Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка програмного модуля автоматичного визначення характеристик об'єктів на динамічних зображеннях із застосуванням нейромереж»

Виконав: студент 4 курсу
групи 1ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Хитрич С.С.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Рейда О.М.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ОТ Томчук М.А.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри ПЗ Романюк О.Н.
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н. професор Романюк О.Н.
"21" березня 2025 р.

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

Хитрич Станіславу Сергійовичу

1. Тема роботи – «Розробка програмного модуля автоматичного визначення характеристик об'єктів на динамічних зображеннях із застосуванням нейромереж».

Керівник роботи: Рейда Олександр Миколайович, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 року № 97.

2. Строк подання студентом роботи 13 червня 2025 року.

3. Вихідні дані до роботи: середовище розробки Visual Studio Code, мова розробки Python 3+, операційна система – Windows 11, згорткові нейроні мережі.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз та постановка задачі; аналіз предметної області; аналіз аналогів; розробка алгоритму роботи системи; розробка модуля детекції та трекінгу об'єктів; розробка алгоритму аналізу параметрів об'єктів; розробка інтерфейсу; тестування додатку; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: графічний інтерфейс веб частини додат
модель роботи додатку; блок-схеми алгоритмів детекції, трекінгу та аналі
тестування додатку.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Рейда О.М., к.т.н., доцент кафедри ПЗ	24.03.2025	02.06.2025

7. Дата видачі завдання 24 березня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз розвитку технологій обробки динамічних зображень та постановка задач дослідження	26.03.2025 – 03.04.2025	вик.
2	Розробка моделі та алгоритмів програмного продукту	04.04.2025 – 16.04.2025	вик.
3	Розробка програмного модулів визначення параметрів об'єктів на динамічних зображеннях	17.04.2025 – 17.05.2025	вик.
4	Тестування програми	18.05.2025 – 28.05.2025	вик.
5	Оформлення матеріалів до захисту БКР	28.05.2025 – 10.06.2025	вик.

Студент _____ Хитрич С.С
(підпис) (прізвище та ініціа

Керівник бакалаврської кваліфікаційної роботи _____ Рейда О.М.
(підпис) (прізвище та ініціал

Анотація

УДК 004.932.2:004.89:519.95

Бакалаврська кваліфікаційна робота складається з 104 сторінки формату А4 на яких 22 рисунки, 2 таблиці, список використаних джерел містить 22 наіменування.

У бакалаврській кваліфікаційній роботі розроблено програмні модулі для визначення параметрів об’єктів на динамічних зображеннях із використанням алгоритмів комп’ютерного зору та методів машинного навчання.

Розроблене програмне забезпечення може використовуватися в системах відеоспостереження, аналізу руху транспорту, автоматизованому контролі виробничих процесів та медичній діагностиці на основі відеоаналізу.

Реалізацію здійснено в середовищі Visual Studio Code, з мовою програмування Python версії 3+ та бібліотек OpenCV, TensorFlow та YOLO для детекції та відстеження об'єктів. Розроблено графічний інтерфейс з використанням HTML, CSS та JavaScript, що забезпечує зручний доступ до функцій аналізу та налаштувань. Передбачено можливість експорту результатів у формати CSV, JSON та XSLX, а також інтерактивну візуалізацію даних.

Проведено тестування програмного забезпечення для оцінки точності та продуктивності реалізованих алгоритмів.

Отримані в бакалаврській кваліфікаційній роботі результати можна використовувати для систем відеоспостережння, аналізу руху транспорту, авторматизованому контролі виробничих процесів та медичній діагностиці на основі відеоаналізу.

Ключові слова: згоркові нейроні мережі, штучний інтелект, трекінг, характеристики, диамічні зображення.

Annotation

UDC 004.932.2:004.89:519.95

The bachelor's thesis consists of 104 A4 pages, including 22 figures, 2 tables, and a list of 22 references.

The thesis presents the development of software modules for determining object parameters in dynamic images using computer vision algorithms and machine learning methods.

The developed software can be applied in video surveillance systems, traffic motion analysis, automated industrial process control, and video-based medical diagnostics.

The implementation was carried out in Visual Studio Code using Python 3+ and the OpenCV, TensorFlow, and YOLO libraries for object detection and tracking. A graphical user interface was developed using HTML, CSS, and JavaScript, providing convenient access to analysis features and settings. The system supports exporting results to CSV, JSON, and XLSX formats, as well as interactive data visualization.

The software was tested to evaluate the accuracy and performance of the implemented algorithms.

The results obtained in the thesis can be used in video surveillance systems, traffic analysis, automated production control, and video-based medical diagnostics.

Keywords: convolutional neural networks, artificial intelligence, tracking, characteristics, dynamic images.

ЗМІСТ

ВСТУП 4
1 АНАЛІЗ РОЗВИТКУ ТЕХНОЛОГІЙ ОБРОБКИ ДИНАМІЧНИХ ЗОБРАЖЕНЬ І ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ 7
1.1 Аналіз стану технологій обробки динамічних зображень для визначення параметрів об'єктів 7
1.2 Порівняльний аналіз аналогів 8
1.3 Аналіз методів розв’язання задачі 11
1.4 Постановка задач дослідження 12
1.5 Висновки 13
2 РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ 15
2.1 Аналіз вхідних даних системи 15
2.2 Розробка моделі системи 17
2.3 Розробка алгоритму динамічного аналізу та виявлення об’єктів 19
2.4 Розробка алгоритму експорту результатів 22
2.5 Висновки 24
3. РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ ВИЗНАЧЕННЯ ПАРАМЕТРІВ ОБ'ЄКТІВ НА ДИНАМІЧНИХ ЗОБРАЖЕННЯХ 25
3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення 25
3.2 Розробка інтерфейсу 26
3.3 Розробка модуля детекції та трекінгу об'єктів на динамічних зображеннях 33
3.4 Розробка модуля аналізу кінетичних параметрів об'єктів на динамічних зображеннях 36
3.5 Висновки 39
4 ТЕСТУВАННЯ ПРОГРАМИ 41
4.1 Тестування системи 41

4.2 Розробка інструкції користувача 49
4.3 Висновки 52
ВИСНОВКИ 53
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 55
ДОДАТКИ 58
Додаток А – ТЕХНІЧНЕ ЗАВДАННЯ ..... **Error! Bookmark not defined.**
Додаток Б – ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ ..... **Error! Bookmark not defined.**
Додаток В – ЛІСТИНГ ПРОГРАМИ 64
Додаток Г – ІЛЮСТРАТИВНА ЧАСТИНА 91

ВСТУП

Обґрунтування вибору теми дослідження. Сучасний розвиток технологій комп’ютерного зору та машинного навчання відкриває нові можливості для аналізу динамічних зображень. Однією з ключових проблем у цій сфері є точне визначення параметрів об'єктів у відеопотоці, що має важливе значення для багатьох галузей, зокрема систем відеоспостереження, транспортної аналітики, автоматизованого контролю виробничих процесів і медичної діагностики [1].

Традиційні методи обробки відео часто обмежені жорсткими алгоритмами та низькою адаптивністю до змінних умов зйомки. Використання сучасних підходів, заснованих на глибокому навчанні нейронних мереж, дозволяє значно підвищити точність і швидкість аналізу. Інтеграція алгоритмів машинного зору в програмні модулі забезпечує можливість детекції, відстеження та оцінки параметрів об’єктів у режимі реального часу [2,3].

Актуальність даної розробки зумовлена необхідністю розробки програмного модуля автоматичного визначення характеристик об’єктів на динамічних зображеннях із застосуванням нейромереж для підвищення ефективності автоматизованого аналізу об'єктів на динамічних зображеннях та потокових даних з можливістю локального розгортання.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Мета роботи полягає у підвищення ефективності обробки та аналізу відеоматеріалів за рахунок використання нейромереж у автоматизованих системах позиціювання об'єктів.

Основними задачами дослідження є:

- розробка модуля детекції об’єктів, класифікації, трекінгу, аналізу їх швидкості, прискорення та розмірів;

- розробка графічного інтерфейсу для роботи із програмним забезпеченням, переглядом та експортом результатів;
- розробка модуля візуалізації результатів;
- модифікувати метод позиціювання об'єктів у потоці даних для підвищення точності визначення кінематичних параметрів;
- розробка алгоритму збереження\експорту результатів;
- інтеграція модулів та алгоритмів у єдиний застосунок для визначення параметрів на динамічних зображеннях.

Об'єкт дослідження – процес розробки системи автоматизованого визначення параметрів об’єктів у динамічних зображеннях.

Предмет дослідження – методи та засоби розробки системи комп’ютерного зору й машинного навчання, що використовуються для аналізу параметрів об'єктів у відеопотоці, а також засоби інтеграції відповідних алгоритмів у програмне забезпечення.

Методи дослідження. Для реалізації поставлених задач були використані такі методи дослідження: методи комп'ютерного зору для розробки алгоритмів виявлення та відстеження об'єктів у відеопотоці, теорія нейронних мереж та глибокого навчання для реалізації модуля детекції об'єктів, методи обробки зображень для попередньої підготовки кадрів та аналізу візуальних характеристик об'єктів, алгоритми трекінгу для реалізації модуля відстеження об'єктів між кадрами, об'єктно-орієнтоване програмування для розробки програмних модулів системи, методи тестування програмного забезпечення для перевірки працездатності створеного програмного продукту.

Наукова новизна отриманих результатів.

Модифіковано метод позиціювання об'єктів у потоці даних, що на відміну від існуючих, використовує «YOLO» модель нейромереж для підвищення точності визначення кінематичних параметрів на основі трекінгу класифікованих об'єктів.

Практична цінність отриманих результатів полягає у можливості використання програмного забезпечення в системах відеоспостереження, аналізу

руху транспорту, автоматизованому контролі виробничих процесів та медичній діагностиці на основі відеоаналізу.

Особистий внесок здобувача. Усі результати, викладені у бакалаврській кваліфікаційній роботі, отримані автором особисто. У науковій праці[4], опублікованій у співавторстві, автору належать такі результати: аналіз методів визначення параметрів об’єктів на динамічних зображеннях. Розроблено програмні модулі комп'ютерної системи для визначення параметрів об’єктів на динамічних зображеннях.

Апробація результатів роботи. Результати роботи доповідалися на всеукраїнській науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії у 2025 році і опубліковані в матеріалі конференції [4].

Публікації. Результати роботи опубліковані на всеукраїнській науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії у 2025 році і опубліковані в матеріалі конференції.

1 АНАЛІЗ РОЗВИТКУ ТЕХНОЛОГІЙ ОБРОБКИ ДИНАМІЧНИХ ЗОБРАЖЕНЬ І ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану технологій обробки динамічних зображень для визначення параметрів об'єктів

Сучасні технології в галузі комп'ютерного зору та машинного навчання сприяють значному покращенню точності та швидкості аналізу динамічних зображень. Традиційні методи аналізу зображень, хоча й використовуються в багатьох сферах, часто обмежені низькою деталізацією та здатні працювати лише з двовимірними зображеннями. Використання новітніх підходів, таких як глибоке навчання та методи відстеження об'єктів у відеопотоці, дозволяє значно підвищити ефективність і точність аналізу.

Сьогодні з'являються потужні алгоритми для детекції та відстеження об'єктів, які дозволяють здійснювати вимірювання різних параметрів об'єктів, таких як розмір, швидкість, траєкторія руху та інші характеристики. Завдяки цьому, застосування методів комп'ютерного зору стало можливим не лише для статичних зображень, а й для складних динамічних відео.

Однією з ключових переваг цих технологій є можливість аналізувати рух об'єктів у режимі реального часу, що має великий потенціал для застосування в системах відеоспостереження, контролю транспортних потоків, промислових процесах та навіть у медичних діагностичних системах. Завдяки машинному навчанню, алгоритми стають здатними навчатися і адаптуватися до змінних умов, що дозволяє зберігати точність і ефективність навіть за різних умов зйомки та змінюваних характеристик об'єктів.

Іншим важливим аспектом є інтеграція цих технологій у зручні програмні засоби, що забезпечують простий доступ до функціоналу користувачів та надають можливість налаштування параметрів аналізу. Важливими є також можливості експорту даних у популярні формати, що дозволяє використовувати отриману інформацію в подальших етапах обробки або інтеграції з іншими системами.

Враховуючи величезний потенціал для застосування в різноманітних галузях, розробка програмних засобів для обробки динамічних зображень із використанням сучасних методів комп'ютерного зору є надзвичайно актуальним завданням. Це дозволить створити інструменти, які зможуть забезпечити точні, надійні та швидкі вимірювання параметрів об'єктів на динамічних зображеннях.

Завданням даного дослідження є розробка програмного продуку для автоматизованого визначення параметрів об'єктів, що базуються на алгоритмах комп'ютерного зору та методах машинного навчання. Це дозволить створити програмне забезпечення, здатне точно і швидко здійснювати аналіз параметрів об'єктів на відео, що стане корисним у таких сферах, як відеоспостереження, аналіз транспорту, промислові контролі та медична діагностика.

1.2 Порівняльний аналіз аналогів

Сучасний ринок рішень для аналізу динамічних зображень представлений декількома відомими хмарними сервісами, зокрема Amazon Rekognition, Google Cloud Vision API та Microsoft Azure Computer Vision. Ці аналоги спрямовані на автоматизацію розпізнавання об'єктів, забезпечення інтеграції з іншими сервісами та ефективну роботу з великими наборами даних. Кожен із них має свої сильні сторони, зокрема у сфері аналітики, масштабування та простоти інтеграції через API, проте деякі обмежені можливостями локального розгортання, роботи в реальному часі та гнучкістю налаштувань. Вони переважно орієнтовані на типові сценарії використання, де потрібна швидка обробка за шаблоном, але не передбачено розширеного налаштування моделей під специфічні задачі користувача.

Amazon Rekognition – це хмарний сервіс від AWS, який дозволяє аналізувати зображення та відео [5]. Перевагами є тісна інтеграція з AWS екосистемою та наявність зручного API для розробників. Недоліками є відсутність підтримки роботи в реальному часі, неможливість локального розгортання, обмежена

гнучкість налаштувань та відсутність підтримки кастомних моделей. Інтерфейс застосунку з результатом аналізу зображено на рисунку 1.1.

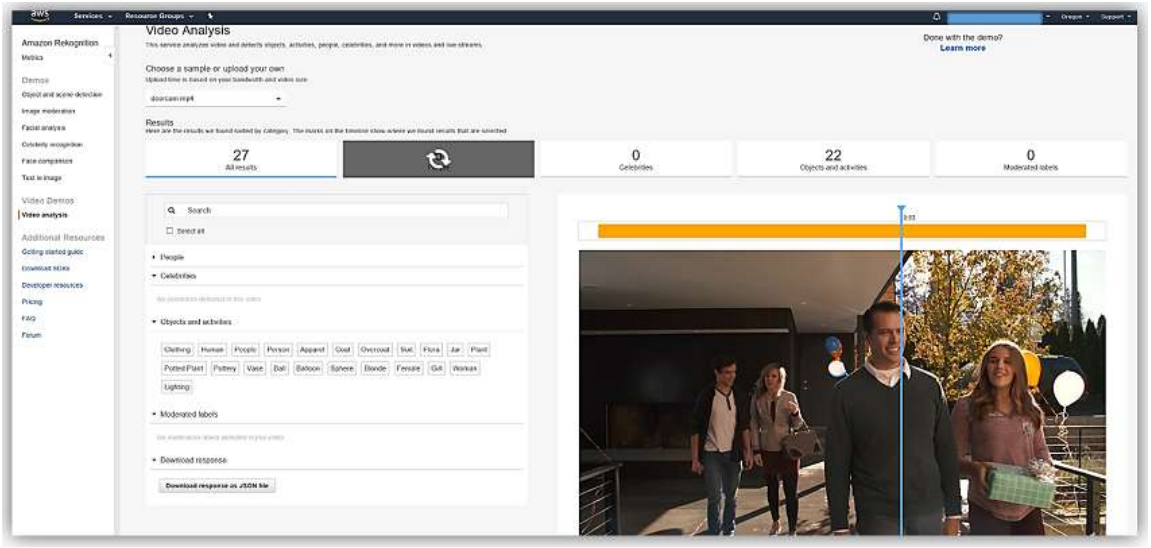


Рисунок 1.1 – Зображення інтерфейсу Amazon Rekognition

Google Cloud Vision API – сервіс від Google, орієнтований на розпізнавання зображень і відео [6]. Перевагами є простота інтеграції через API, підтримка роботи з великими наборами даних. Недоліками є неможливість обробки даних у реальному часі, відсутність локального розгортання, недостатня підтримка кастомних моделей. Інтерфейс зображено на рисунку 1.2.

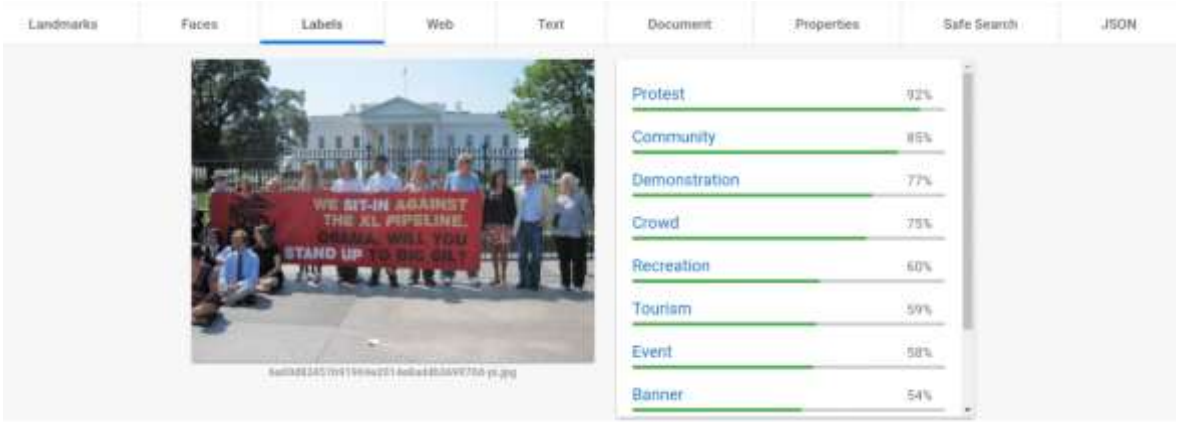


Рисунок 1.2 – Інтерфейс прикладу використання Google Cloud Vision API

Microsoft Azure Computer Vision – це платформа від Microsoft, яка забезпечує аналіз зображень і відео [7]. Перевагами є гнучкі налаштування, підтримка роботи з великими наборами даних та використання кастомних моделей. Недоліками є менш ефективна інтеграція через API та відсутність можливості локального розгортання. Інтерфейс з сторінкою результатів зображено на рисунку 1.3.

