.error_occurred.emit(f"Detection error: {str(e)}")
                return
        self.update_frame.emit(frame, detections)
    def run(self):
        if self.detector is None:
            self.error_occurred.emit("Detector not initialized properly")
            return
        try:
            source = int(self.source)
        except ValueError:
```

```
            source = self.source
        self.cap = cv2.VideoCapture(source)
        if not self.cap.isOpened():
            self.error_occurred.emit(f"Error: Could not open video source {source}")
            return
        self.running = True
        frames_count = 0
        start_time = cv2.getTickCount()
        while self.running:
            if not self.paused:
                ret, frame = self.cap.read()
                if not ret:
                    if isinstance(self.source, str) and os.path.isfile(self.source):
                        self.cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
                        self.current_frame = 0
                        continue
                    else:
                        self.video_ended.emit()
                        break
                self.current_frame += 1
                self.process_frame(frame)
                frames_count += 1
                if frames_count >= 10:
                    end_time = cv2.getTickCount()
                    elapsed_time = (end_time - start_time) / cv2.getTickFrequency()
                    self.fps = frames_count / elapsed_time
                    self.update_fps.emit(self.fps)
                    frames_count = 0
                    start_time = cv2.getTickCount()
            self.msleep(15)
        if self.cap is not None:
            self.cap.release()
class MainWindow(QMainWindow):
    """Main application window"""
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Система виявлення об'єктів на аерофотознімках")
        self.setMinimumSize(1000, 700)
        self.current_frame = None
        self.class_counts = {}
        # Initialize UI
        self.init_ui()
        # Initialize video thread
        self.video_thread = VideoThread()
        self.video_thread.video_ended.connect(self.on_video_ended)
        # Connect signals
        self.video_thread.update_frame.connect(self.update_frame)
        self.video_thread.update_fps.connect(self.update_fps)
        self.video_thread.error_occurred.connect(self.show_error)
        # Start video
        self.start_video()
    def init_ui(self):
        self.central_widget = QWidget(self)
```

```
        self.setCentralWidget(self.central_widget)
        main_layout = QVBoxLayout(self.central_widget)
        # Video display
        self.display_label = QLabel()
        self.display_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
        self.display_label.setStyleSheet("background-color: black;")
        self.display_label.setMinimumSize(640, 480)
        main_layout.addWidget(self.display_label)
        # Create UI components
        self.create_toolbar()
        self.create_settings_panel()
        # Status bar
        self.statusBar = QStatusBar()
        self.setStatusBar(self.statusBar)
        self.fps_label = QLabel("FPS: 0")
        self.statusBar.addPermanentWidget(self.fps_label)
    def create_toolbar(self):
        self.toolbar = self.addToolBar("Головна панель інструментів")
        # Add toolbar actions
        self.add_toolbar_action("document-open", "Відкрити файл", self.open_file)
        self.add_toolbar_action("camera-video", "Використати камеру", self.use_camera)
        self.toolbar.addSeparator()
        self.add_toolbar_action("media-playback-start", "Пауза", self.toggle_play_pause)
        self.add_toolbar_action("media-skip-backward", "Попередній кадр", self.prev_frame)