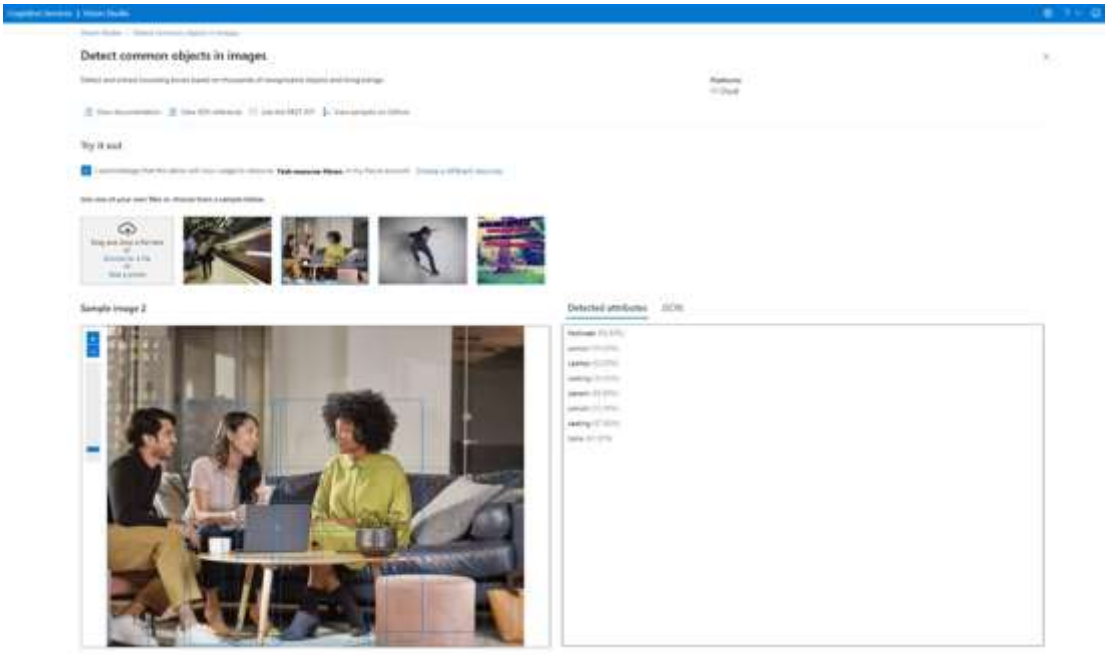


Рисунок 1.3 – Інтерфейс Microsoft Azure Computer Vision

Порівняння критеріїв переваг та недоліків аналогів наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

Критерій	Amazon Rekognition	Google Cloud Vision API	Microsoft Azure Computer Vision	Власна розробка
Робота в реальному часі	0	0	0	1
Можливість локального розгортання	0	0	0	1
Гнучкість налаштувань	0	0	1	1

Продовження таблиці 1.1

Критерій	Amazon Rekognition	Google Cloud Vision API	Microsoft Azure Computer Vision	Власна розробка
Підтримка кастомних моделей	0	0	1	1
Простота інтеграції через API	1	1	0	1
Підтримка роботи з великими наборами даних	1	0	1	1
Можливість масштабування	0	1	1	1
Сумарний коефіцієнт	2	2	4	7

Отже, виходячи з переваг та недоліків аналогічних застосунків, розроблений застосунок повинен підтримувати обробку динамічних зображень у реальному часі та забезпечувати можливість локального розгортання для кращого контролю за даними. Він має бути високогнучким щодо налаштувань, дозволяти використання кастомних моделей, забезпечувати просту інтеграцію через API, ефективно працювати з великими наборами даних та підтримувати масштабованість системи.

1.3 Аналіз методів розв’язання задачі

Для вирішення задачі детекції та трекінгу об’єктів було розглянуто декілька підходів, що базуються на сучасних алгоритмах комп’ютерного зору та глибокого навчання. Аналіз існуючих методів дозволяє визначити їх сильні та слабкі сторони, а також обрати найбільш ефективний підхід для реалізації програмного модуля.

Один із традиційних методів — використання бібліотеки OpenCV [8,9]. Вона забезпечує швидку обробку зображень та відео, має велику кількість вбудованих алгоритмів та добре підходить для базових завдань комп’ютерного зору. Проте, її основний недолік — обмежена точність у складних умовах, зокрема при низькому освітленні чи часткових перекриттях об’єктів.

Інший підхід — застосування TensorFlow Object Detection API, який використовує глибокі нейронні мережі для детекції об’єктів [10,11]. Він забезпечує

високу точність та можливість навчання на кастомних датасетах. Водночас, цей метод вимагає значних обчислювальних ресурсів і має меншу швидкість роботи в реальному часі, що може бути критичним фактором у певних сценаріях.

Алгоритм ByteTrack дозволяє здійснювати трекінг об’єктів у відеопотоці, поєднуючи детекцію з використанням глибоких нейронних мереж для асоціації треків [12]. Цей підхід забезпечує стійкість до шумів та ефективно працює у випадках, коли об’єкти тимчасово зникають із кадру. Проте, його продуктивність значною мірою залежить від точності детектора, що використовується разом із ним.

Також було розглянуто MediaPipe, який пропонує високу точність і швидкість для задач розпізнавання поз, трекінгу рук та облич [13]. Він добре працює на мобільних пристроях завдяки оптимізованим моделям, однак має обмеження у можливостях кастомного навчання та може поступатися у гнучкості налаштувань.

Найефективнішим рішенням для поставленої задачі є використання YOLO (You Only Look Once) [14]**.** Даний алгоритм відрізняється балансом між високою швидкістю обробки та високою точністю детекції. YOLO працює в реальному часі, що є ключовою перевагою у порівнянні з іншими методами. Крім того, алгоритм добре адаптується до різних умов, забезпечує стійкість до шумів і має відносно простий процес налаштування.

На основі порівняльного аналізу методів було прийнято рішення використовувати YOLO як основний алгоритм для реалізації програмного модуля. Це забезпечить оптимальне співвідношення продуктивності, точності та гнучкості в рамках поставленої задачі.

1.4 Постановка задач дослідження

На основі аналізу існуючих методів та програмних засобів для детекції та трекінгу об’єктів було визначено перелік завдань, які необхідно реалізувати для успішної розробки програмного забезпечення з використанням YOLO.

Основними завданнями які необхідно розв’язати у процесі розробки програм є:

- розробка алгоритму детекції об’єктів за допомогою YOLO, що забезпечить високу швидкість та точність розпізнавання;
- інтеграція алгоритму з модулем трекінгу, що дозволить відстежувати переміщення об’єктів у реальному часі;
- налаштування програмного модуля для роботи з потоковими вхідними даними, що забезпечить його ефективне застосування в реальних умовах;
- оптимізація продуктивності алгоритму, враховуючи апаратні можливості та вимоги до швидкості обробки;
- розробка зручного інтерфейсу користувача, який дозволить налаштовувати параметри детекції та переглядати результати роботи моделі;
- проведення тестування розробленого програмного модуля, аналіз його продуктивності та відповідності поставленим вимогам.

1.5 Висновки

У першому розділі було проведено глибокий аналіз існуючих методів і програмних засобів для детекції та трекінгу об'єктів. Розглянуто сучасні хмарні сервіси – Amazon Rekognition, Google Cloud Vision API та Microsoft Azure Computer Vision.

Порівняльний аналіз функціональних можливостей аналогів свідчить про їх обмеження: відсутність підтримки обробки даних у реальному часі обмеженість розгортання на локальних серверах, що обмежує контроль та безпеку даних. Крім того, ці сервіси мають меншу гнучкість налаштувань і недостатню підтримку кастомних моделей. З іншого боку, хмарні рішення переважно вирізняються простотою інтеграції через API, підтримкою роботи з великими наборами даних та можливістю масштабування.

На основі проведеного аналізу аналогічних продуктів, власний застосунок має підтримувати обробку динамічних зображень у реальному часі, можливість

локального розгортання для підвищення контролю за даними, високу гнучкість налаштувань, підтримку кастомних моделей, просту інтеграцію через API, а також ефективну роботу з великими наборами даних та масштабованість.

Визначено, що для аналізу та детекції зображень найбільш ефективним підходом є використання алгоритму YOLO, що забезпечує оптимальне співвідношення швидкості, точності та гнучкості.

Таким чином, доцільність розробки власного програмного забезпечення для детекції та трекінгу об'єктів підтверджується як з точки зору методичних переваг використання алгоритму YOLO, так і завдяки комплексному функціоналу, що перевищує можливості сучасних хмарних сервісів.

2 РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ

2.1 Аналіз вхідних даних системи

Система обробки відеоданих призначена для виявлення, класифікації та відстеження об'єктів у відеоматеріалах різного походження. Для забезпечення гнучкості та універсальності система підтримує два основних типи вхідних даних:

Відеофайли представляють собою передзаписаний контент, який користувач завантажує через веб-інтерфейс. Система підтримує різноманітні формати відеофайлів, включаючи MP4, AVI, MOV, що дозволяє працювати з матеріалами з різних джерел без необхідності попередньої конвертації. Кожен завантажений файл проходить первинну валідацію для перевірки цілісності та відповідності підтримуваним форматам. Відеофайли характеризуються фіксованою тривалістю, що дозволяє системі заздалегідь планувати ресурси для обробки та відображати користувачу очікуваний час завершення.

Відеопотоки є джерелом даних у реальному часі, що надходять з IP-камер, систем відеоспостереження або інших потокових джерел. Система підтримує найпоширеніші протоколи потокового відео, такі як RTSP (Real Time Streaming Protocol), HTTP та HTTPS. Особливістю роботи з потоками є необхідність безперервної обробки даних, що надходять, та ефективного управління ресурсами для запобігання перевантаженню системи. На відміну від відеофайлів, потоки не мають визначеної кінцевої точки, тому система повинна підтримувати механізми тривалої роботи з можливістю періодичного збереження результатів та очищення тимчасових даних.

Для обох типів вхідних даних система виконує аналіз метаданих, що включає:

1. Роздільну здатність відео – визначає кількість пікселів у кадрі, що впливає на детальність аналізу та необхідні обчислювальні ресурси. Система підтримує широкий діапазон роздільних здатностей, від низьких (320x240) до

високих (1920x1080 і вище), автоматично адаптуючи параметри обробки для оптимального балансу між точністю та швидкістю.

2. Частоту кадрів (FPS) - характеризує кількість кадрів на секунду, що впливає на плавність відображення руху та точність відстеження об'єктів. Стандартні значення варіюються від 24 до 30 кадрів на секунду для звичайного відео та до 60 кадрів для високоякісних джерел. Для потокового відео з високою частотою кадрів система може застосовувати вибіркову обробку для економії ресурсів.

3. Тривалість відео - доступна для відеофайлів і дозволяє оцінити загальний обсяг даних, що підлягають обробці. Вимірюється в секундах або у форматі години:хвилини:секунди і використовується для планування процесу обробки та відображення прогресу користувачу.

4. Формат кодування - визначає алгоритм стиснення відео (H.264, H.265, VP9 тощо), що впливає на якість зображення та ефективність декодування. Система автоматично визначає формат кодування та застосовує відповідні бібліотеки для декодування з максимальною ефективністю.

Окрім аналізу самих відеоданих, система потребує від користувача ряд параметрів для обробки:

1. Поріг впевненості детекції - числове значення в діапазоні від 0 до 1, що визначає мінімальний рівень достовірності для прийняття виявлення об'єкта. Чим вищий поріг, тим менше хибних виявлень, але водночас зростає ризик пропуску реальних об'єктів. Значення за замовчуванням встановлено на 0.5.

2. Класи об'єктів для виявлення – перелік категорій об'єктів, які система має відстежувати.

3. Частота обробки кадрів – параметр для потокового відео, який визначає, скільки кадрів на секунду буде аналізуватися. Для відеопотоків з високою частотою кадрів (наприклад, 30 FPS) можна встановити нижчу частоту обробки (наприклад, 5 FPS), що дозволить значно знизити навантаження на систему.

2.2 Розробка моделі системи

Система побудована за модульним принципом, де кожен компонент відповідає за окрему функціональну область і може розроблятися, тестуватися та оновлюватися незалежно від інших компонентів. Для кращого розуміння роботи модулів, розроблено UML діаграму діяльності (рис. 2.4) у середовищі сервісу diagrams.net.

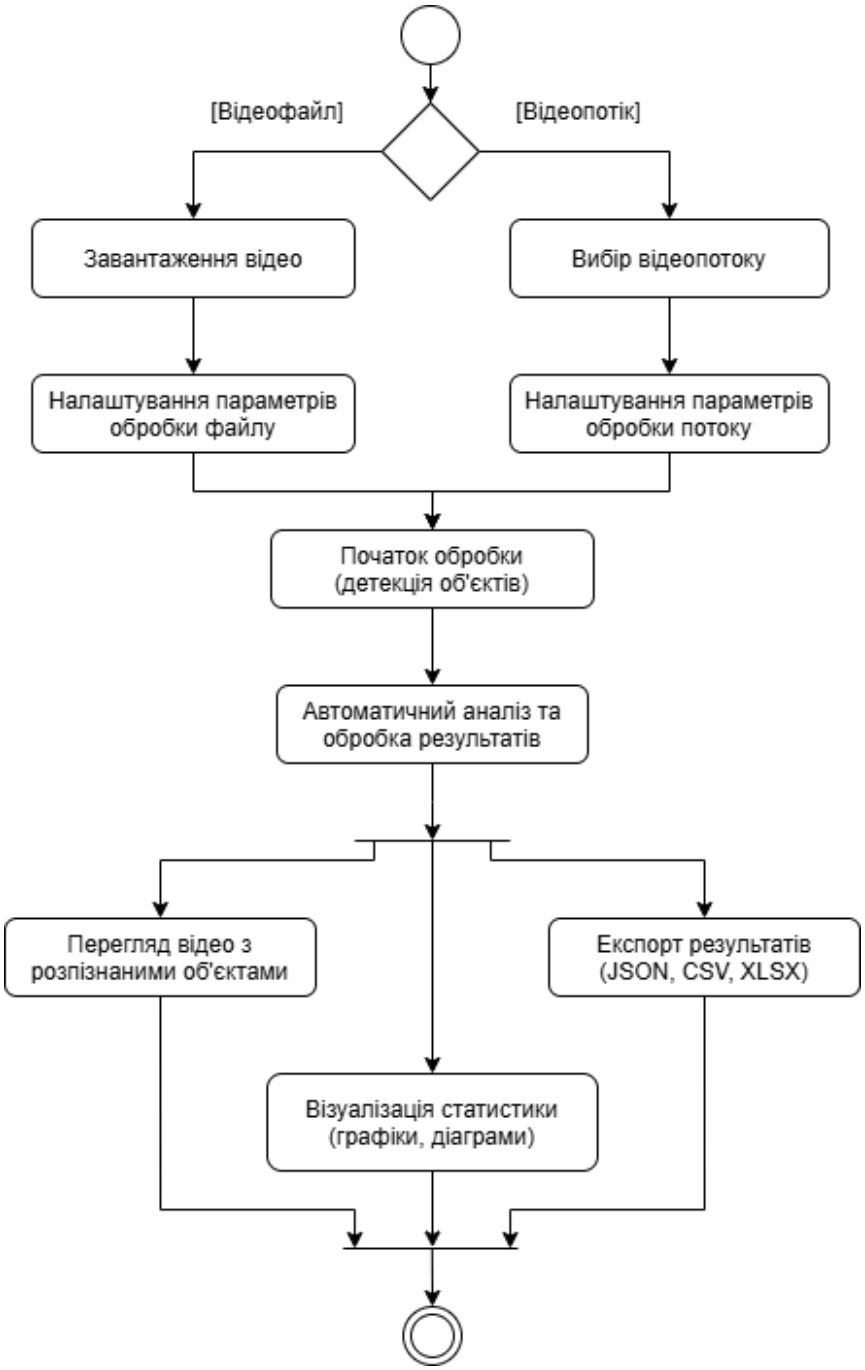


Рисунок 2.1 – Діаграма діяльності застосунку

Згідно з діаграмою, користувач має два основні шляхи взаємодії з системою: обробка відеофайлу або відеопотоку. При виборі відеофайлу користувач спочатку завантажує відео у систему, а потім налаштовує параметри обробки файлу. Якщо обирається відеопотік, користувач вводить адресу потоку та налаштовує параметри його обробки.

Після налаштування параметрів система розпочинає обробку – виконується детекція об'єктів на відео з використанням обраних налаштувань. Далі відбувається автоматичний аналіз та обробка результатів, після чого користувач отримує доступ до трьох основних функцій: перегляд відео з розпізнаними об'єктами, експорт результатів у різних форматах (JSON, CSV, XLSX) та візуалізація статистики у вигляді графіків і діаграм.

Серверна частина представлена набором взаємопов'язаних сервісів, кожен з яких виконує спеціалізовані функції:

1. Сервіс обробки відео (ProcessingService) – відповідає за аналіз відео та виявлення об'єктів. Реалізує завантаження й валідацію відео, декодування кадрів, детекцію об'єктів, їх трекінг, асинхронну обробку завдань, генерацію метаданих і обробку помилок.

2. Сервіс зберігання даних (StorageService) – забезпечує збереження відеофайлів, кадрів з результатами, ескізів і експортованих даних з відповідним структуруванням.

3. Сервіс управління завданнями (JobService) – координує створення, виконання та моніторинг завдань. Відстежує статус, керує чергами, взаємодіє з іншими сервісами та зберігає статистику.

4. Сервіс відеопотоків (StreamService) – обробляє відео з різних потокових джерел. Включає валідацію, буферизацію, моніторинг з'єднань і автоматичне перепідключення. Адаптується до нестабільних потоків.

5. Сервіс результатів (ResultService) – збирає, обробляє й зберігає результати аналізу. Підтримує агрегацію, генерацію статистики, візуалізацій, експорт у різних форматах та фільтрацію.

Для зберігання даних система використовує реляційну базу даних, в якій розміщуються:

1. Метадані завдань обробки - інформація про завдання, їх параметри, статуси та часові показники.
2. Результати детекції об'єктів - структуровані дані про виявлені об'єкти з їх характеристиками.
3. Налаштування системи - конфігураційні параметри, що впливають на роботу різних компонентів.

Взаємодія між компонентами системи організована за принципом слабкого зв'язування, що забезпечує гнучкість і спрощує тестування:

1. Клієнт-серверна взаємодія – веб-інтерфейс взаємодіє з серверною частиною через API, що забезпечує чітке розділення відповідальності та дозволяє інтегрувати сервіс для роботи з іншими системами.
2. Міжсервісна взаємодія – сервіси взаємодіють через визначені інтерфейси, передаючи дані в стандартизованих форматах.
3. Доступ до даних – компоненти системи отримують доступ до даних через абстракцію (репозиторій), що забезпечує незалежність від конкретної реалізації бази даних.
4. Управління залежностями – необхідні сервіси передаються в компоненти через ін'єкцію залежностей.

Така організація забезпечує модульність, масштабованість та гнучкість системи, дозволяючи легко тестувати, розширювати функціональність, замінювати окремі компоненти та адаптувати систему до різних сценаріїв використання.

2.3 Розробка алгоритму динамічного аналізу та виявлення об’єктів

Для забезпечення ефективної обробки відеоданих та отримання точних результатів виявлення об'єктів було розроблено два ключові алгоритми, які визначають функціонування системи: алгоритм динамічного аналізу виявлених

об'єктів у відеопотоці та алгоритм адаптивного експорту результатів у форматах типу: JSON, CSV, XLSX.

Розроблений алгоритм динамічного аналізу виявлених об'єктів (рис. 2.5) забезпечує послідовне опрацювання кадрів відеопотоку, виявлення об'єктів та їх відстеження. Цей алгоритм адаптований для роботи як з завантаженими відеофайлами, так і з потоковим відео в реальному часі.

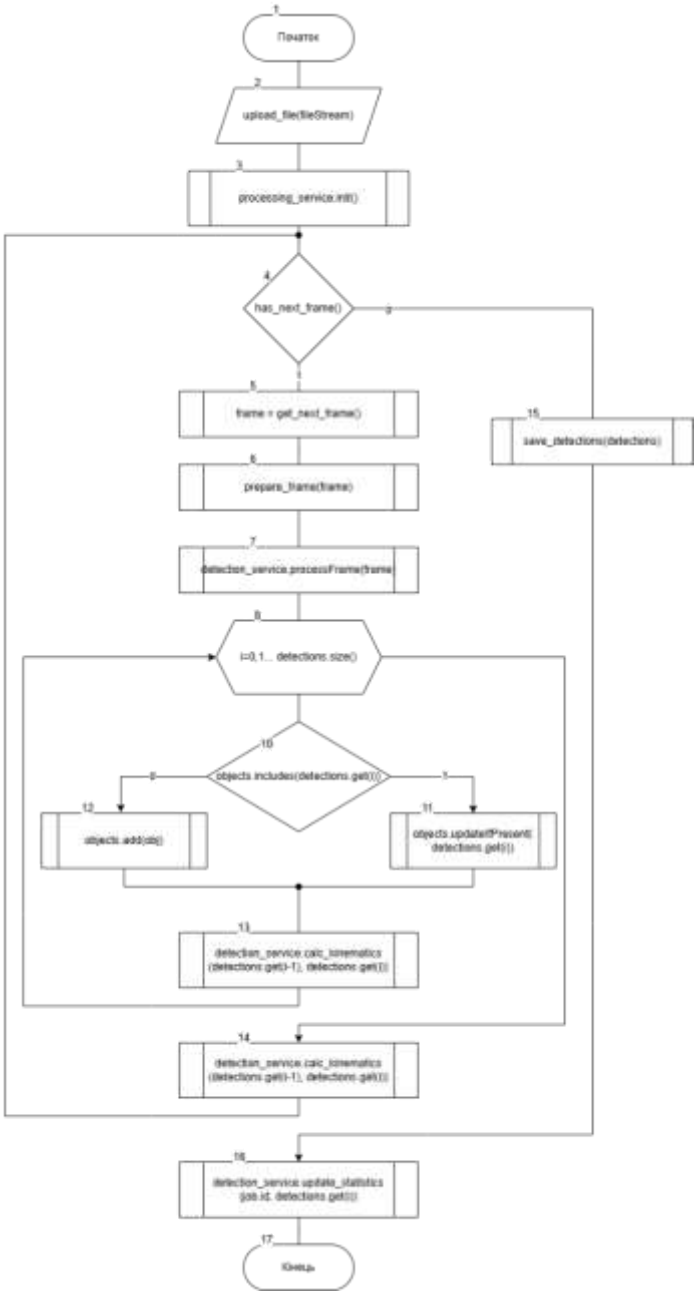


Рисунок 2.2 – Блок схема алгоритму аналізу виявлених об'єктів

Згідно з даним алгоритмом, процес починається із завантаження необхідних параметрів моделі детекції. Ця фаза включає ініціалізацію моделі нейронної мережі, налаштування порогів впевненості та інших параметрів, що впливають на точність розпізнавання.

Наступним етапом є ініціалізація системи відстеження (трекінгу) об'єктів, яка дозволяє ідентифікувати та слідкувати за об'єктами протягом послідовних кадрів. Ця система використовує алгоритми, що дозволяють зберігати унікальні ідентифікатори об'єктів навіть при їх частковому перекритті або короткочасному зникненні з кадру.

Після завершення підготовчих етапів розпочинається основний цикл обробки відеопотоку, який починається з отримання поточного кадру. Для кожного кадру виконується перевірка наявності наступного кадру. Якщо наступного кадру немає, система переходить до збереження кінцевої статистики та завершення роботи. Якщо наступний кадр існує, система продовжує аналіз.

Обробка кожного кадру починається з препроцесингу, який включає масштабування кадру до оптимального розміру для моделі детекції та нормалізацію зображення. Після цього виконується детекція об'єктів на кадрі, яка повертає список виявлених об'єктів з інформацією про їхнє положення (координати обмежувальних рамок), клас та рівень впевненості моделі.

Отримані детекції проходять фільтрацію за порогом впевненості, де відкидаються результати з низьким рівнем достовірності. Це дозволяє зменшити кількість хибних спрацьовувань.

Для кожного виявленого об'єкта після фільтрації система визначає, чи є об'єкт новим. Якщо об'єкт новий, для нього створюється унікальний ідентифікатор та ініціалізуються параметри відстеження. Якщо об'єкт уже був виявлений раніше, система оновлює його параметри, такі як положення та розмір.

Після визначення статусу об'єкта система виконує його класифікацію та обрахування кінетичних параметрів, використовуючи результати моделі розпізнавання. Включно з обрахуванням кінематичних параметрів, обчислюється

траєкторія руху об'єкта на основі його попередніх та поточних координат, що дозволяє уточнювати параметри типу відносної швидкості та прискорення.

Після обробки всіх виявлених об'єктів на поточному кадрі система оновлює статистику виявлених об'єктів, включаючи підрахунок кількості об'єктів різних класів, тривалість їх перебування в кадрі, швидкість, прискорення тощо.

Після обробки всіх кадрів алгоритм зберігає підсумкові дані та позначає виконання задачі як завершене.

Розроблений алгоритм забезпечує високу точність виявлення та відстеження об'єктів у відеопотоці завдяки поєднанню сучасних методів детекції на основі нейронних мереж та ефективних методів трекінгу. Використання фільтрації за порогом впевненості та розрахунок траєкторій руху дозволяє зменшити кількість хибних спрацьовувань та забезпечити стабільне відстеження об'єктів навіть у складних умовах.

2.4 Розробка алгоритму експорту результатів

Другий ключовий алгоритм системи — алгоритм адаптивного експорту результатів (рис. 2.3), який забезпечує гнучке збереження та зорбаження даних аналізу в різних форматах відповідно до потреб користувача.

Згідно з цим алгоритмом, процес починається із завантаження даних обробленого відео, які містять інформацію про виявлені об'єкти, їхні класи та інші параметри. Наступним кроком є отримання потрібного файлу для експорту даних із запиту.

Після цього відбувається визначення формату експорту відповідно до вибору користувача або налаштувань системи. Система підтримує експорт у кілька популярних форматів, таких як JSON, CSV, XLSX кожен з яких має свої переваги та варіанти використання.

Для кожного виявленого об'єкта система виконує підготовку даних у відповідному форматі. Залежно від обраного формату експорту дані форматуються відповідним чином:

- Для JSON створюється ієрархічна структура даних з можливістю включення складних вкладених об'єктів та масивів.
- Для CSV дані організовуються у вигляді плоскої таблиці з рядками та стовпцями, де кожен рядок відповідає одній детекції.
- Для XLSX підготовка включає створення структури електронної таблиці з можливістю форматування та організації даних на кількох аркушах.

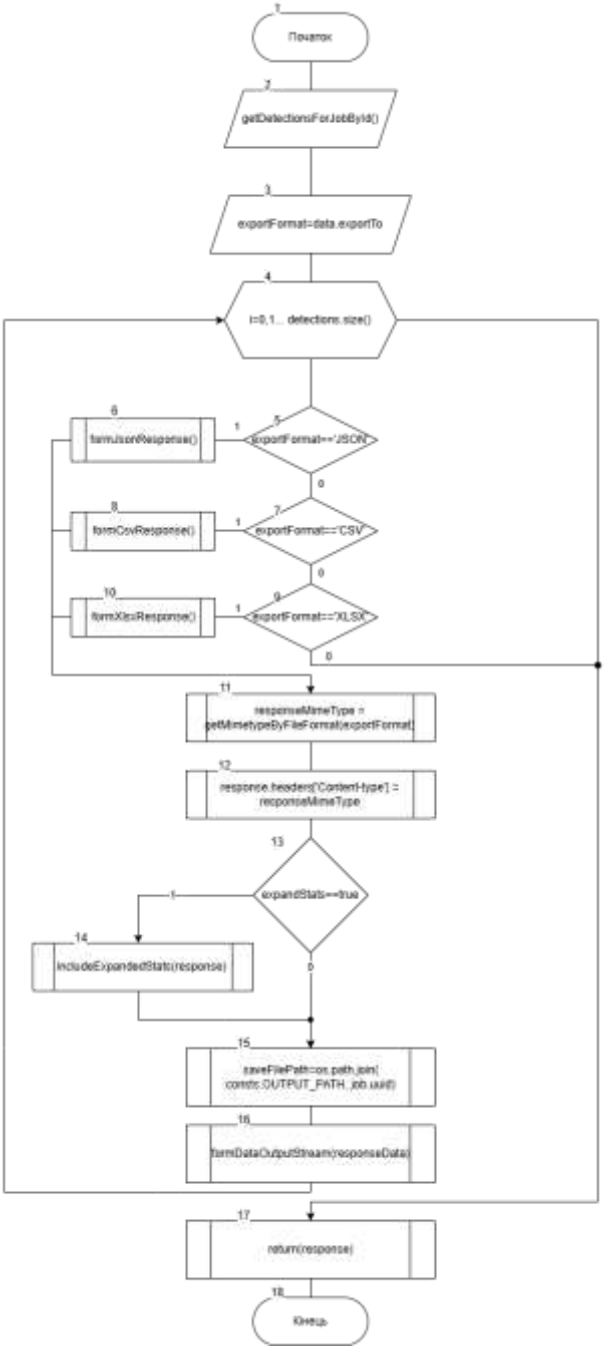


Рисунок 2.3 – Блок схема експорту результатів

Після визначення формату система переходить до налаштування метаданих файлу, які містять загальну інформацію про відео, параметри аналізу та статистичні дані.

Наступний етап передбачає налаштування рівня деталізації даних. Користувач може обрати між включенням розширеної статистики, яка містить детальну інформацію про кожен кадр, або обмеженням до агрегованої статистики, яка надає узагальнені дані про виявлені об'єкти.

Після підготовки всіх необхідних даних система формує шлях для збереження файлу, виконує запис даних у файл обраного формату та перевіряє. Після цього виконується повернення результатів з формованих типів та отриманих даних по детекціях.

Розроблений алгоритм адаптивного експорту забезпечує гнучкість у подані та збереженні результатів аналізу, дозволяючи користувачам отримувати дані у форматі, найбільш підходящому для їхніх потреб. Підтримка різних форматів експорту та можливість налаштування рівня деталізації даних робить систему універсальним інструментом для роботи з результатами відеоаналітики в різних сценаріях використання.

Реалізація цих двох алгоритмів у системі забезпечує ефективне виявлення, відстеження та аналіз об'єктів у відеопотоці, а також зручне збереження та візуалізацію результатів для подальшого аналізу та використання.

2.5 Висновки

В результаті проведеної роботи з розробки структури та алгоритмів програмного продукту для детектування та відстеження об'єктів у відеопотоці було створено комплексну систему, що відповідає сучасним вимогам до функціональності, продуктивності та зручності використання.

3. РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ ВИЗНАЧЕННЯ ПАРАМЕТРІВ ОБ'ЄКТІВ НА ДИНАМІЧНИХ ЗОБРАЖЕННЯХ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

При розробці програмного забезпечення для визначення параметрів об'єктів на динамічних зображеннях важливо обрати оптимальні технології для розробки кожного модуля програмних засобів. Python є потужним інструментом для розробки систем комп'ютерного зору. Використання Python забезпечує можливість створення гнучкого програмного рішення для обробки відеопотоків та аналізу параметрів об'єктів з високою точністю.

OpenCV (Open Source Computer Vision Library) – це відкрита бібліотека комп'ютерного зору, яка надає широкий спектр функцій для обробки зображень та відео [17]. Вона була розроблена компанією Intel та є однією з найпопулярніших бібліотек для завдань комп'ютерного зору. OpenCV дозволяє розробникам виконувати різноманітні операції з зображеннями, такі як фільтрація, трансформація, аналіз руху, детекція об'єктів та багато іншого, без необхідності написання складних алгоритмів з нуля.

NumPy – це фундаментальна бібліотека для наукових обчислень у Python [18]. Вона була створена для забезпечення ефективної роботи з багатовимірними масивами та матрицями. NumPy надає потужні інструменти для маніпуляції даними, математичних операцій та статистичного аналізу. В контексті обробки зображень, NumPy є невід'ємною частиною, оскільки дозволяє ефективно представляти зображення як масиви та виконувати над ними різноманітні операції з високою продуктивністю.

YOLOv8 (You Only Look Once) – це алгоритм для детекції об'єктів на зображеннях в режимі реального часу. Він був розроблений компанією Ultralytics з метою виявлення об'єктів різних класів з високою точністю та швидкодією. YOLOv8 надає широкий спектр функцій, включаючи точне визначення об'єктів на

кадрах відеопотоку за один прохід нейронної мережі, що робить його ідеальним для застосувань, які вимагають обробки в режимі реального часу. Датасет COCO (Common Objects in Context) забезпечує попередньо навчені моделі для розпізнавання 80 різних класів об'єктів, що дозволяє швидко інтегрувати алгоритм детекції у різноманітні системи.

ByteTrack – це алгоритм для відстеження множинних об'єктів у відеопотоці. Він працює за рахунок поєднання результатів детекції високої впевненості для кращого відслідковування результатів з проміжними результатами із низькою впевнінестю. ByteTrack дозволяє надійно відстежувати об'єкти навіть у складних сценаріях, коли вони тимчасово зникають з поля зору або перекриваються іншими об'єктами. Завдяки використанню візуальних ознак алгоритм здатний підтримувати стабільну ідентифікацію об'єктів протягом тривалого часу, що є критично важливим для аналізу їх параметрів.

Поєднання цих технологій дозволяє розробити програмні засоби для визначення параметрів об'єктів на динамічних зображеннях. Python та бібліотеки OpenCV і NumPy забезпечують ефективну кодову базу для обробки відеопотоків та аналізу зображень, в той час як YOLOv8 надає потужні інструменти для детекції об'єктів. Імплементація алгоритму ByteTrack забезпечує стабільне відстеження об'єктів між кадрами, що необхідне для точного визначення їх параметрів у часі.

Отже, вибір технологій для розробки програмних засобів визначення параметрів об'єктів на динамічних зображеннях, що використовують Python, OpenCV, NumPy, YOLOv8 та ByteTrack, має вирішальне значення для створення ефективного і точного інструменту аналізу відеоданих. Поєднання цих інструментів надає комплексний підхід до розробки, що дозволяє створювати продукт з високим рівнем функціональності, надійності та продуктивності.

3.2 Розробка інтерфейсу

Інтерфейс користувача є ключовим компонентом системи, що забезпечує доступ до всіх функціональних можливостей та візуалізації результатів аналізу у

зручній для сприйняття формі. При розробці інтерфейсу особлива увага приділялася інтуїтивності використання, інформативності та адаптивності до різних пристроїв.

Програмний додаток реалізований як веб-застосунок з використанням сучасного стеку технологій: на серверній стороні працює фреймворк Flask (Python), а для клієнтської частини застосований HTML5 та CSS-фреймворк Bulma, що забезпечує сучасний, респонсивний дизайн та широкі можливості кастомізації [15,16]. Для інтерактивних елементів та асинхронної взаємодії з сервером використовується JavaScript.

Головна сторінка (рис. 3.1) служить точкою входу в систему та фокусується на простоті та доступності основних функцій.

Рисунок 3.1 – Головна сторінка

Центральним елементом сторінки є форма для вибору та завантаження джерела відео, яка складається з наступних компонентів:

1. Перемикач типу джерела – реалізований як група радіокнопок з двома варіантами: "Відеофайл" та "Відеопотік". При зміні вибору динамічно оновлюється інтерфейс, відображаючи відповідні поля введення та налаштування.

2. Секція завантаження файлу – відображається при виборі типу "Відеофайл" і містить елемент вибору файлу з підтримкою drag-and-drop та

попереднього перегляду назви і розміру вибраного файлу. Інтерфейс також відображає підтримувані формати файлів для зручності користувача.

3. Секція налаштування потоку – відображається при виборі типу "Відеопотік" і містить поле для введення URL потоку з підказками щодо підтримуваних протоколів (RTSP, HTTP, HTTPS) та додатковими налаштуваннями, такими як частота кадрів для обробки та режим буферизації.

4. Додаткові налаштування – включають поле для назви завдання (за замовчуванням використовується ім'я файлу або URL) та можливі опції попередньої обробки відео.

5. Кнопка запуску обробки – великий акцентний елемент, що запускає процес обробки з вибраними параметрами. Текст кнопки динамічно змінюється залежно від вибраного типу джерела ("Обробити" для файлів, "Запустити потік" для потоків).

Додатково на головній сторінці розміщені інформаційні блоки, що роз'яснюють можливості системи та особливості роботи з різними типами джерел, а також навігаційне меню для переходу до інших розділів системи.

Сторінка результатів (рисунок 3.2) є найскладнішим та інформаційно насиченим компонентом інтерфейсу, оскільки вона відображає результати обробки відео та надає інструменти для їх аналізу.

Рисунок 3.2 – Сторінка результатів

Структурно сторінка поділена на дві основні частини:

1. Бічна панель – розташована зліва і містить список усіх завдань з обробки з індикаторами статусу:

- "В черзі" (жовтий) - завдання очікує обробки;
- "Обробка" (синій) - завдання обробляється в даний момент.
- "Завершено" (зелений) - обробка успішно завершена з можливістю перегляду результатів;
- "Помилка" (червоний) - виникла помилка під час обробки.

Для кожного завдання відображається іконка типу джерела (файл або потік), назва та статус. Активне завдання виділяється кольором, і його деталі відображаються в основній частині сторінки.

2. Тіло – займає більшу частину екрану і містить блок короткої інформації про дату початку та завершення обробки, результат обробки та можливістю видалити. Крім цього детальну інформацію про вибране завдання, організовано у вигляді вкладок:

відео — вкладка з відеоплеєром, на якому відображаються обмежувальні рамки, мітки класів і ідентифікатори виявлених об'єктів. Для потокових джерел доступна функція перегляду в реальному часі з можливістю зупинки потоку та збереження поточного кадру.

аналіз — вкладка з інформацією про відео (назва, тривалість, розмір, дата, кількість об'єктів), параметри детекції (модель, поріг, класи, швидкість), деталізованим списком об'єктів і, для потоків, блоком стану з можливістю зупинки/оновлення. Також включає розширену статистику точності виявлення та порівняльний аналіз ефективності різних класів об'єктів.

графіки — вкладка з інтерактивними діаграмами: кругова (розподіл класів), лінійна (динаміка детекцій), стовпчаста (середня впевненість), ще одна лінійна (поява об'єктів у часі). Всі графіки підтримують масштабування та фільтрацію за часовими інтервалами для детального аналізу тенденцій виявлення об'єктів.