        self.add_toolbar_action("media-skip-forward", "Наступний кадр", self.next_frame)
        self.toolbar.addSeparator()
        self.add_toolbar_action("media-playback-start", "Старт/Стоп", self.toggle_processing)
        self.add_toolbar_action("camera-photo", "Знімок екрана", self.take_screenshot)
    def add_toolbar_action(self, icon_name, text, slot):
        action = QAction(QIcon.fromTheme(icon_name) or QIcon(), text, self)
        action.triggered.connect(slot)
        self.toolbar.addAction(action)
        return action
    def create_settings_panel(self):
        self.settings_dock = QDockWidget("Налаштування", self)
        settings_container = QWidget()
        settings_layout = QVBoxLayout(settings_container)
        # Add settings groups
        self.add_detection_settings(settings_layout)
        self.add_display_settings(settings_layout)
        self.settings_dock.setWidget(settings_container)
        self.addDockWidget(Qt.DockWidgetArea.RightDockWidgetArea, self.settings_dock)
    def add_detection_settings(self, layout):
        group = QGroupBox("Параметри виявлення")
        group_layout = QGridLayout()
        # Add detection settings controls
        self.add_slider_control(group_layout, "Поріг впевненості:", 25, 0)
        self.add_slider_control(group_layout, "Поріг IoU:", 45, 1)
        self.add_model_selector(group_layout, 2)
        group.setLayout(group_layout)
        layout.addWidget(group)
    def add_display_settings(self, layout):
        group = QGroupBox("Налаштування відображення")
```

```
        group_layout = QVBoxLayout()
        # Add class checkboxes
        for cls in ["tank", "ifv", "car", "person"]:
            checkbox = QCheckBox(cls)
            checkbox.setChecked(True)
            group_layout.addWidget(checkbox)
        group.setLayout(group_layout)
        layout.addWidget(group)
    def start_video(self):
        if not self.video_thread.isRunning():
            self.video_thread.start()
            self.statusBar.showMessage("Обробка відео запущена")
    def toggle_processing(self):
        self.video_thread.process_enabled = not self.video_thread.process_enabled
        status = "увімкнено" if self.video_thread.process_enabled else "вимкнено"
        self.statusBar.showMessage(f"Обробку {status}")
    def take_screenshot(self):
        if self.current_frame is None:
            return
        file_path, _ = QFileDialog.getSaveFileName(
            self,
            "Зберегти знімок екрана",
            f"screenshot_{int(time.time())}.png",
            "Зображення (*.png *.jpg)"
        )
        if file_path:
            cv2.imwrite(file_path, self.current_frame)
            self.statusBar.showMessage(f"Знімок збережено: {file_path}")
    def show_error(self, message: str):
        QMessageBox.critical(self, "Error", message)
        if "Failed to initialize" in message:
            self.toggle_processing()
    def closeEvent(self, event):
        self.video_thread.stop()
        event.accept()
    def toggle_play_pause(self):
        if self.video_thread.paused:
            self.video_thread.resume()
            self.statusBar.showMessage("Відтворення відео")
        else:
            self.video_thread.pause()
            self.statusBar.showMessage("Відео на паузі")
    def on_video_ended(self):
        self.video_thread.pause()
        self.statusBar.showMessage("Відео завершено")
    def prev_frame(self):
        self.video_thread.prev_frame()
    def next_frame(self):
        self.video_thread.next_frame()
if __name__ == "__main__":
    app = QApplication(sys.argv)
    app.setStyle("Fusion")
    window = MainWindow()
```

```
    window.show()
    sys.exit(app.exec())
```

core/detection.py

```python
"""
Detection module for object detection using YOLOv8.
Handles model loading, inference, and detection processing.