експорт — вкладка для вибору й збереження результатів аналізу у форматах CSV, JSON, XLSX з попереднім переглядом вмісту.

Сторінка налаштувань дозволяє користувачам конфігурувати різні аспекти системи відповідно до своїх потреб. Інтерфейс організований у вигляді вкладок для групування споріднених налаштувань:

Налаштування детекції - керують процесом виявлення об'єктів:

- вибір моделі YOLO (YOLOv5, YOLOv8 за замовчуванням);
- повзунок для встановлення порогу впевненості з візуальним індикатором вибраного значення;
- чекбокси для вибору класів об'єктів для виявлення (люди, автомобілі, велосипеди, мотоцикли та інші).

Налаштування відеопотоку - специфічні параметри для роботи з потоковими джерелами з акцентом на стабільність з'єднання:

- частота кадрів обробки (від 1 до 30 FPS) з автоматичним регулюванням залежно від потужності системи;
- режим буферизації (реальний час або з буферизацією) для оптимізації затримки та якості;
- максимальна тривалість сеансу запису з автоматичним розділенням на сегменти;
- опції збереження знімків виявлених об'єктів з налаштуванням якості та формату;
- налаштування автоматичного перезавантаження при втраті з'єднання з гнучкими інтервалами повторних спроб.

Налаштування експорту – визначають параметри експорту результатів з підтримкою різних сценаріїв використання:

- формат експорту за замовчуванням (JSON, CSV, XLSX) з можливістю створення користувацьких шаблонів;
- чекбокси для вибору даних, що включаються в експорт, з попереднім переглядом структури файлу.

Додатково на сторінці налаштувань зображена інформація про активні відеопотоки (якщо такі є) з можливістю керування ними в реальному часі, включаючи моніторинг стану з'єднання та якості потоку. Також присутня секція налаштувань з'єднання для потокових джерел, що включає параметри тайм-аутів, спроб перепідключення та аутентифікації з підтримкою різних протоколів безпеки.

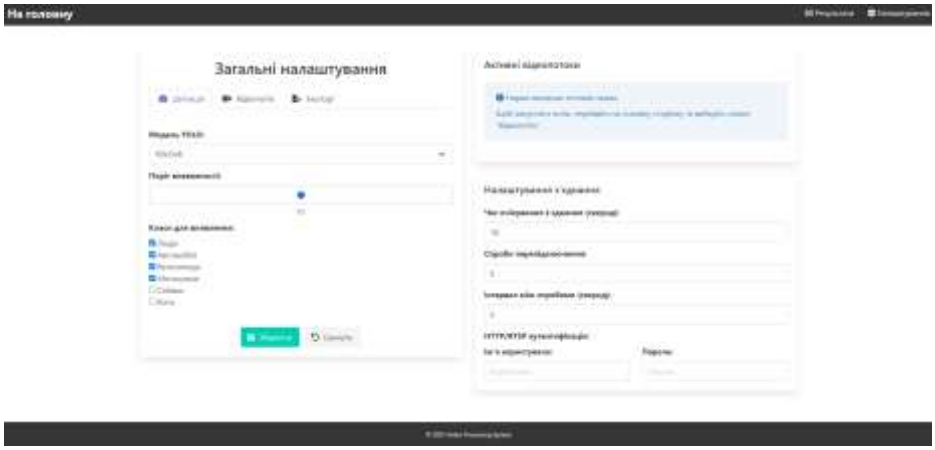


Рисунок 3.3 – Сторінка налаштувань

Для забезпечення єдиного користувацького досвіду всі сторінки системи мають спільні елементи:

1. Навігаційна панель — ключовий елемент інтерфейсу, який розташований зверху. Забезпечує швидкий досутп до всіх основних розділів системи. Містить логотип, який функціонує як посилання на головну сторінку, посилання на результати й сторінку з налаштуваннями. На мобільних девайсах панель згортається в бургер-меню (рис. 3.4).

2. Підвал — містить загальну інформацію про систему та права розробників.

3. Сповіщення — тимчасові повідомлення про успішні дії чи помилки, з’являються зверху у правому куті й зникають автоматично. Статус повідомлення позначений 3 кольорами:

- зелений – операція виконана успішно;

- жовтий – виникла не критична проблема з виконанням операції, тому користувачу відображається попередження з описом інциденту;
- червоний – виникла критична помилка, яка не дала змоги якісно виконати дію.

4. Індикатори завантаження — показують процес довгих операцій у вигляді анімацій..

5. Адаптивний дизайн — інтерфейс підлаштовується під різні пристрої для зручного користування.

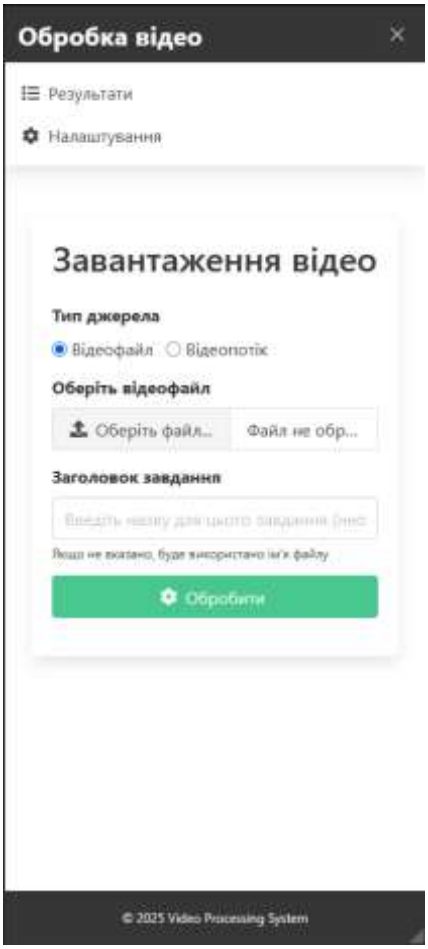


Рисунок 3.4 – Розгорнуте бургерне меню навігації

Такий підхід до інтерфейсу забезпечує уніфіковане відображення елементів на всіх сторінках, підвищує зручність користування та його розуміння.

3.3 Розробка модуля детекції та трекінгу об'єктів на динамічних зображеннях

Модуль детекції і трекінгу (рис. 3.5) є ключовим компонентом системи: він формує цілісне уявлення про динаміку сцени та безпосередньо впливає на якість подальшого аналізу поведінки об'єктів.

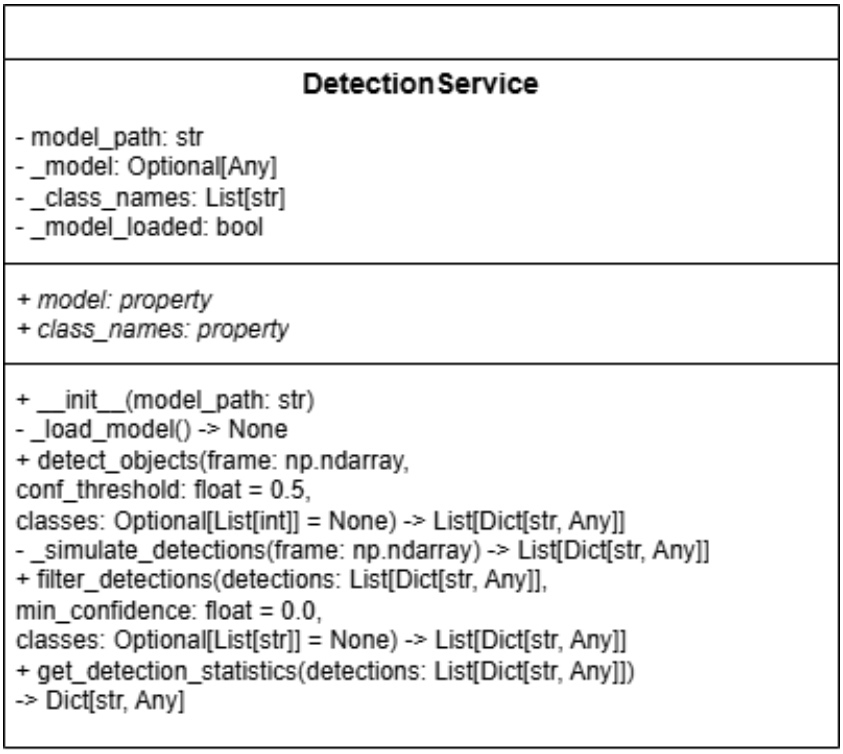


Рисунок 3.5 – Клас сервісу для модуля детекції

Розроблений модуль має модульну архітектуру, що забезпечує гнучкість та розширюваність. Основні компоненти включають: менеджер відеопотоку, який відповідає за отримання та попередню обробку кадрів з різних джерел; препроцесор зображень для підготовки кадрів до аналізу; модуль детекції об'єктів, що реалізує алгоритми виявлення на основі YOLOv8; координатор детекції-трекінгу для управління взаємодією; та аналізатор траєкторій для аналізу рухів об'єктів. Взаємодія між компонентами побудована за принципом "конвеєра обробки", де кожен компонент передає результати наступному.

Для виявлення об'єктів використовується модель YOLOv8, інтегрована з використанням бібліотеки Ultralytics YOLO. Система динамічно обирає

оптимальну конфігурацію моделі залежно від доступних ресурсів. Налаштовані параметри інференсу включають поріг достовірності (0.4), поріг перекриття (0.45) та оптимальний розмір вхідного зображення (640x640 пікселів). Для підвищення продуктивності використовується GPU-прискорення через CUDA, пакетна обробка кадрів та оптимізація моделі за допомогою ONNX Runtime (рис. 3.6) [19].

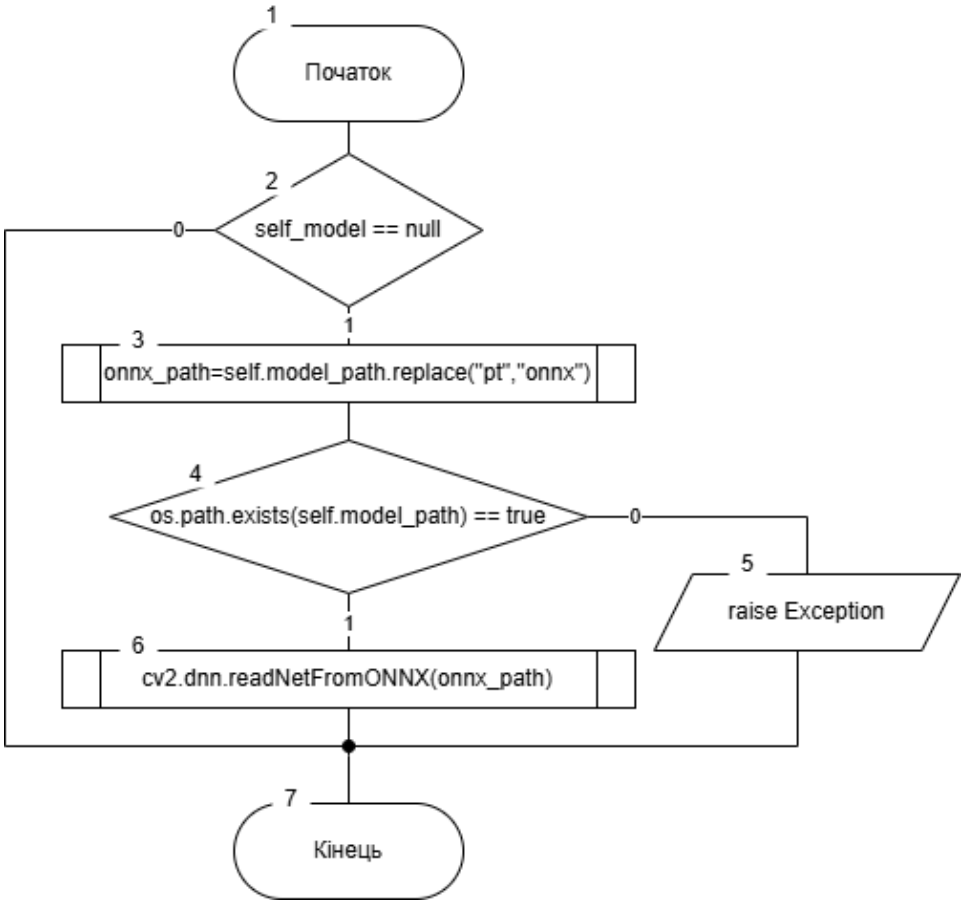


Рисунок 3.6 – Читання моделі формату ONNX

Процес детекції об'єктів включає попередню обробку кадру (масштабування, нормалізацію, перетворення кольорового простору), інференс моделі YOLOv8, постобробку результатів (фільтрацію за порогом достовірності, застосування Non-Maximum Suppression) та формування структурованих результатів з інформацією про кожен виявлений об'єкт.

Для трекінгу об'єктів використовується функціонал бібліотеки Ultralytics – метод .track (рис. 3.7), модифікований для підвищення стабільності.

Процес трекінгу охоплює оновлення стану трекерів з прогнозуванням нових позицій, обчислення візуальних ознак нових детекцій, зіставлення треків з детекціями, оновлення треків на основі результатів зіставлення та фільтрацію результатів для усунення нестабільних треків. Цей алгоритм забезпечує стабільне відстеження об'єктів навіть при частковому перекритті, зміні зовнішнього вигляду та короткочасному виході з кадру.

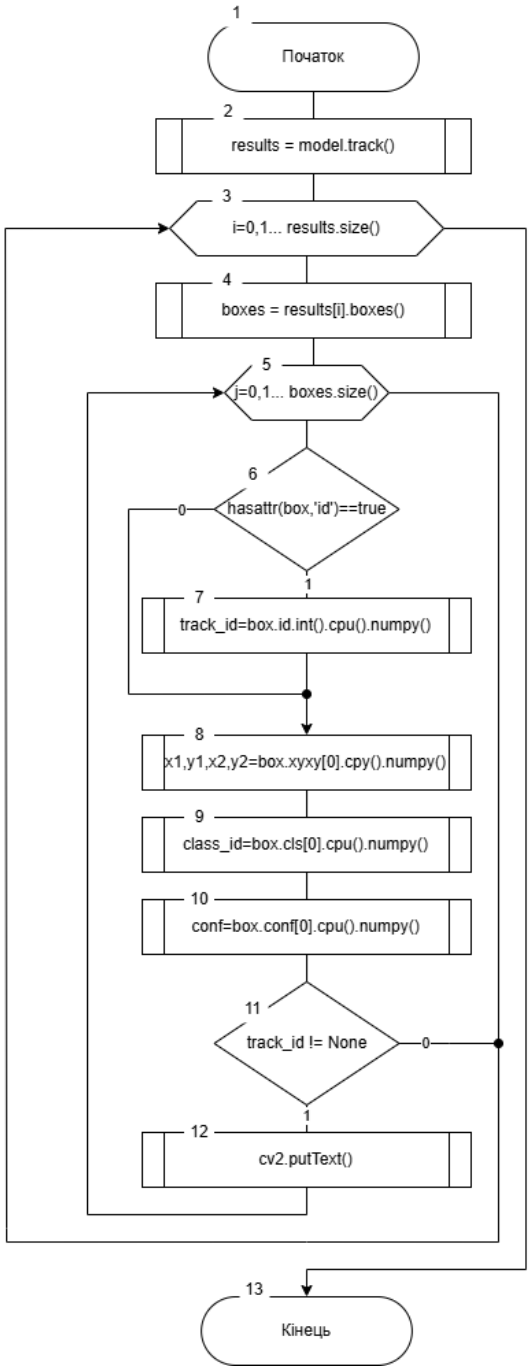


Рисунок 3.7 – Використання модифікованого .track для трекінгу

Для забезпечення роботи в режимі реального часу реалізовано оптимізації: селективну детекцію (не на кожному кадрі), паралельну обробку в окремих потоках, кешування проміжних результатів, динамічне масштабування кадрів залежно від ресурсів та пакетну обробку для ефективного використання GPU. Ці оптимізації дозволяють досягти швидкості до 30 кадрів на секунду на типовому обладнанні з GPU середнього класу.

Модуль надає уніфікований інтерфейс для взаємодії з іншими компонентами системи, включаючи методи для налаштування параметрів детекції та трекінгу, єдиний метод обробки нових кадрів, функції отримання статистики про виявлені об'єкти та інтерфейс для контролю обчислювальних ресурсів. Такий інтерфейс забезпечує гнучку інтеграцію з іншими модулями системи, зокрема з модулем визначення параметрів об'єктів.

3.4 Розробка модуля аналізу кінетичних параметрів об'єктів на динамічних зображеннях

Модуль вимірювання параметрів об'єктів є ключовим компонентом системи, що перетворює результати детекції та трекінгу у кількісні характеристики, необхідні для аналізу та прийняття рішень. Цей модуль відповідає за точне визначення геометричних, кінематичних та інших параметрів об'єктів, виявлених на динамічних зображеннях.

Розроблений модуль має модульну архітектуру, що забезпечує гнучкість та можливість адаптації до різних задач вимірювання. Основні компоненти модуля включають аналізатор геометричних параметрів для визначення розмірів та форми об'єктів; аналізатор кінематичних параметрів, що обчислює параметри руху; аналізатор взаємодій, який досліджує взаємодії між об'єктами; та менеджер вимірювань, що координує роботу всіх аналізаторів.

Модуль побудований за принципом конвеєра обробки даних, де вхідними даними є результати детекції та трекінгу об'єктів, а на виході формуються структуровані дані про параметри об'єктів з оцінкою точності та достовірності.

У системі реалізовано спеціалізований метод для покадрового визначення кінематичних параметрів об'єктів, які були виявлені на відео. Цей метод дозволяє отримувати інформацію про швидкість, прискорення, напрямок руху та компоненти векторів швидкості й прискорення на основі зміщення об’єктів між послідовними кадрами.

На першому етапі реалізовано механізм встановлення відповідності між об'єктами в поточному та попередньому кадрах. Якщо об’єкти мають унікальні ідентифікатори, використовується пряме зіставлення за ID. У випадку відсутності ID застосовується евристичне зіставлення на основі координат центру об’єкта та класу, що дозволяє наближено визначити відповідність за просторовою близькістю. Це забезпечує гнучкість методу та його застосовність у різних сценаріях відстеження.

Після встановлення відповідності виконується розрахунок кінематичних характеристик. Швидкість визначається як відношення зміщення центра об’єкта до інтервалу часу між кадрами. Далі обчислюється напрямок руху у вигляді нормалізованого вектора. Якщо наявна інформація про швидкість об’єкта в попередньому кадрі, додатково розраховується прискорення як зміна вектора швидкості. Результати містять як векторні компоненти прискорення, так і скалярну величину його модуля.

Метод інтегрується з іншими модулями системи, зокрема з модулями обробки потоків та збереження результатів.

У випадку, якщо об'єкт не був виявлений у попередньому кадрі або не вдалося знайти відповідність, метод ініціалізує його кінематичні параметри як нульові, що дозволяє уникнути помилок у подальшому аналізі.

Алгоритм визначення кінетичних параметрів у вигляді блок-схеми зображений на рисунку 3.8.

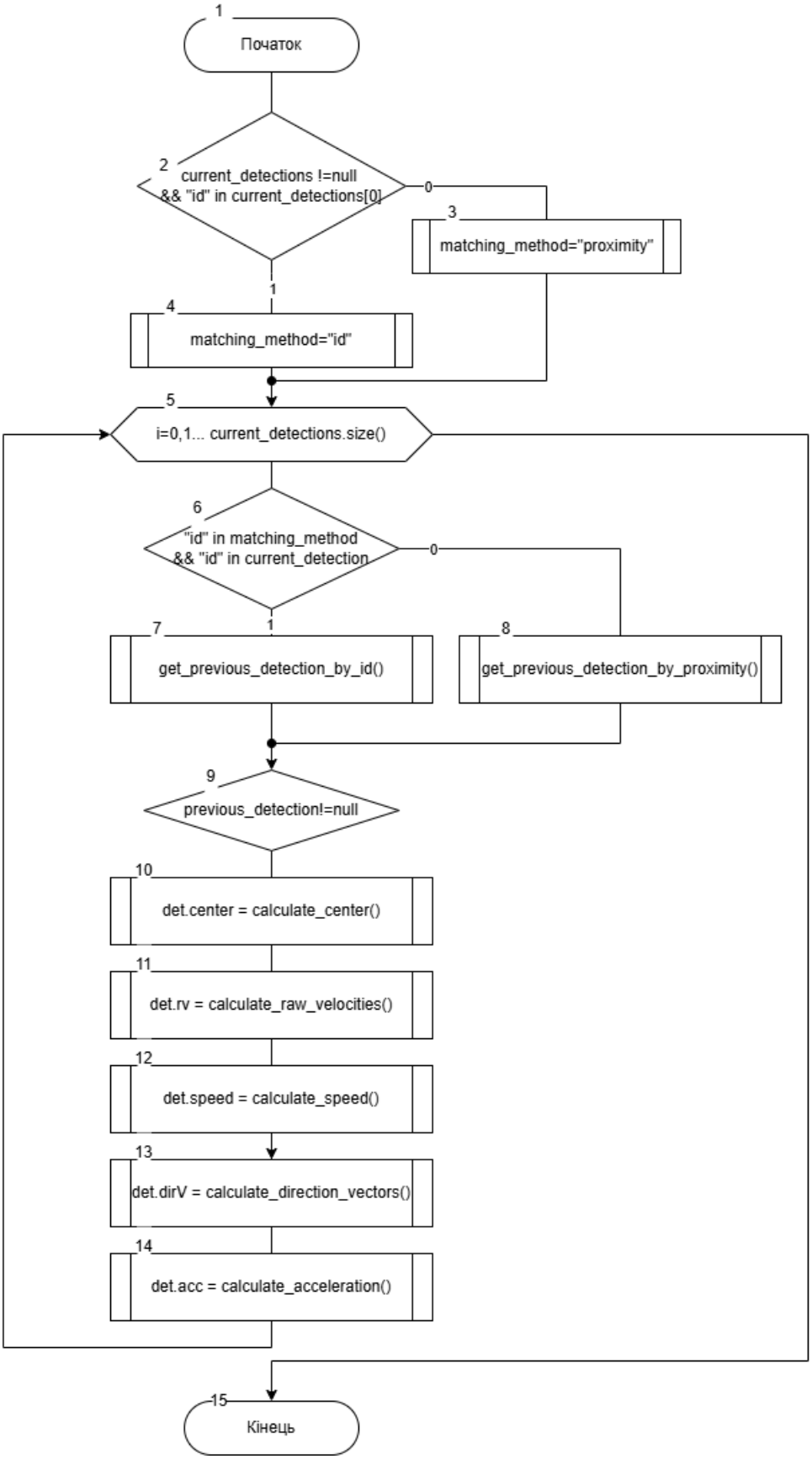


Рисунок 3.8 – Алгоритм визначення кінетичних параметрів

Для повноцінного розуміння сцени важливо аналізувати не тільки параметри окремих об'єктів, але й їх взаємодію. Система включає методи визначення відстаней між об'єктами для аналізу просторових відношень; виявлення перетинів траєкторій для аналізу потенційних взаємодій; аналізу групової динаміки для виявлення соціальних взаємодій; та виявлення взаємозалежних рухів для розуміння причинно-наслідкових зв'язків. Ці методи забезпечують контекстне розуміння сцени та дозволяють ідентифікувати складні патерни поведінки.

Важливим аспектом системи є забезпечення надійності та достовірності вимірювань. Для цього реалізовано методи оцінки похибок вимірювань для розуміння надійності отриманих даних; статистичної фільтрації для підвищення стабільності вимірювань; агрегації даних для отримання узагальнених характеристик; та валідації результатів для перевірки коректності вимірювань. Ці методи забезпечують надійність результатів, що критично важливо для подальшого аналізу.

Модуль надає уніфікований інтерфейс для взаємодії з іншими компонентами системи. Вхідний інтерфейс приймає дані від модуля детекції та трекінгу, включаючи координати об'єктів та їх ідентифікатори. Конфігураційний інтерфейс дозволяє налаштовувати параметри вимірювань, обираючи методи калібрування та алгоритми обробки. Вихідний інтерфейс надає структуровані дані про геометричні та кінематичні параметри об'єктів, а також інформацію про їх взаємодії.

Така архітектура інтерфейсів забезпечує гнучку інтеграцію модуля з іншими компонентами системи, дозволяючи користувачам ефективно отримувати цінну інформацію про об'єкти на динамічних зображеннях.

3.5 Висновки

У третьому розділі було розроблено програмні модулі для визначення параметрів об'єктів на динамічних зображеннях. На основі проведеного аналізу було обрано технологічний стек, що включає Python, OpenCV, YOLOv8, ByteTrack та HTML5+CSS.

Розроблено модуль детекції та трекінгу об'єктів, що поєднує алгоритм YOLOv8 для виявлення об'єктів та ByteTrack для їх відстеження між кадрами. Створено модуль вимірювання параметрів об'єктів, який забезпечує визначення геометричних і кінематичних характеристик об'єктів, а також аналіз їх взаємодій. Реалізовано методи калібрування камери для перетворення піксельних координат у реальні фізичні розміри та алгоритми для визначення швидкості, прискорення й траєкторій руху.

Впроваджено підходи до статистичної обробки та валідації результатів, що підвищують надійність вимірювань в різних умовах. Забезпечено ефективну інтеграцію всіх компонентів через уніфіковані інтерфейси, що дозволяє гнучко адаптувати систему до різних задач.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування системи

Тестування програмного забезпечення – це процес перевірки програмного продукту з метою виявлення помилок, дефектів або неполадок та перевірки відповідності вимогам замовника. Основна мета тестування – забезпечити якість та надійність програмного продукту перед його випуском у виробництво або початком експлуатації.

Для початку тестування необхідно встановити тестову версію програми на комп'ютер з операційною системою Windows або Linux, яка буде використовуватись під час тестування. Перед початком тестування також важливо забезпечити належну документацію, включаючи специфікації вимог та план тестування. Проведення тестування включає кілька етапів та видів тестів.

Першим етапом є функціональне тестування, яке спрямоване на перевірку правильності роботи програми. Це включає в себе тести на введення та виведення даних, тести на реакцію програми на негативні вхідні дані, а також тести на відповідність бізнес логіці програми.

Тестування основного функціоналу застосунку було виконано з використанням методу чорної скриньки. Перелік тест-кейсів було наведено в таблиці 4.1.

Таблиця 4.1 –Тестування роботи застосунку

№	Назва тест-кейсу	Методика тестування	Очікуваний результат	Результат
1	Запуск веб-застосунку	1. Відкрити браузер 2. Перейти за URL адресою системи	Завантажено головну сторінку з інтерфейсом	Виконано
2	Перевірка навігаційного меню	1. Перевірити наявність логотипу 2. Перевірити посилання на "Результати" 3. Посилання на "Налаштування"	Всі елементи навігації відображаються коректно	Виконано

Продовження таблиці 4.1

№	Назва тест-кейсу	Кроки	Очікуваний результат	Результат
3	Завантаження відеофайлу	1. Натиснути "Завантажити відео" 2. Обрати файл з диску 3. Підтвердити завантаження	Файл завантажено, відображено прев'ю	Виконано
4	Запуск детекції об'єктів	1. Обрати завантажений відеофайл 2. Натиснути "Почати детекцію" 3. Дочекатися завершення обробки	Детекція запущена, показано індикатор прогресу	Виконано
5	Перегляд результатів аналізу	1. Перейти до розділу "Результати" 2. Обрати проаналізований файл 3. Переглянути детальну інформацію	Відображено результати з виявленими об'єктами	Виконано
6	Експорт результатів	1. Відкрити результати аналізу 2. Натиснути "Експорт" 3. Обрати формат (CSV/JSON) 4. Зберегти файл	Файл з результатами збережено в обраному форматі	Виконано
7	Налаштування параметрів детекції	1. Перейти до "Налаштувань" 2. Змінити поріг достовірності на 0.6 3. Зберегти зміни	Параметри збережено, система використовує нові значення	Виконано
8	Робота з великим відеофайлом	1. Завантажити відео розміром >500MB 2. Запустити детекцію 3. Моніторити споживання ресурсів	Файл оброблено без помилок, стабільна робота системи	Виконано
9	Обробка некоректного файлу	1. Спробувати завантажити текстовий файл 2. Підтвердити завантаження	Відображено помилку "Неподтримуваний формат файлу"	Виконано
10	Адаптивність інтерфейсу	1. Змінити розмір вікна браузера 2. Перевірити відображення на мобільному пристрої 3. Протестувати бургер-меню	Інтерфейс адаптується під різні розміри екрану	Виконано

Другим етапом є тестування відмов, спрямоване на виявлення дефектів та неполадок у програмному забезпеченні. Проводиться аналіз програмного забезпечення з точки зору його здатності відновлювати роботу після виникнення помилок або відмов. Такі сценарії тестування можуть включати штучне введення неправильних даних, непередбачувані вхідні параметри або спроби взаємодії з програмою в умовах, які відповідають екстремальним ситуаціям або навіть випадкам, коли програмне забезпечення працює у відсутності зв'язку з іншими компонентами або системою.

Особлива увага приділяється обробці виняткових ситуацій, наприклад, відсутності доступу до файлової системи, розриву з'єднання з джерелом відеопотоку або пошкоджених відеокадрів. У таких випадках тестується поведінка системи щодо коректного повідомлення користувача про помилку, відновлення стабільного стану без повного перезапуску програми та збереження вже оброблених даних. Також перевіряється, чи виконується журналювання помилок і чи генерується відповідна інформація для подальшої діагностики. Це дозволяє оцінити не лише стійкість системи, а й її готовність до роботи у реальних умовах експлуатації, де відмови можуть бути випадковими або періодичними.

Після завершення функціонального тестування та тестування відмов, проводиться додаткове тестування для визначення продуктивності програмного забезпечення. Цей етап включає тести швидкодії, які спрямовані на оцінку швидкості виконання певних операцій або завдань програмою. Під час цього тестування вимірюються час відповіді програми на різноманітні запити або дії користувача.

Окремим етапом тестування може бути тестування сумісності, яке спрямоване на перевірку взаємодії програмного забезпечення з різними операційними системами, версіями програм та апаратними платформами. Під час цього тестування перевіряється, чи працює програма на різних конфігураціях та середовищах, а також чи виникають конфлікти або проблеми під час взаємодії з іншими програмами.

Під час першого етапу тестування системи обробки відеопотоків перевіряється реакція програми на неправильні формати відеофайлів. При спробі завантаження відео, програма пропонує користувачеві завантажити відеофайл у підтримуваному форматах: MP4, AVI, MOV, MKV (рис. 4.1).

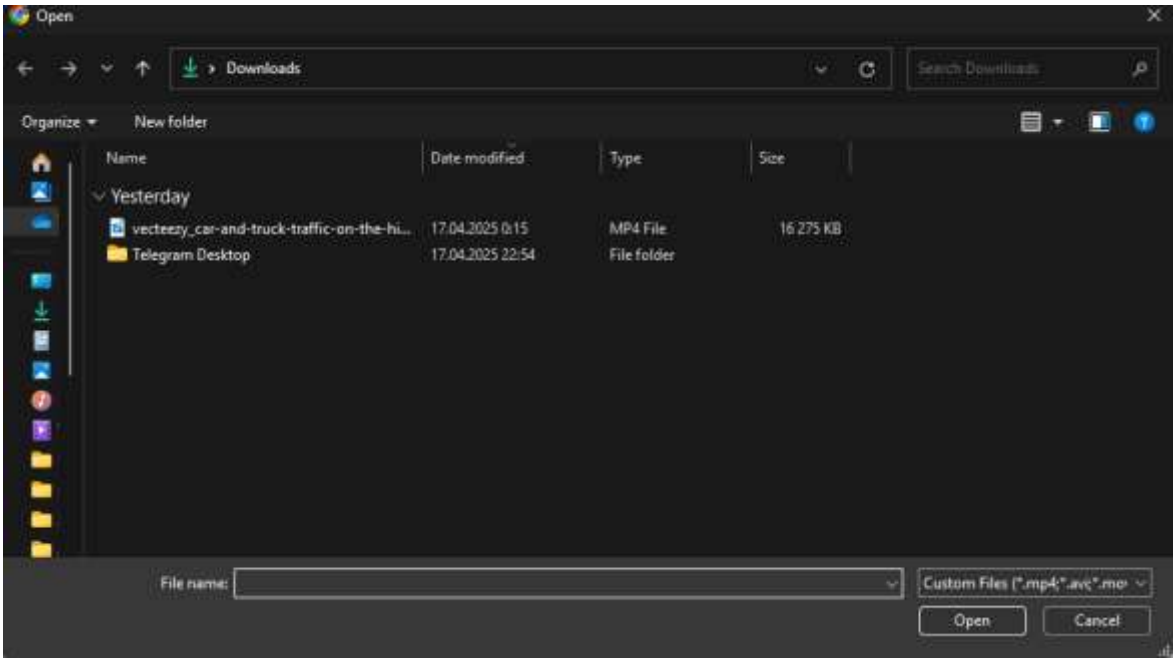


Рисунок 4.1 – Вікно вибору файлів для завантаження

Після проведення тестування програмного забезпечення на завантаження відеофайлів неправильного формату, було прийнято рішення провести додаткове тестування, спрямоване на перевірку реакції програмного забезпечення на відсутність вхідних даних. Це було зроблено з метою визначення, як програма реагує на таку ситуацію та чи забезпечує вона користувача відповідними повідомленнями або інструкціями. У результаті цього тестування було виявлено, що програмне забезпечення обробляє такий сценарій використання користувачем та надає інформацію про можливий шлях рішення (рис. 4.2).

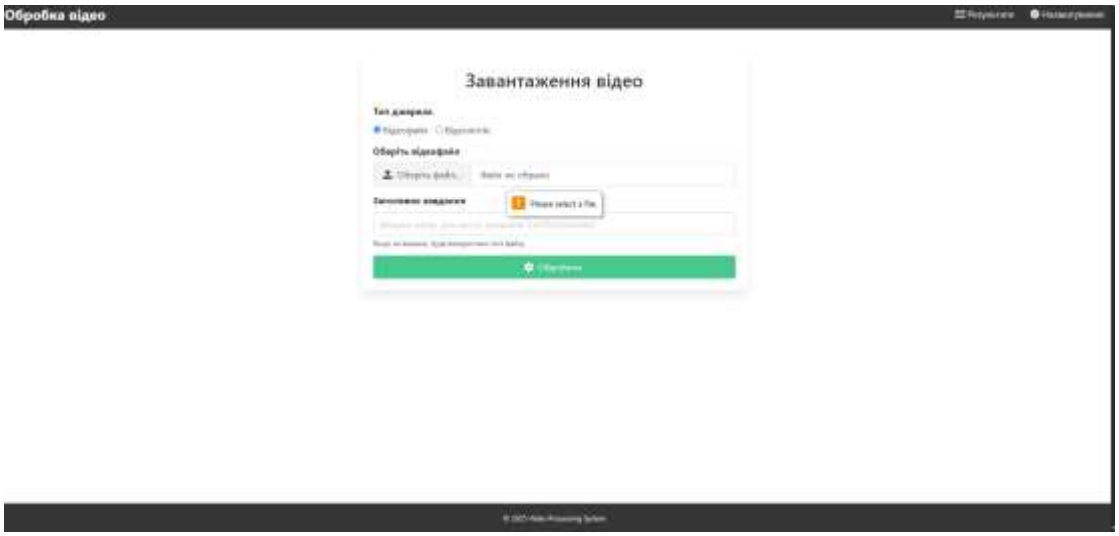


Рисунок 4.2 – Спроба розпочати обробку без обраного файлу

Після успішної перевірки функціональності програмного забезпечення на правильність обробки вхідних даних, наступним кроком є проведення тестування на реакцію програми на спробу обробки пошкодженого відеофайлу. В цьому випадку, використовується відеофайл з пошкодженими метаданими або фрагментами. На рисунку 4.3 наведена реакція системи під час обробки пошкодженого відеофайлу.

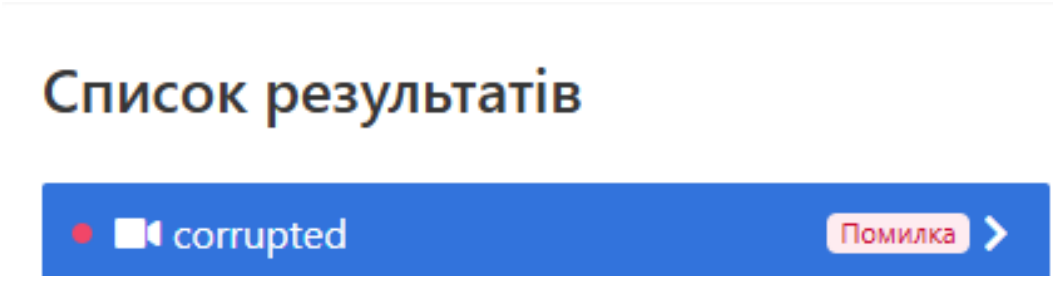


Рисунок 4.3 – Помилка при обробці неккоректного файлу

Після перевірки завантаження у робоче середовище програмного забезпечення даних, які були недійсними або пошкодженими, перейдемо до завантаження звичайного відеофайлу. Цей процес дозволить переконатися, що програмне забезпечення працює коректно з валідними даними та може ефективно обробляти їх. Після завершення завантаження проаналізуємо результати, які

програмне забезпечення виведе щодо завантаженого відеофайлу у головне робоче середовище. Результати тестування зображені на рисунку 4.4.