"""
import cv2
import numpy as np
from pathlib import Path
from typing import List, Dict, Any
from ultralytics import YOLO
class YOLOv8Detector:
    """YOLOv8 object detector"""
    def __init__(self, model_path: str):
        self.model_path = model_path
        self.model = None
        self.classes = {}
        self.load_model()
    def load_model(self) -> bool:
        try:
            self.model = YOLO(self.model_path)
            self.classes = self.model.names
            return True
        except Exception as e:
            print(f"Error loading model: {str(e)}")
            return False
    def detect(self, frame: np.ndarray) -> List[Dict[str, Any]]:
        if self.model is None:
            return []
        try:
            results = self.model(frame, verbose=False)[0]
            detections = []
            for r in results.boxes.data.tolist():
                x1, y1, x2, y2, conf, cls = r
                detections.append({
                    'bbox': (x1, y1, x2, y2),
                    'confidence': conf,
                    'class_id': int(cls),
                    'class_name': self.classes[int(cls)]
                })
            return detections
        except Exception as e:
            print(f"Error during detection: {str(e)}")
            return []
    @staticmethod
    def get_available_models() -> List[str]:
        models_dir = Path("models")
        if not models_dir.exists():
            return []
```

```
        return [f.stem for f in models_dir.glob("*.pt")]
```

core/config.py

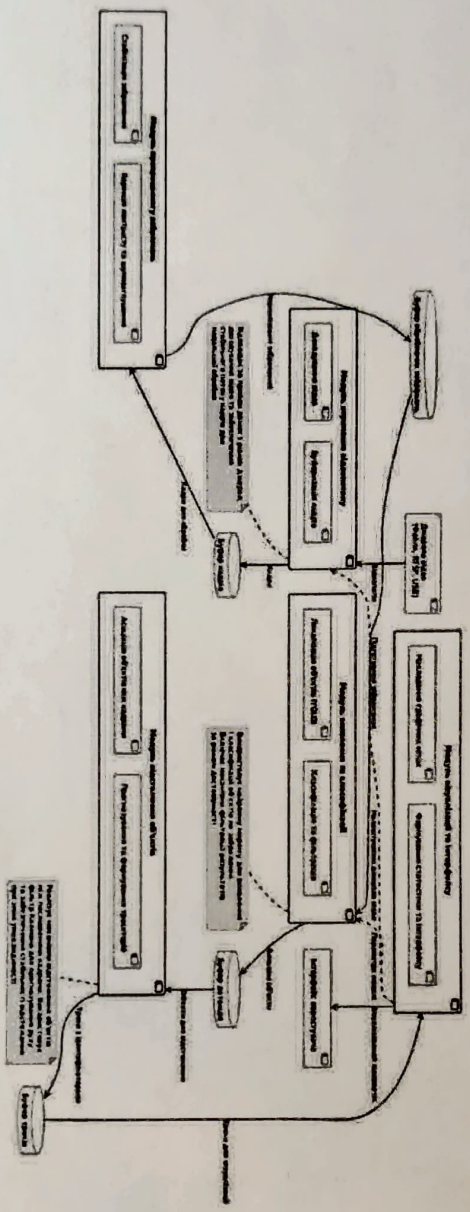
```python
"""
Configuration module.
Contains global configuration settings.
"""
from pathlib import Path
class Config:
    """Global configuration settings"""
    # Model settings
    MODEL_PATH = "models/my_model.pt"
    CONFIDENCE_THRESHOLD = 0.25
    IOU_THRESHOLD = 0.45
    # Processing settings
    TARGET_SIZE = (640, 640)
    FPS_TARGET = 30
    # Display settings
    COLORS = {
        'tank': (0, 255, 0),      # Green
        'ifv': (255, 0, 0),       # Blue
        'car': (0, 0, 255),       # Red
        'person': (255, 255, 0)    # Cyan
    }
    # File paths
    MODELS_DIR = Path("models")
    OUTPUT_DIR = Path("output")
    @classmethod
    def setup(cls):
        """Setup configuration"""
        cls.MODELS_DIR.mkdir(exist_ok=True)
        cls.OUTPUT_DIR.mkdir(exist_ok=True)
```

data_pypeline.py

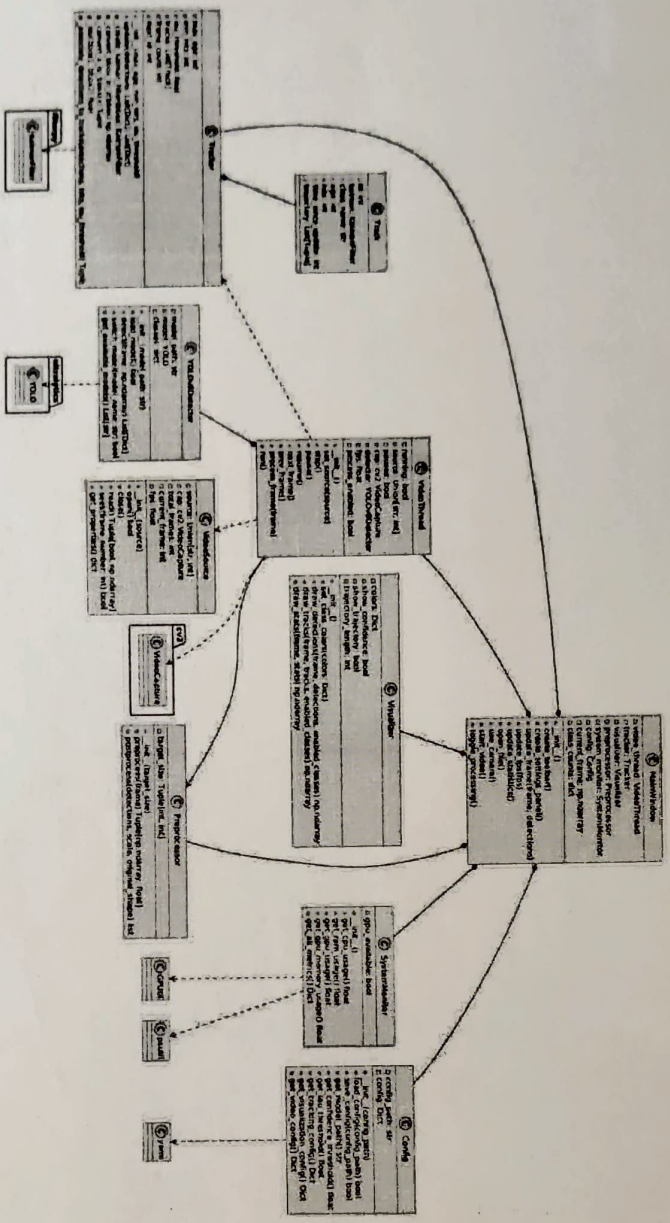
```python
import cv2, numpy as np
from typing import Union, Iterator, Tuple, Optional

class VideoSource:
    def __init__(self, src: Union[str, int] = 0):
        self.src = int(src) if str(src).isdigit() else src
        self.cap: Optional[cv2.VideoCapture] = None
        self.total = self.pos = 0

    def open(self) -> bool:
        self.cap = cv2.VideoCapture(self.src)
        if not self.cap.isOpened(): return False
        if isinstance(self.src, str):
            self.total = int(self.cap.get(cv2.CAP_PROP_FRAME_COUNT))
        return True
```

```
def close(self):
    self.cap and self.cap.release()

def read(self) -> Tuple[bool, Optional[np.ndarray]]:
    if not self.cap: return False, None
    ok, frame = self.cap.read()
    if not ok and isinstance(self.src, str):
        self.cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
        self.pos = 0
        ok, frame = self.cap.read()
    if ok: self.pos = (self.pos + 1) % max(self.total, 1)
    return ok, frame

def seek(self, idx: int) -> bool:
    if not (self.cap and 0 <= idx < self.total): return False
    self.cap.set(cv2.CAP_PROP_POS_FRAMES, idx)
    self.pos = idx
    return True

def __iter__(self) -> Iterator[np.ndarray]: return self
def __next__(self) -> np.ndarray:
    ok, frame = self.read()
    if not ok: raise StopIteration
    return frame
```

ГРАФІЧНА ЧАСТИНА



					КвРПЗ.2101072.01.04.ВД	
Змн.	Арк.	№ докум.	Підпис	Дата	Програмна система для автоматизованого виявлення та класифікації об'єктів на аерофотознімках у режимі реального часу Діаграма класів	Літ. / Маса. / Масштаб
Розробив		Грабков Р.С.		01.06		Аркуш 1 / Аркушів 3
Керівник		Бедратюк Л. П.		01.06		
						ХНУ, ІПЗ-21-1
Н. Контр.		Яшина О.М.		01.06		
Зав. каф.		Бедратюк Л. П.		01.06		



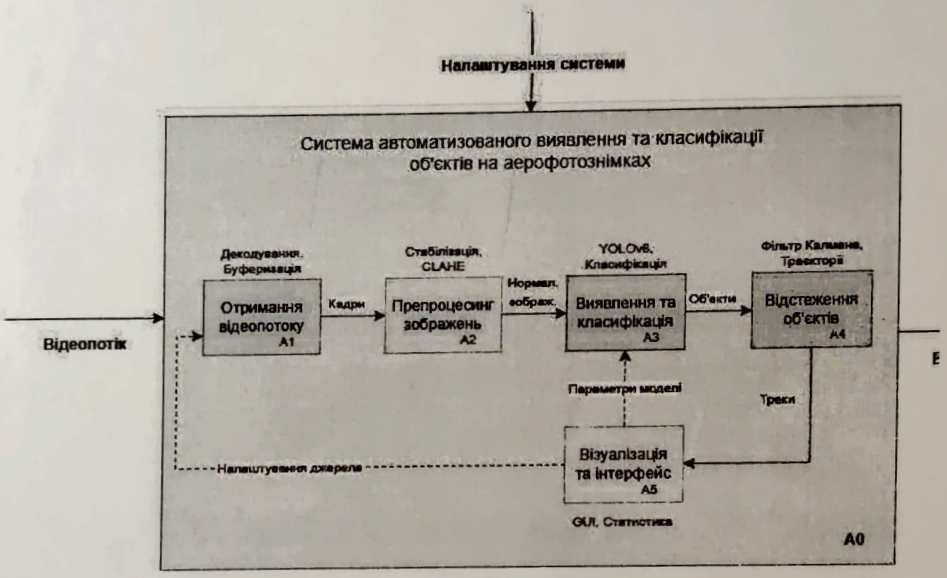
КвРІПЗ.2101072.01.04.ВД

Змн.	Арк.	№ докум.	Підпис	Дата
Розробив		Грабков Р.С.		01.06
Керівник		Бедратюк Л. П.		01.06
				01.06
Н. Контр.		Яшина О.М.		
Зав. каф.		Бедратюк Л. П.		01.06

Програмна система для автоматизованого виявлення та класифікації об'єктів на аерофотознімках у режимі реального часу
Діаграма класів та інтерфейсів

Літ.	Маса.	Масштаб
Аркуш 2	Аркушів 3	

ХНУ, ІПЗ-21-1



					КвРІПЗ.2101072.01.04.ВД			