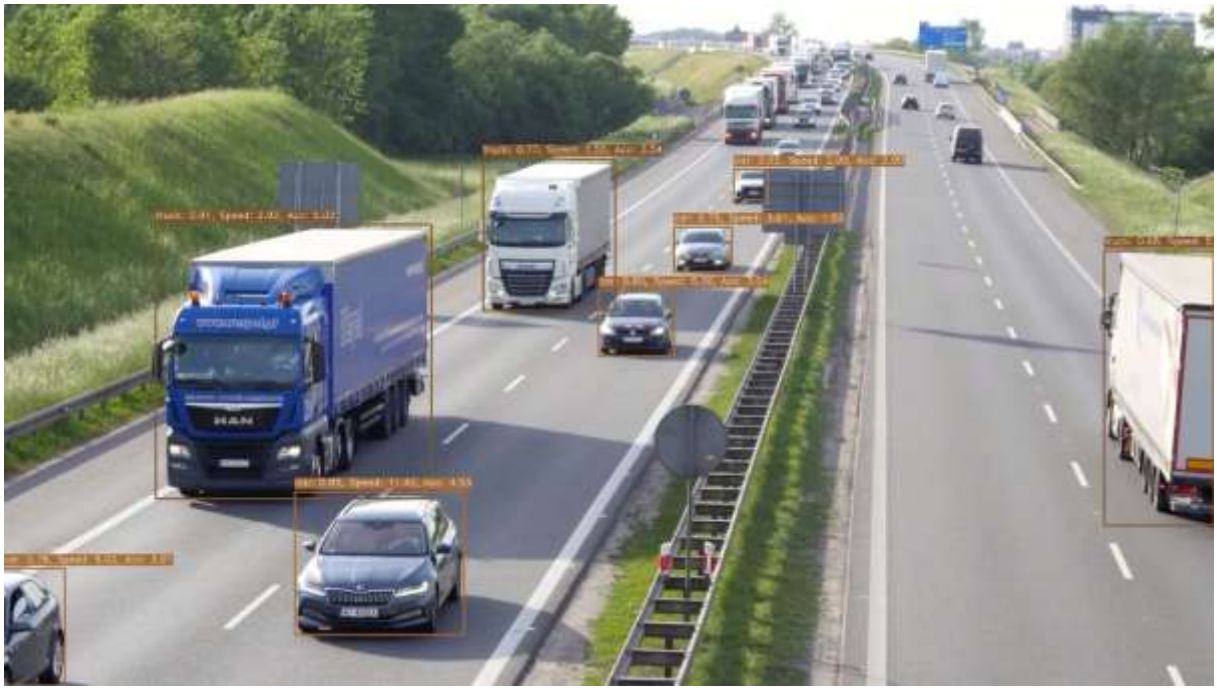


Рисунок 4.4 – Результат успішної детекції

У рамках тестування програмного забезпечення також було проведено аналіз його сумісності з різними версіями операційних систем та мінімальною конфігурацією апаратного забезпечення. Зокрема, програмний продукт був перевірений на сумісність з наступними операційними системами: Windows 11, Ubuntu 24.0.2.

У додаток до тестування обробки звичайних відеофайлів, програмне забезпечення було також протестовано на підключенні до відеопотоку через RTSP протокол. Результати цього тестування зображені на рисунку 4.5. За результатами можна побачити, що програмне забезпечення здатне до обробки потокового відео в реальному часі, включаючи детекцію об'єктів та їх аналіз.

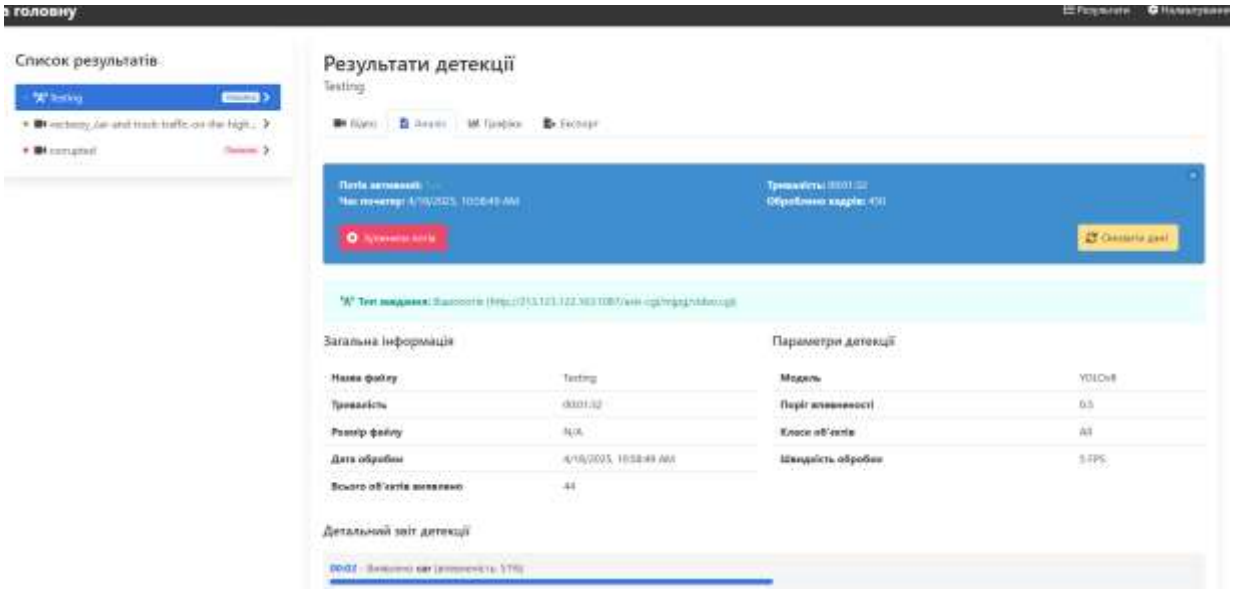


Рисунок 4.5 – Результат обробки публічного стріму

Після успішного завершення тестування завантаження, аналізу та обробки вхідних даних програмним забезпеченням, було проведено тестування функціоналу візуалізації та відображення результатів обробки відео. Для цього необхідно було завантажити відеофайл у робоче середовище програмного забезпечення та запустити процес обробки, натиснувши на відповідну кнопку в інтерфейсі.

Після виконання обробки завантаженого відео, було звірено результати детекції у форматі статистики. На рисунку 4.6 зображені результати генерації статистики та графіків на основі обробленого відео. Як можна побачити з результатів візуалізації даних, система визначає та структурує сформовані дані у вигляді кругового графіку розподілу класів, стовпчастої діграми за об’ємом детекцій та окремим графіком рівень середньої впевненості для кожного із ідентифікованих класів, також формується графік що зображує періодичність та частоту детекцій різних класів протягом всього відеофайлу або потоку.

Крім цього, при наведенні курсору на елементи графіків відображається розширена інформація про відповідний сегмент даних.

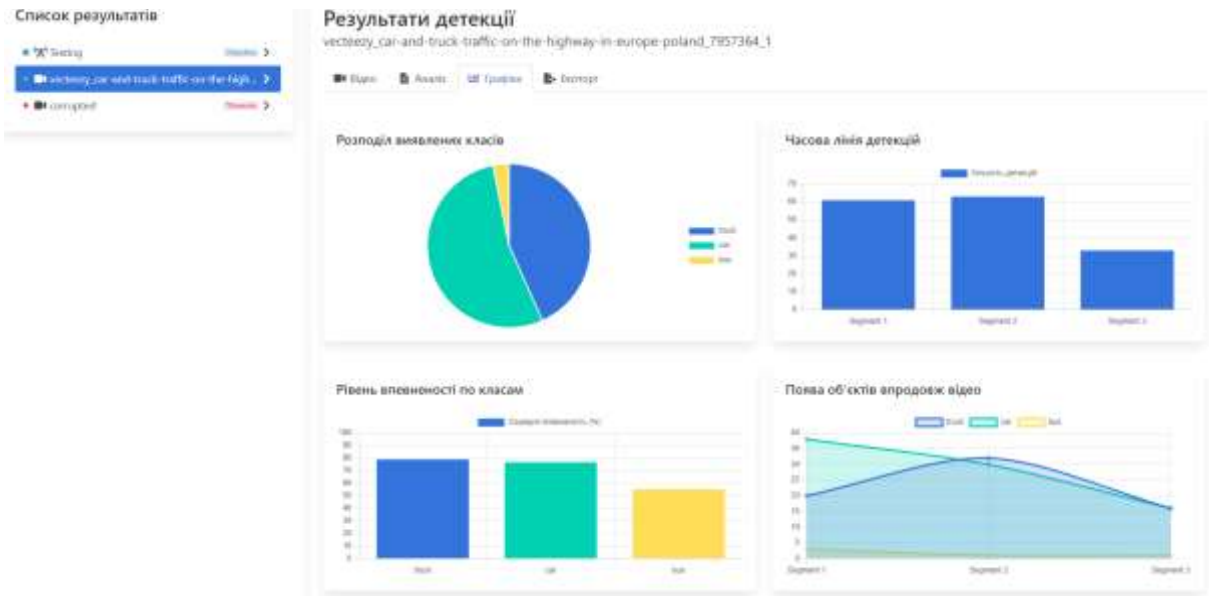


Рисунок 4.6 – Статистика детекцій, на прикладі відео

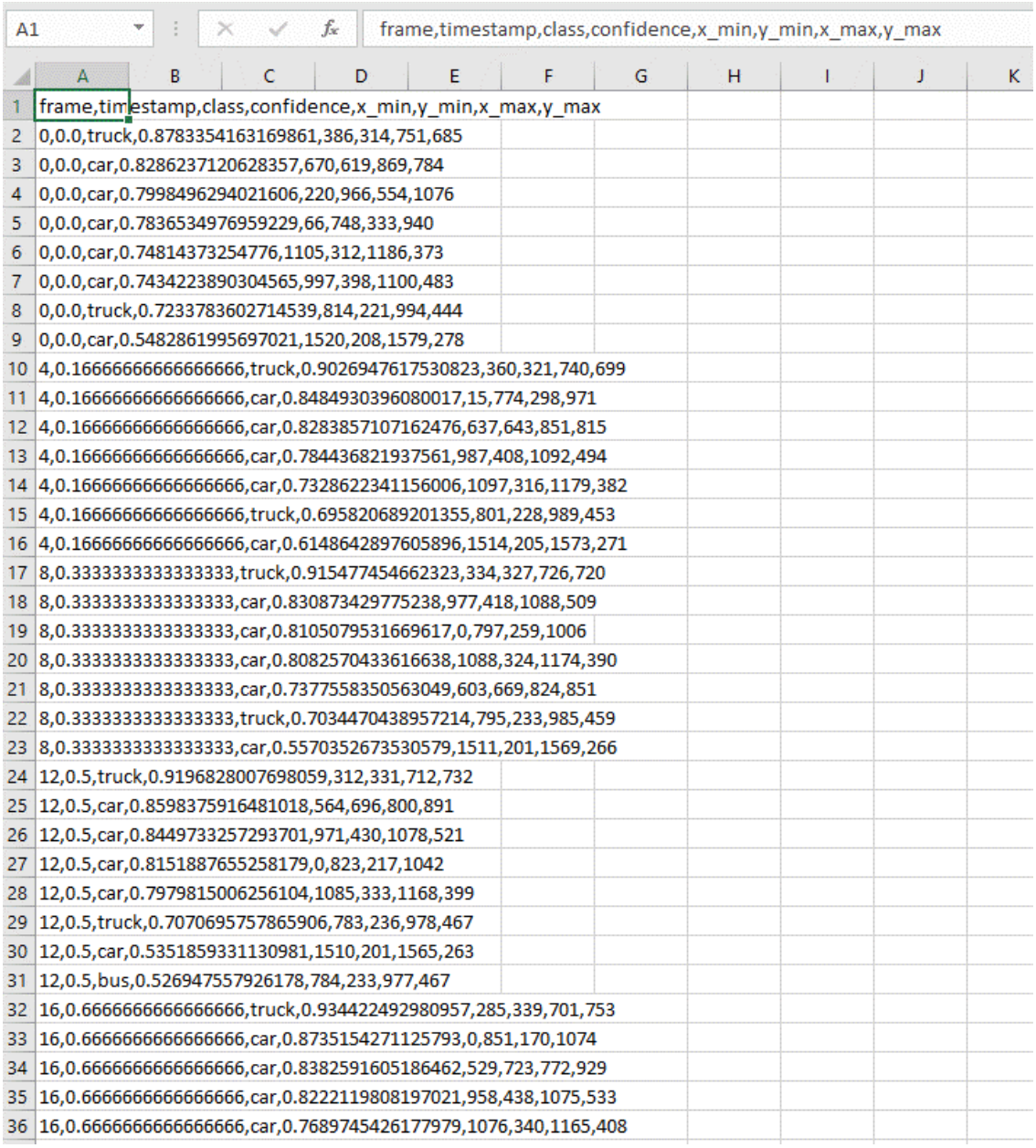
Під час тестування програмного забезпечення також було проведено аналіз процесу експорту даних у наступні формати:

- CSV;
- JSON;
- XLSX.

Для перевірки коректності роботи цієї функції використовувався типовий відеофайл, який пройшов обробку в системі відеоаналітики. Після завершення обробки було здійснено експорт результатів аналізу до зазначених форматів.

Особливу увагу було приділено експорту результатів детекції автомобільного трафіку у форматі CSV. Цей формат був обраний завдяки своїй зручності для перегляду, аналізу та подальшої обробки даних у табличному вигляді. Дані з детекції включали часові мітки, координати об’єктів, класифікацію транспортних засобів та інші параметри, що були зібрані системою під час аналізу відео.

На рисунку 4.7 представлено фрагмент таблиці з результатами, експортованими у форматі CSV.



	frame,timestamp,class,confidence,x_min,y_min,x_max,y_max
1	frame,timestamp,class,confidence,x_min,y_min,x_max,y_max
2	0,0.0,truck,0.8783354163169861,386,314,751,685
3	0,0.0,car,0.8286237120628357,670,619,869,784
4	0,0.0,car,0.7998496294021606,220,966,554,1076
5	0,0.0,car,0.7836534976959229,66,748,333,940
6	0,0.0,car,0.74814373254776,1105,312,1186,373
7	0,0.0,car,0.7434223890304565,997,398,1100,483
8	0,0.0,truck,0.7233783602714539,814,221,994,444
9	0,0.0,car,0.5482861995697021,1520,208,1579,278
10	4,0.16666666666666666,truck,0.9026947617530823,360,321,740,699
11	4,0.16666666666666666,car,0.8484930396080017,15,774,298,971
12	4,0.16666666666666666,car,0.8283857107162476,637,643,851,815
13	4,0.16666666666666666,car,0.784436821937561,987,408,1092,494
14	4,0.16666666666666666,car,0.7328622341156006,1097,316,1179,382
15	4,0.16666666666666666,truck,0.695820689201355,801,228,989,453
16	4,0.16666666666666666,car,0.6148642897605896,1514,205,1573,271
17	8,0.3333333333333333,truck,0.915477454662323,334,327,726,720
18	8,0.3333333333333333,car,0.830873429775238,977,418,1088,509
19	8,0.3333333333333333,car,0.8105079531669617,0,797,259,1006
20	8,0.3333333333333333,car,0.8082570433616638,1088,324,1174,390
21	8,0.3333333333333333,car,0.7377558350563049,603,669,824,851
22	8,0.3333333333333333,truck,0.7034470438957214,795,233,985,459
23	8,0.3333333333333333,car,0.5570352673530579,1511,201,1569,266
24	12,0.5,truck,0.9196828007698059,312,331,712,732
25	12,0.5,car,0.8598375916481018,564,696,800,891
26	12,0.5,car,0.8449733257293701,971,430,1078,521
27	12,0.5,car,0.8151887655258179,0,823,217,1042
28	12,0.5,car,0.7979815006256104,1085,333,1168,399
29	12,0.5,truck,0.7070695757865906,783,236,978,467
30	12,0.5,car,0.5351859331130981,1510,201,1565,263
31	12,0.5,bus,0.526947557926178,784,233,977,467
32	16,0.6666666666666666,truck,0.934422492980957,285,339,701,753
33	16,0.6666666666666666,car,0.8735154271125793,0,851,170,1074
34	16,0.6666666666666666,car,0.8382591605186462,529,723,772,929
35	16,0.6666666666666666,car,0.8222119808197021,958,438,1075,533
36	16,0.6666666666666666,car,0.7689745426177979,1076,340,1165,408

Рисунок 4.7 – Експорт результатів детекції у форматі CSV

Це дозволяє ефективно використовувати отримані дані для подальшого аналізу та обробки.

4.2 Розробка інструкції користувача

Інструкція користувача передбачає визначення технічних вимог для запуску програмного продукту. Деталі щодо мінімальної та рекомендованої конфігурації розміщено в таблиці 4.1.

Таблиця 4.1 – Вимоги до апаратного забезпечення

Вимоги до конфігурування	Рекомендовано	Мінімально
Процесор	Intel Core i7	Intel Core i5
Оперативна пам'ять	16 GB RAM	8 GB RAM
Вільний простір на диску	128 GB	32 GB
Тип носія	NVME SSD	SATA SSD
Відеокарта	NVIDIA GeForce RTX 2060 або новіше	NVIDIA GeForce GTX 1060 6GB
Операційна система	Windows 11 або Ubuntu 24+	Windows 10 або Ubuntu 22+

Після інсталяції та запуску програмного забезпечення, користувач переходить на головний екран системи обробки відеопотоків. Для початку роботи з системою, користувач повинен виконати наступні кроки:

1. Завантаження відеофайлу або підключення до потоку:

- Для завантаження відеофайлу: натисніть кнопку "Обрати файл" і виберіть відеофайл у підтримуваному форматі (MP4, AVI, MOV, MKV, WEBM).
- Для підключення до відеопотоку: виберіть опцію "Відеопотік", введіть URL потоку в форматі RTSP, HTTP або HTTPS та вкажіть бажану швидкість обробки кадрів (FPS).

2. Налаштування параметрів обробки:

- Вказати назву для завдання обробки (опційно).
- Для потокового відео: налаштувати швидкість обробки кадрів (рекомендовано 5 FPS для стандартного використання).

3. Запуск обробки:

- Натиснути кнопку "Обробити" для відеофайлів або "Запустити потік" для потокового відео.
- Система перенаправить користувача на сторінку результатів, де можна буде відстежувати прогрес обробки.

4. Перегляд результатів:

- Після завершення обробки або в режимі реального часу для потоків, користувач може переглядати результати на сторінці результатів.
- Доступні вкладки: "Відео" (для перегляду обробленого відео), "Аналіз" (для перегляду детальної інформації про виявлені об'єкти), "Графіки" (для візуалізації статистики) та "Експорт" (для вивантаження результатів у різних форматах).

5. Робота з обробленими результатами:

- Перегляд відеорезультатів: вкладка "Відео" дозволяє переглядати оброблене відео з накладеними боксами детекції.
- Аналіз результатів: вкладка "Аналіз" відображає детальну інформацію про виявлені об'єкти, їх кількість, час появи тощо.
- Візуалізація статистики: вкладка "Графіки" містить діаграми розподілу класів об'єктів, часової лінії детекцій та іншу аналітичну інформацію.
- Експорт даних: вкладка "Експорт" дозволяє зберегти результати аналізу у форматах CSV, JSON, XLSX.

6. Налаштування системи:

Сторінка "Налаштування" дозволяє змінювати параметри роботи системи, включаючи поріг впевненості детекції, вибір класів об'єктів для виявлення, налаштування відеопотоків та параметри експорту.