					Програмна система для автоматизованого виявлення та класифікації об'єктів на аерофотознімках у режимі реального часу Діаграма IDEF0 декомпозиції системи	*Літ.*	*Маса.*	*Масштаб*
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*				
Розробив		Грабков Р.С.		01.06		Аркуш 3	Аркушів 3	
Керівник		Бедратюк Л. П.		01.06				
Н. Контр.		Яшина О.М.		01.06		ХНУ, ІПЗ-21-1		
Зав. каф.		Бедратюк Л. П.		01.06				

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Грабкова Романа Сергійовича
факультет ІТ, IVкурс, група ІПЗ-21-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності в Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлена. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщена та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-обчислювального комплексу StrikePlagiarism та/або програмно-технічного засобу AntiPlagiarism і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

01.06.2025
дата

підпис

Mon Jun 02 01:12:06 EEST 2025, Форкун Юрій Вікторович, Хмельницький національний університет, ХНУ

Anti-Plagiarism (UA) v-15.281 Educational

The maximum coincidence with one document 3.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 11%**

	Document		Sum coincidence on the DB	
ID: 242708 Title: Програмна система для автоматизованого виявлення та класифікації об'єктів на аерофотознімках в режимі реального часу Added in a DB: 2025-06-02 Authors: ГРАБКОВ Роман Heads: Бедратюк Леонід Петрович, д-р фіз.-мат. наук, професор Consultants: Opponents:	Symbols	Lexemes	Symbols	Lexemes
	107873	1591	4943 (5%)	62 (4%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Грабков Роман

Співавтор:

Назва: БКР_Програмна система для автоматизованого виявлення та класифікації об'єктів на аерофотознімках в режимі реального часу_

Науковий керівник:

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:3.6%

Коефіцієнт подібності 2:0.2%

Мікропробіли: 0

Заміна букв: 0

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2025-06-01 14:26:53.0

Після аналізу Звіту подібності констатую наступне:

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обґрунтування:

Дата
01.06.2025

експерт

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Грабков Роман Сергійович

Тема Програмна система для автоматизованого виявлення та класифікації об'єктів на аерофотознімках в режимі реального часу.