7. Керування завданнями:

- На сторінці "Результати" відображаються всі завдання з обробки відео з їх поточним статусом.
- Для завдань у процесі обробки доступні опції зупинки та оновлення статусу.
- Для завершених завдань доступні опції перегляду результатів та видалення.

Для оптимальної роботи системи рекомендується:

- Використовувати браузер Google Chrome або Mozilla Firefox останньої версії.

- Забезпечити стабільне підключення до мережі Інтернет при роботі з відеопотоками.
- Не завантажувати відеофайли розміром більше 2 ГБ за один раз.
- Використовувати графічні процесори з наявними CUDA ядрами для машини на якій здійснюється обробка відеозображень, задля покращеної швидкодії обробки кадрів.

4.3 Висновки

У четвертому розділі проведено тестування програмних модулів системи обробки відеопотоків, під час якого перевірялися реакція на помилки, підтримка різних форматів вхідних даних і сумісність із різними джерелами відео. Система успішно працює з форматами MP4, AVI, MOV, MKV та потоками через RTSP, HTTP і HTTPS, забезпечуючи реальну класифікацію об'єктів, візуалізацію результатів та генерацію статистики. Особливу увагу приділено перевірці на стійкість до пошкоджених даних і тестуванню на різних апаратних конфігураціях.

У процесі тестування виявлено та усунуто дрібні недоліки, а також додано нові функції для покращення зручності використання. Розроблено детальну інструкцію користувача з порадами щодо оптимальних налаштувань, яка дозволяє легко працювати з системою навіть без технічної підготовки.

Крім того було розроблено інструкцію користувача по експлуатації програмного застосунку.

ВИСНОВКИ

У бакалаврській кваліфікаційній роботі було розроблено систему обробки відеопотоків з функціями детекції та аналізу об'єктів у реальному часі. Для розробки використовувалися сучасні технології веб-розробки та комп'ютерного зору, а реалізація програмного продукту відповідає методичним вказівкам.

Під час дослідження було проведено аналіз сучасного стану проблеми відеоаналітики. Були розглянуті основні аналоги програмних рішень, виявлені їхні недоліки та переваги, які було враховано при розробці власної системи. На основі цього порівняння були сформульовані основні завдання та вимоги до розроблюваної системи [21,22].

Під час аналізу технологій розробки було обґрунтовано вибір мови програмування Python для серверної частини, використання фреймворку Flask для створення веб-інтерфейсу, бібліотеки OpenCV для обробки відео та нейронної мережі YOLOv8 для детекції об'єктів. Для клієнтської частини було використано HTML, CSS (Bulma), JavaScript з бібліотеками Chart.js для візуалізації даних.

Під час розробки застосунку було зроблено наступні завдання:

- розробка модуля детекції об’єктів, класифікації, трекінгу, аналізу їх швидкості, прискорення та розмірів;
- розробка графічного інтерфейсу для роботи із програмним забезпеченням, переглядом та експортом результатів;
- розробка модуля візуалізації результатів;
- модифікувати метод позиціювання об'єктів у потоці даних для підвищення точності визначення кінематичних параметрів;
- розробка алгоритму збереження\експорту результатів;
- інтеграція модулів та алгоритмів у єдиний застосунок для визначення параметрів на динамічних зображеннях.

Було розроблено схеми загальної архітектури системи, алгоритму обробки відеопотоків та модуля детекції об'єктів. Також було розроблено модель системи з

використанням UML діаграм, що демонструють структуру та взаємодію компонентів програмного забезпечення.

Важливою особливістю розробленої системи є її здатність працювати як з відеофайлами, так і з потоковим відео в режимі реального часу, що розширює сферу застосування програмного продукту. Система демонструє високу точність детекції різних класів об'єктів (людей, транспортних засобів тощо) і забезпечує зручні інструменти для аналізу та візуалізації результатів.

Тестування програми довело повну працездатність даного програмного продукту та відповідність поставленому технічному завданню. Система успішно обробляє різні формати відеофайлів, підтримує підключення до відеопотоків через RTSP, HTTP та HTTPS протоколи, ефективно виконує детекцію об'єктів та генерує детальну статистику. Також було розроблено інструкцію користувача, яка забезпечує швидке освоєння системи.

Розроблена система обробки відеопотоків може бути впроваджена в різних сферах, включаючи відеоспостереження, контроль безпеки, аналіз трафіку, моніторинг громадських просторів та інші галузі, де потрібна автоматизована обробка відео та детекція об'єктів. Модульна архітектура системи забезпечує можливість подальшого розширення функціональності та інтеграції з іншими системами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Zhang, Y., Liu, R., Singh, A., & Chen, M. (2025). A Retrospective and Emerging Trend Survey on Deep Learning-Based Multi-Object Tracking. Artificial Intelligence Review, vol. 58(8), doi:10.1007/s10462-025-11212-y. pp 42.

2. Xiong, Y., Zhang, H., & Li, M. (2024). Development and challenges of object detection: A survey. Neurocomputing, *598*, 128102. pp 25

3. Zheng, L., Zhou, T., Jiang, R., Peng, Y. (2022). Survey of Video Object Detection Algorithms Based on Deep Learning. In Proceedings of the 2021 4th International Conference on Algorithms, Computing and Artificial Intelligence (ACAI ’21). New York, NY: ACM. pp 71.

4. Хитрич С.С., Рейда О.М. Методи визначення параметрів об'єктів на динамічних зображеннях. Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії. Вінниця: Вінницький національний технічний університет, 2025.

5. Amazon Rekognition [Електроний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/rekognition/ (дата звернення 20.04.2025).

6. Google cloud vision API [Електроний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/vision (дата звернення 20.04.2025).

7. Microsoft Azure Computer Vision [Електроний ресурс] – Режим доступу до ресурсу: https://azure.microsoft.com/en-us/products/ai-services/ai-vision (дата звернення 20.04.2025).

8. Howse, J., & Minichino, J. (2025). Learning OpenCV 5: Computer Vision with Python, 4th ed. Packt Publishing. (432 pp.)

9. OpenCV Documentation. Модулі Video I/O та Video Analysis. [Електронний ресурс] – Режим доступу: https://docs.opencv.org/4.x/ (дата звернення 23.04.2025)

10. TensorFlow Models GitHub – розділ Object Detection API (README). [Електронний ресурс] – Режим доступу:

https://github.com/tensorflow/models/tree/master/research/object_detection (дата звернення 23.04.2025)

11. TensorFlow 2 Object Detection API Tutorial. Read the Docs. [Електронний ресурс] – Режим доступу: https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/ (дата звернення 23.04.2025)

12. Du, Y., Zhao, Z., Song, Y., Zhao, Y., Su, F., Gong, T., & Meng, H. (2023). StrongSORT: Make DeepSORT Great Again. IEEE Transactions on Multimedia, 25, 8725–8737.

13. Moryossef, A. (2024). Optimizing Hand Region Detection in MediaPipe Holistic Full-Body Pose Estimation to Improve Accuracy and Avoid Downstream Errors. arXiv:2405.03545. 9 pp.

14. Cheng, T., Song, L., Ge, Y., Liu, W., Wang, X., & Shan, Y. (2024). YOLO-World: Real-Time Open-Vocabulary Object Detection. Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2024), pp. 16901–16911.

15. Aggarwal, S. (2023). Flask Framework Cookbook (3rd ed.). Packt Publishing. 318 pp.

16. Thomas, J., Potiekhin, O., Lauhakari, M., Shah, A., & Berning, D. (2024). Creating Interfaces with Bulma. In: Pro Web Development with Bulma (Chapter 13). Packt Publishing. (pp. 9–34)

17. Chen, J. (2023). Learn OpenCV with Python by Examples: Implement Computer Vision Algorithms Provided by OpenCV with Python for Image Processing, Object Detection and Machine Learning (2nd ed.). 316 pp.

18. BookAuthority. (2025). Mastering NumPy: The Ultimate Guide to Data Manipulation in Python. BookAuthority list.

19. ONNX Runtime Documentation. [Електронний ресурс] – Режим доступу до ресурсу: https://onnxruntime.ai/docs/ (дата звернення: 23.04.2025).

20. Rosebrock, A. Measuring size of objects in an image with OpenCV. [Електронний ресурс] – Режим доступу до ресурсу:

https://pyimagesearch.com/2016/03/28/measuring-size-of-objects-in-an-image-with-opencv/ (дата звернення: 23.04.2025).

21. Методичні вказівки до виконання курсових проєктів з дисципліни "Проєктний практикум" для студентів спеціальності 121 "Інженерія програмного забезпечення" / Уклад. В.В. Войтко, Г. О. Черноволик – Вінниця: ВНТУ, 2022. – 44 с.

22. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення» / уклад. : О. Н. Романюк, Г. О. Черноволик. – Вінниця : ВНТУ, 2022. – 51 с.

ДОДАТКИ

59

Додаток А – Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедрою ПЗ
д.т.н., проф. О. Н. Романюк
"25" березня 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка програмного модуля автоматичного визначення характеристик об'єктів на динамічних зображеннях із застосуванням нейромереж» за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доцент О.М. Рейда
"24" березня 2025 р.
Виконав:
студент гр. 1ПІ-21Б С.С. Хитрич
"24" березня 2025 р.

Вінниця – 2025

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка програмного модуля автоматичного визначення характеристик об’єктів на динамічних зображеннях із застосуванням нейромереж».

Галузь застосування – інформаційні технології, комп’ютерний зір.

2. Підстава для розробки.

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ № 97 від «20» березня 2025 р. ректора по ВНТУ про закріплення тем БКР.

3. Мета та призначення розробки.

Мета роботи полягає у підвищення ефективності обробки та аналізу відеоматеріалів за рахунок використання нейромереж у автоматизованих системах позиціювання об'єктів

Призначення роботи – розробка та програмна реалізація модуля для автоматичного визначення характеристик об’єктів на динамічних зображеннях.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БКР.

1. Zhang, Y., Liu, R., Singh, A., & Chen, M. (2025). *A Retrospective and Emerging Trend Survey on Deep Learning-Based Multi-Object Tracking*. *Artificial Intelligence Review*, vol. 58(8), doi:10.1007/s10462-025-11212-y. pp 42.

2. Zheng, L., Zhou, T., Jiang, R., Peng, Y. (2022). Survey of Video Object Detection Algorithms Based on Deep Learning. In Proceedings of the 2021 4th International Conference on Algorithms, Computing and Artificial Intelligence (ACAI ’21). New York, NY: ACM. pp 71.

3. Cheng, T., Song, L., Ge, Y., Liu, W., Wang, X., & Shan, Y. (2024). YOLO-World: Real-Time Open-Vocabulary Object Detection. Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2024), pp. 16901–16911.

4. Chen, J. (2023). Learn OpenCV with Python by Examples: Implement Computer Vision Algorithms Provided by OpenCV with Python for Image Processing, Object Detection and Machine Learning (2nd ed.). 316 pp.

5. Технічні вимоги

Вхідні дані: середовище розробки Visual Studio Code; мова програмування Python; Flask для серверної частини; нейронна модель YOLOv8.

Вихідні дані до роботи: модель розробки – модуль для автоматичного аналізу характеристик об’єктів на динамічних зображеннях, REST API для взаємодії з модулем, веб-інтерфейс.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред’являється по закінченню робіт:

1. Пояснювальна записка до БКР;

2. Технічне завдання;

3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз розвитку технологій обробки динамічних зображень та постановка задач дослідження	26.03.2025 – 03.04.2025
2	Розробка моделі та алгоритмів програмного продукту	04.04.2025 – 16.04.2025
3	Розробка програмного модулів визначення параметрів об’єктів на динамічних зображеннях	17.04.2025 – 17.05.2025
4	Тестування програми	18.05.2025 – 28.05.2025
5	Оформлення матеріалів до захисту БКР	28.05.2025 – 10.06.2025

10. Порядок контролю та прийняття

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

63

Додаток Б – Протокол перевірки кваліфікаційної роботи

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка програмного модуля автоматичного визначення характеристик об'єктів на динамічних зображеннях із застосуванням нейромереж

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра програмного забезпечення, ФІТКІ
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 9,2 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

(прізвище, ініціали, посада) (підпис)

(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник (підпис) Рейда О.М. к.т.н., доц. каф. ПЗ
(прізвище, ініціали, посада)

Здобувач (підпис) Хитрич С.С.
(прізвище, ініціали)