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3 ; кількість сторінок записки 82

1.Короткий зміст пояснювальної записки та прийнятих рішень Кваліфікаційна робота присвячена розробці програмної системи для автоматизованого виявлення та класифікації об'єктів на аерофотознімках у реальному часі. Актуальність теми обумовлена зростаючим використанням безпілотних літальних апаратів у різних сферах, зокрема в військовій, де автоматизація процесів розпізнавання об'єктів стає критично важливою. Мета роботи полягає у створенні ефективної системи, що використовує сучасні методи комп'ютерного зору та глибокого навчання. Для досягнення цієї мети поставлено завдання, які охоплюють аналіз предметної області, обґрунтування вибору алгоритмів, розробку архітектури системи, реалізацію алгоритмів виявлення та тестування програмного забезпечення.

2. Висновок про відповідність роботи поставленому завданню Завдання роботи охоплюють: аналіз предметної області, обґрунтування вибору алгоритмів, розробку архітектури системи, реалізацію алгоритмів виявлення та тестування. Результати засвідчують успішне виконання всіх завдань: проведено ґрунтовний аналіз, розроблено монолітну архітектуру з багатопотоковістю, реалізовано алгоритми та проведено тестування. Отримані результати повністю відповідають меті роботи.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи Розділ 1 систематизує знання про еволюцію технологій повітряного спостереження, висвітлює сучасні методи комп'ютерного зору та сфери їх практичного застосування. Комплексний огляд наявних програмних рішень надає чітке розуміння ключових підходів до автоматизованого виявлення об'єктів. Розділ демонструє глибоке розуміння предметної області, містить релевантні посилання на джерела та підкріплений конкретними прикладами використання безпілотних літальних апаратів у різних галузях економіки. Автор наводить статистичні дані щодо ефективності застосування дронів, що переконливо обґрунтовує актуальність досліджуваних технологій і підтверджує практичну значущість роботи.

Розділ 2 представляє чітко структуровану архітектурну концепцію системи виявлення об'єктів. Автор провів ретельний порівняльний аналіз архітектурних підходів, результати якого систематизовано у детальній таблиці з критеріями оцінки та обґрунтуванням вибору оптимального рішення. Вибір монолітної архітектури з елементами багатопотоковості є технічно обґрунтованим та відповідає специфіці завдань комп'ютерного зору.

Розділ містить вичерпний опис структури системи, детальну характеристику обраних технологій та їх інтеграції, а також комплексний опис інтерфейсу користувача. Вимоги

до інтерфейсу користувача детально описані з урахуванням функціональних потреб. Незначними недоліками є відсутність графічних схем архітектурних підходів та візуальних макетів інтерфейсу користувача, що могло б покращити сприйняття та розуміння проектних рішень

Розділ 3 всебічно висвітлює етапи програмної реалізації системи, включаючи детальний опис процесу підготовки даних, методики навчання моделей машинного навчання та архітектури програмних модулів. Автор представив фрагменти коду ключових модулів системи, що демонструє практичну реалізацію теоретичних концепцій.

Особливу увагу приділено комплексному тестуванню системи – детально описано методологію тестування кожного з модулів, проведено експлораторне ручне тестування та метричне тестування точності моделей, результати якого систематизовано у відповідній таблиці. Розділ містить критерії оцінки функціональності та результати верифікації роботи алгоритмів. Незначним недоліком є відсутність фрагментів коду юніт-тестів, які б проілюстрували практичні аспекти тестування окремих компонентів системи.

4. Позитивні сторони роботи Кваліфікаційна робота демонструє оригінальність та якість рішень, зокрема, в розробці системи автоматизованого виявлення об'єктів. Описано історію розвитку технологій, що свідчить про глибоке розуміння предметної області. Вибір технологій та архітектури системи обґрунтований, а опис інтерфейсу користувача акцентує увагу на зручності взаємодії.

5. Негативні сторони роботи Незначними недоліками роботи є відсутність графічних схем архітектурних підходів у другому розділі, візуальних макетів інтерфейсу користувача та фрагментів коду юніт-тестів у третьому розділі, що могло б покращити візуальне сприйняття та практичне розуміння окремих аспектів системи.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано відповідно до теми кваліфікаційної роботи та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота є актуальною та має високу практичну значущість, оскільки розроблена система може бути ефективно використана для автоматизованого виявлення об'єктів військової техніки у реальному часі. Зміст роботи повністю відповідає темі та поставленим завданням, демонструючи інноваційний підхід до вирішення складних завдань комп'ютерного зору. Робота характеризується високою якістю виконання всіх розділів та комплексним підходом до дослідження..

8. Інші зауваження Відсутні.

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота заслуговує оцінки "відмінно". Вона виконана в повному обсязі з дотриманням усіх основних вимог, демонструє глибоке розуміння предметної області, містить ґрунтовний аналіз та якісну практичну реалізацію. Незначні недоліки не впливають на загальну високу якість дослідження.

РЕЦЕНЗЕНТ Чешун Віктор Миколайович

канд. техн. наук, доц., доцент кафедри кібербезпеки

"4" червня 2025 р. _____
(підпис)

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ КАФЕДРИ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Назва кваліфікаційної роботи «Програмна система для автоматизованого виявлення та класифікації об'єктів на аерофотознімках в режимі реального часу»
Автор Грабков Роман Сергійович
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»
Рівень вищої освіти Бакалавр
Спеціальність 121 – Інженерія програмного забезпечення
Науковий керівник: Бедратюк Леонід Петрович, д-р фіз.-мат. наук, професор

На основі аналізу кваліфікаційної роботи на дотримання вимог академічної доброчесності (у т.ч. відсутності ознак академічного плагіату) з урахуванням результатів перевірки роботи спеціалізованим програмним засобом(ами) комісія зробила такий висновок:

№	Висновок	Позначка про відповідність
1	Ознаки академічного плагіату	
1.1	Запозичення, виявлені в роботі, є законними і не є академічним плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних, якщо потрібно). Робота приймається до захисту.	
1.2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована.	
1.3	Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота може бути допущена до захисту після того як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат.	
1.4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
2	Інші види порушень академічної доброчесності	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) усі запозичення фрагментарні, або мають належним чином оформленні посилання;

2) система фіксувала технічні особливості (наприклад, поєднання латиниці й українських індексів), а не модифікацію тексту.

Сумарний обсяг запозичень — 3.8%, що відповідає науковим стандартам і не впливає на якість кваліфікаційної роботи.

Дата 6.06.2021

Завідувач кафедри _____ Підпис _____ Ім'я, ПРІЗВИЩЕ

Гарант освітньої програми _____ Підпис _____ Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи _____ Підпис _____ Ім'я, ПРІЗВИЩЕ