Додаток В – Лістинг програми

app.py

```
from flask import Flask, render_template, request, redirect, url_for, flash
from flask import send_file, jsonify
from datetime import datetime, timezone
import os
import tempfile
import pytz
import csv
import uuid
import logging
import xlsxwriter
from flask_migrate import Migrate

from config import Config
from models import db, Settings, VideoJob
from services.service_factory import ServiceFactory
from constants import (
    JobStatus,
    SourceType,
    FileFormats,
    ExportFormats,
    ProcessingSettings,
)

# Configure logging
logging.basicConfig(
    level=logging.INFO, format="%(asctime)s - %(name)s - %(levelname)s - %(message)s"
)
logger = logging.getLogger(__name__)

# Initialize Flask application
app = Flask(__name__)
app.config.from_object(Config)

# Initialize database
db.init_app(app)
migrate = Migrate(app, db)

# Create service factory and initialize all services
service_factory = ServiceFactory(app)
services = service_factory.create_all_services()

# Extract services for easier access
storage_service = services["storage_service"]
video_service = services["video_service"]
media_service = services["media_service"]
```

```
stream_service = services["stream_service"]
detection_service = services["detection_service"]
job_service = services["job_service"]
result_service = services["result_service"]
processing_service = services["processing_service"]

stream_service.set_job_service(job_service)
stream_service.set_processing_service(processing_service)


@app.template_filter("format_datetime")
def format_datetime(value, format="%d.%m.%Y %H:%M:%S", timezone_name="Europe/Kiev"):
    """Format datetime to specified timezone"""
    if value is None:
        return ""

    if hasattr(value, "tzinfo") and value.tzinfo is not None:
        local_timezone = pytz.timezone(timezone_name)
        local_dt = value.astimezone(local_timezone)
    else:
        utc_dt = value.replace(tzinfo=timezone.utc)
        local_timezone = pytz.timezone(timezone_name)
        local_dt = utc_dt.astimezone(local_timezone)
    return local_dt.strftime(format)


@app.route("/")
def index():
    """Render the home page"""
    return render_template("index.html", page_name="index")


@app.route("/process", methods=["POST"])
def process():
    """Process a new video file or stream"""
    try:
        # Get source type from form
        source_type = request.form.get("sourceType")

        # Get custom job title if provided
        job_title = request.form.get("jobTitle", "").strip()

        # Generate a unique job ID
        job_id = str(uuid.uuid4())

        if source_type == SourceType.FILE:
            # Handle video file upload
            video_file = request.files.get("video")

            if not video_file or video_file.filename == "":
```

```
                flash("No video file selected", "error")
                return redirect(url_for("index"))

            # Save the uploaded file
            file_path = video_service.save_video(video_file)

            if not file_path:
                flash(
                    f"Invalid video format. Please upload {', '.join(FileFormats.VIDEO_FORMATS)}
files.",
                    "error",
                )
                return redirect(url_for("index"))

            # Set name to custom title or fall back to filename
            name = job_title if job_title else os.path.basename(file_path)

            # Create a new VideoJob instance
            new_job = VideoJob(
                id=job_id,
                name=name,
                source_type=SourceType.FILE,
                source_path=file_path,
                status=JobStatus.QUEUED,
            )

            # Save job to database
            db.session.add(new_job)
            db.session.commit()

            # Start processing
            processing_service.process_video(job_id, file_path)

            # Redirect to result page with job ID
            flash("Video processing started", "success")
            return redirect(url_for("results"))

        elif source_type == SourceType.STREAM:
            # Handle stream URL
            stream_url = request.form.get("streamUrl")

            if not stream_url:
                flash("No stream URL provided", "error")
                return redirect(url_for("index"))

            # Validate stream URL
            if not stream_service.validate_stream(stream_url):
                flash("Invalid stream URL format", "error")
                return redirect(url_for("index"))
```

```
            # Get stream FPS setting
            try:
                stream_fps = int(request.form.get("streamFps", 5))
            except ValueError:
                stream_fps = 5  # Default to 5 FPS if invalid value

            # Set name to custom title or fall back to stream URL
            name = job_title if job_title else stream_url

            # Create a new VideoJob instance for stream
            new_job = VideoJob(
                id=job_id,
                name=name,
                source_type=SourceType.STREAM,
                source_path=stream_url,
                status=JobStatus.QUEUED,
                fps=stream_fps,
            )

            # Save job to database
            db.session.add(new_job)
            db.session.commit()

            # Start processing
            processing_service.process_stream(job_id, stream_url, stream_fps)

            # Redirect to result page with job ID
            flash("Stream processing started", "success")
            return redirect(url_for("results"))

        else:
            flash("Invalid source type", "error")
            return redirect(url_for("index"))

    except Exception as e:
        logger.error(f"Error processing request: {str(e)}", exc_info=True)
        flash(f"An error occurred: {str(e)}", "error")
        return redirect(url_for("index"))


@app.route("/api/stream/<stream_id>/stop", methods=["POST"])
def api_stop_stream(stream_id):
    """API endpoint to stop an active stream"""
    try:
        job = db.session.get(VideoJob, stream_id)
        if not job:
            return jsonify({"error": "Stream not found"}), 404
```

```
        if job.source_type != SourceType.STREAM:
            return jsonify({"error": "Job is not a stream"}), 400

        if job.status != JobStatus.PROCESSING:
            return jsonify({"error": "Stream is not currently processing"}), 400

        success = processing_service.cancel_job(stream_id)
        if success:
            job.status = JobStatus.CANCELLED
            job.end_time = datetime.utcnow()
            db.session.commit()

            stream_service.disconnect_stream(job.source_path)
            return jsonify(
                {
                    "success": True,
                    "message": "Stream stopped successfully",
                    "stream_id": stream_id,
                }
            )
        else:
            return (
                jsonify(
                    {"error": "Failed to stop stream. Stream may already be stopped."}
                ),
                400,
            )

    except Exception as e:
        logger.error(f"Error stopping stream {stream_id}: {str(e)}", exc_info=True)
        return jsonify({"error": str(e)}), 500


@app.route("/api/jobs/<job_id>/complete", methods=["POST"])
def api_complete_job(job_id):
    """API endpoint to gracefully stop and complete a job (particularly for streams)"""
    try:
        # Get the job to ensure it exists
        job = db.session.get(VideoJob, job_id)
        if not job:
            return jsonify({"success": False, "error": "Job not found"}), 404

        # Log the request for debugging
        logger.info(
            f"Graceful completion request received for job {job_id} (type: {job.source_type},
status: {job.status})"
        )

        # Check if this is a stream or a regular job
```

```
if job.source_type == SourceType.STREAM:
    # This is a stream, use the stream service to gracefully stop it
    success, error_message = stream_service.gracefully_stop_stream(job_id)

    if success:
        return jsonify(
            {
                "success": True,
                "message": "Stream gracefully completed",
                "job_id": job_id,
            }
        )
    else:
        return (
            jsonify(
                {
                    "success": False,
                    "error": error_message
                    or "Failed to gracefully complete stream",
                }
            ),
            400,
        )
else:
    # For non-stream jobs, attempt to gracefully stop processing
    if job.status == JobStatus.PROCESSING:
        success = processing_service.gracefully_stop_job(job_id)

        if success:
            return jsonify(
                {
                    "success": True,
                    "message": "Job gracefully completed",
                    "job_id": job_id,
                }
            )
        else:
            return (
                jsonify(
                    {
                        "success": False,
                        "error": "Failed to gracefully complete job",
                    }
                ),
                400,
            )
    else:
        # For jobs not in processing state, just return the current status
        return jsonify(
```

```
                    {
                        "success": True,
                        "message": f"Job is in {job.status} state, no action taken",
                        "job_id": job_id,
                        "status": job.status,
                    }
                )

    except Exception as e:
        logger.error(f"Error completing job/stream {job_id}: {str(e)}", exc_info=True)
        return jsonify({"success": False, "error": str(e)}), 500


@app.route("/api/jobs/<job_id>/start", methods=["POST"])
def api_start_job(job_id):
    """API endpoint to manually start processing a job"""
    try:
        job = db.session.get(VideoJob, job_id)

        if not job:
            return jsonify({"error": "Job not found"}), 404

        # Only allow starting jobs that are in startable statuses
        if job.status not in JobStatus.STARTABLE_STATUSES:
            return (
                jsonify(
                    {
                        "error": f"Cannot start job with status '{job.status}'. Only {',
'.join(JobStatus.STARTABLE_STATUSES)} jobs can be started."
                    }
                ),
                400,
            )

        # Check if the source file/stream exists
        if job.source_type == SourceType.FILE and not os.path.exists(job.source_path):
            return jsonify({"error": "Source file not found"}), 400

        # Reset job status to queued
        job.status = JobStatus.QUEUED
        job.error_message = None
        job.progress = 0
        job.start_time = datetime.utcnow()
        job.end_time = None
        db.session.commit()

        # Start processing based on source type
        if job.source_type == SourceType.FILE:
            processing_service.process_video(job.id, job.source_path)
```

```
        elif job.source_type == SourceType.STREAM:
            processing_service.process_stream(job.id, job.source_path, job.fps or 5)

        return jsonify(
            {"success": True, "message": "Job processing started", "job_id": job_id}
        )

    except Exception as e:
        logger.error(f"Error starting job {job_id}: {str(e)}", exc_info=True)
        return jsonify({"error": str(e)}), 500


@app.route("/results")
def results():
    """Display results page with all processing jobs"""
    try:
        # Get pagination parameters
        page = request.args.get("page", 1, type=int)
        status_filter = request.args.get("status")
        source_filter = request.args.get("source")

        # Get paginated results
        pagination = result_service.get_all_results(
            page=page, per_page=10, status=status_filter, source_type=source_filter
        )

        return render_template(
            "results.html",
            pagination=pagination,
            status_filter=status_filter,
            source_filter=source_filter,
            page_name="results",
        )
    except Exception as e:
        logger.error(f"Error loading results page: {str(e)}", exc_info=True)
        flash(f"An error occurred: {str(e)}", "error")
        return redirect(url_for("index"))


@app.route("/api/jobs", methods=["GET"])
def api_get_jobs():
    """API endpoint to get all jobs with optional filtering"""
    try:
        status_filter = request.args.get("status")
        source_filter = request.args.get("source_type")
        page = request.args.get("page", 1, type=int)
        per_page = request.args.get("per_page", 10, type=int)

        query = VideoJob.query
```

```
        if status_filter:
            query = query.filter(VideoJob.status == status_filter)
        if source_filter:
            query = query.filter(VideoJob.source_type == source_filter)
        query = query.order_by(VideoJob.start_time.desc())
        jobs = query.all()

        serialized_jobs = []
        for job in jobs:
            serialized_jobs.append(
                {
                    "id": job.id,
                    "name": job.name,
                    "source_type": job.source_type,
                    "source_path": job.source_path,
                    "status": job.status,
                    "start_time": (
                        job.start_time.isoformat() if job.start_time else None
                    ),
                    "end_time": job.end_time.isoformat() if job.end_time else None,
                    "progress": job.progress,
                    "fps": job.fps,
                    "duration": job.duration,
                    "total_frames": job.total_frames,
                    "total_detections": job.total_detections,
                }
            )

        return jsonify(serialized_jobs)

    except Exception as e:
        logger.error(f"Error getting jobs: {str(e)}", exc_info=True)
        return jsonify({"error": str(e)}), 500


@app.route("/result/<job_id>")
def result_detail(job_id):
    """Display detailed results for a specific job"""
    try:
        # Get job details
        job_details = result_service.get_result_details(job_id)

        if not job_details:
            flash("Job not found", "error")
            return redirect(url_for("results"))

        # Get statistics
        statistics = result_service.get_job_statistics(job_id)
```

```
        # Prepare data for template
        return render_template(
            "result_detail.html",
            job=job_details["job"],
            frames=job_details["frames"],
            detections_by_frame=job_details["detections_by_frame"],
            statistics=statistics,
            page_name="result_detail",
        )
    except Exception as e:
        logger.error(
            f"Error loading result details for job {job_id}: {str(e)}", exc_info=True
        )
        flash(f"An error occurred: {str(e)}", "error")
        return redirect(url_for("results"))


@app.route("/api/jobs/<job_id>", methods=["GET"])
def api_get_job_details(job_id):
    """API endpoint to get detailed information about a specific job"""
    try:
        # Get job details
        job = db.session.get(VideoJob, job_id)

        if not job:
            return jsonify({"error": "Job not found"}), 404

        # Get job details using job service or result service
        if hasattr(job_service, "get_job_details"):
            result_data = job_service.get_job_details(job_id)
        else:
            result_data = result_service.get_result_details(job_id)

        if not result_data:
            # If result_data is None, create a minimal response
            return jsonify(
                {
                    "job": {
                        "id": job.id,
                        "name": job.name,
                        "source_type": job.source_type,
                        "source_path": job.source_path,
                        "status": job.status,
                        "start_time": (
                            job.start_time.isoformat() if job.start_time else None
                        ),
                        "end_time": job.end_time.isoformat() if job.end_time else None,
                        "fps": job.fps,
                        "duration": job.duration,
```

```
                "total_frames": job.total_frames,
                "total_detections": job.total_detections,
                "avg_confidence": job.avg_confidence,
                "most_common_class": job.most_common_class,
            },
            "frames": {
                "items": [],
                "total": 0,
                "page": 1,
                "pages": 0,
                "has_next": False,
                "has_prev": False,
            },
            "detections_by_frame": {},
        }
    )

# Process detections_by_frame to ensure consistent structure
processed_detections_by_frame = {}

if "detections_by_frame" in result_data:
    for frame_id, detections in result_data["detections_by_frame"].items():
        # Ensure each detection has the expected fields
        processed_detections = []
        for detection in detections:
            processed_detection = {
                "class_name": detection.get("class_name", "unknown"),
                "confidence": detection.get("confidence", 0.0),
                "x_min": detection.get("x_min", 0),
                "y_min": detection.get("y_min", 0),
                "x_max": detection.get("x_max", 0),
                "y_max": detection.get("y_max", 0),
                "area": detection.get("area", 0),
            }
            processed_detections.append(processed_detection)

        processed_detections_by_frame[frame_id] = processed_detections

# Create a serializable version of the response
serializable_response = {
    "job": {
        "id": job.id,
        "name": job.name,
        "source_type": job.source_type,
        "source_path": job.source_path,
        "status": job.status,
        "start_time": job.start_time.isoformat() if job.start_time else None,
        "end_time": job.end_time.isoformat() if job.end_time else None,
        "fps": job.fps,
```

```
            "duration": job.duration,
            "total_frames": job.total_frames,
            "total_detections": job.total_detections,
            "avg_confidence": job.avg_confidence,
            "most_common_class": job.most_common_class,
        },
        "frames": {
            "items": (
                result_data["frames"].items
                if hasattr(result_data["frames"], "items")
                else []
            ),
            "total": (
                result_data["frames"].total
                if hasattr(result_data["frames"], "total")
                else 0
            ),
            "page": (
                result_data["frames"].page
                if hasattr(result_data["frames"], "page")
                else 1
            ),
            "pages": (
                result_data["frames"].pages
                if hasattr(result_data["frames"], "pages")
                else 0
            ),
            "has_next": (
                result_data["frames"].has_next
                if hasattr(result_data["frames"], "has_next")
                else False
            ),
            "has_prev": (
                result_data["frames"].has_prev
                if hasattr(result_data["frames"], "has_prev")
                else False
            ),
        },
        "detections_by_frame": processed_detections_by_frame,
    }

    # Return JSON response
    return jsonify(serializable_response)

except Exception as e:
    logger.error(
        f"Error getting job details for job {job_id}: {str(e)}", exc_info=True
    )
    return jsonify({"error": str(e)}), 500
```

```
@app.route("/api/jobs/<job_id>/video", methods=["GET"])
def api_get_job_video(job_id):
    """API endpoint to serve the video for a job (original or processed)"""
    try:
        job = db.session.get(VideoJob, job_id)

        if not job:
            return jsonify({"error": "Job not found"}), 404

        # Get the type of video requested
        video_type = request.args.get("type", "result")  # 'result' or 'original'
        quality = request.args.get("quality", "preview")  # 'preview' or 'full'

        # For completed jobs with processed results
        if video_type == "result" and job.status == JobStatus.COMPLETED:
            # Check if preview is requested and available
            if (
                quality == "preview"
                and job.preview_video_path
                and os.path.exists(job.preview_video_path)
            ):
                logger.debug(f"Serving preview video: {job.preview_video_path}")
                return send_file(job.preview_video_path, mimetype="video/mp4")
            # Check if full quality is available
            elif job.result_video_path and os.path.exists(job.result_video_path):
                logger.debug(f"Serving full quality video: {job.result_video_path}")
                return send_file(job.result_video_path, mimetype="video/mp4")

        # Fall back to original video
        if (
            job.source_type == SourceType.FILE
            and job.source_path
            and os.path.exists(job.source_path)
        ):
            logger.debug(f"Serving original video: {job.source_path}")
            return send_file(job.source_path, mimetype="video/mp4")
        else:
            logger.warning(f"Video file not available for job {job_id}")
            return jsonify({"error": "Video file not available"}), 404

    except Exception as e:
        logger.error(f"Error serving video for job {job_id}: {str(e)}", exc_info=True)
        return jsonify({"error": f"Failed to serve video: {str(e)}"}), 500


@app.route("/api/jobs/<job_id>/download", methods=["GET"])
def api_download_job_video(job_id):
```

```
"""API endpoint to download the video for a job"""
try:
    job = db.session.get(VideoJob, job_id)

    if not job:
        return jsonify({"error": "Job not found"}), 404

    # Get the quality parameter
    quality = request.args.get(
        "quality", "full"
    )  # Default to full quality for downloads
    video_type = request.args.get("type", "result")  # 'result' or 'original'

    # For completed jobs, offer processed video for download
    if job.status == JobStatus.COMPLETED and video_type == "result":
        # Check if full quality is requested and available
        if (
            quality == "full"
            and job.result_video_path
            and os.path.exists(job.result_video_path)
        ):
            # Return the full quality processed video
            logger.debug(
                f"Sending full quality processed video: {job.result_video_path}"
            )
            return send_file(
                job.result_video_path,
                as_attachment=True,
                download_name=f"{job.name}_processed.mp4",
            )
        # Check if preview is requested and available
        elif (
            quality == "preview"
            and job.preview_video_path
            and os.path.exists(job.preview_video_path)
        ):
            # Return the preview quality video
            logger.debug(
                f"Sending preview quality processed video: {job.preview_video_path}"
            )
            return send_file(
                job.preview_video_path,
                as_attachment=True,
                download_name=f"{job.name}_processed_preview.mp4",
            )

    # Fall back to original video
    if (
        job.source_type == SourceType.FILE
```

```
            and job.source_path
            and os.path.exists(job.source_path)
        ):
            # Return the original video file
            logger.debug(f"Sending original video file: {job.source_path}")
            return send_file(
                job.source_path,
                as_attachment=True,
                download_name=job.name or f"video_{job_id}.mp4",
            )
        else:
            logger.warning(f"Video file not available for job {job_id}")
            return jsonify({"error": "Video file not available"}), 404

    except Exception as e:
        logger.error(
            f"Error downloading video for job {job_id}: {str(e)}", exc_info=True
        )
        return jsonify({"error": f"Failed to download video: {str(e)}"}), 500


@app.route("/api/jobs/<job_id>/export/<format>", methods=["GET"])
def api_export_job_data(job_id, format):
    """API endpoint to export job data in various formats"""
    try:
        job = db.session.get(VideoJob, job_id)

        if not job:
            return jsonify({"error": "Job not found"}), 404

        # Get request parameters
        include_metadata = (
            request.args.get("include_metadata", "true").lower() == "true"
        )
        include_bboxes = request.args.get("include_bboxes", "true").lower() == "true"
        include_confidence = (
            request.args.get("include_confidence", "true").lower() == "true"
        )

        # Get job details
        job_details = result_service.get_result_details(job_id)

        # Prepare export data
        export_data = {}

        # Add metadata if requested
        if include_metadata:
            export_data["metadata"] = {
                "id": job.id,
```

```
        "name": job.name,
        "source_type": job.source_type,
        "source_path": job.source_path,
        "status": job.status,
        "start_time": job.start_time.isoformat() if job.start_time else None,
        "end_time": job.end_time.isoformat() if job.end_time else None,
        "fps": job.fps,
        "duration": job.duration,
        "resolution": (
            f"{job.resolution_width}x{job.resolution_height}"
            if job.resolution_width and job.resolution_height
            else None
        ),
        "total_frames": job.total_frames,
        "total_detections": job.total_detections,
    }

# Prepare detections data
detections_list = []
if job_details and "frames" in job_details:
    for frame in job_details["frames"].items:
        frame_id = frame["id"]
        timestamp = frame["timestamp"]

        # Skip if no detections for this frame
        if frame_id not in job_details.get("detections_by_frame", {}):
            continue

        # Add each detection
        for detection in job_details["detections_by_frame"][frame_id]:
            detection_data = {
                "frame": frame_id,
                "timestamp": timestamp,
                "class": detection.get("class_name", ""),
            }

            # Add confidence if requested
            if include_confidence:
                detection_data["confidence"] = detection.get("confidence", 0)

            # Add bounding box if requested
            if include_bboxes:
                detection_data["x_min"] = detection.get("x_min", 0)
                detection_data["y_min"] = detection.get("y_min", 0)
                detection_data["x_max"] = detection.get("x_max", 0)
                detection_data["y_max"] = detection.get("y_max", 0)

            detections_list.append(detection_data)
```

```
export_data["detections"] = detections_list

# Generate response based on requested format
format = format.lower()

if format == ExportFormats.JSON:
    return jsonify(export_data)

elif format == ExportFormats.CSV:
    # Create CSV in a temporary file
    fd, temp_path = tempfile.mkstemp(suffix=".csv")
    with os.fdopen(fd, "w", newline="") as csvfile:
        # Determine fields based on first detection (if any)
        if detections_list:
            fieldnames = detections_list[0].keys()
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()
            writer.writerows(detections_list)
        else:
            # Write empty CSV with basic headers
            writer = csv.writer(csvfile)
            writer.writerow(["frame", "timestamp", "class"])

    # Return the CSV file
    return send_file(
        temp_path,
        as_attachment=True,
        download_name=f"{job.name or 'job_' + job_id}_detections.csv",
        mimetype=ExportFormats.MIME_TYPES[ExportFormats.CSV],
    )

elif format == ExportFormats.XLSX:
    # Create a temporary file for the Excel workbook
    fd, temp_path = tempfile.mkstemp(suffix=".xlsx")
    os.close(fd)  # Close the file descriptor

    # Create a workbook and add worksheets
    workbook = xlsxwriter.Workbook(temp_path)

    # Add metadata worksheet if requested
    if include_metadata:
        metadata_sheet = workbook.add_worksheet("Metadata")

        # Define formats
        header_format = workbook.add_format(
            {
                "bold": True,
                "bg_color": "#4B8BBE",
                "font_color": "white",
```

```
            "border": 1,
        }
    )

    cell_format = workbook.add_format({"border": 1})

    # Add metadata headers
    metadata_sheet.write(0, 0, "Property", header_format)
    metadata_sheet.write(0, 1, "Value", header_format)

    # Set column widths
    metadata_sheet.set_column(0, 0, 20)
    metadata_sheet.set_column(1, 1, 40)

    # Add metadata rows
    row = 1
    for key, value in export_data["metadata"].items():
        metadata_sheet.write(row, 0, key, cell_format)
        metadata_sheet.write(row, 1, str(value), cell_format)
        row += 1

# Add detections worksheet
detections_sheet = workbook.add_worksheet("Detections")

# Define formats
header_format = workbook.add_format(
    {
        "bold": True,
        "bg_color": "#4B8BBE",
        "font_color": "white",
        "border": 1,
    }
)

cell_format = workbook.add_format({"border": 1})

# Determine headers based on options
headers = ["Frame", "Timestamp", "Class"]
if include_confidence:
    headers.append("Confidence")
if include_bboxes:
    headers.extend(["X Min", "Y Min", "X Max", "Y Max"])

# Write headers
for col, header in enumerate(headers):
    detections_sheet.write(0, col, header, header_format)

# Set column widths
detections_sheet.set_column(0, 0, 10)  # Frame
```

```
detections_sheet.set_column(1, 1, 15)  # Timestamp
detections_sheet.set_column(2, 2, 20)  # Class
if include_confidence:
    detections_sheet.set_column(3, 3, 15)  # Confidence

# Add detections
if detections_list:
    row = 1
    for detection in detections_list:
        col = 0

        # Write basic fields
        detections_sheet.write(
            row, col, detection.get("frame", ""), cell_format
        )
        col += 1
        detections_sheet.write(
            row, col, detection.get("timestamp", ""), cell_format
        )
        col += 1
        detections_sheet.write(
            row, col, detection.get("class", ""), cell_format
        )
        col += 1

        # Write confidence if requested
        if include_confidence:
            confidence_value = detection.get("confidence", 0)
            confidence_percent = (
                f"{confidence_value * 100:.2f}%"
                if isinstance(confidence_value, (int, float))
                else confidence_value
            )
            detections_sheet.write(
                row, col, confidence_percent, cell_format
            )
            col += 1

        # Write bounding box if requested
        if include_bboxes:
            detections_sheet.write(
                row, col, detection.get("x_min", 0), cell_format
            )
            col += 1
            detections_sheet.write(
                row, col, detection.get("y_min", 0), cell_format
            )
            col += 1
            detections_sheet.write(
```

```
                            row, col, detection.get("x_max", 0), cell_format
                        )
                        col += 1
                        detections_sheet.write(
                            row, col, detection.get("y_max", 0), cell_format
                        )

                    row += 1

            # Close the workbook
            workbook.close()

            # Return the Excel file
            return send_file(
                temp_path,
                as_attachment=True,
                download_name=f"{job.name or 'job_' + job_id}_detections.xlsx",
                mimetype=ExportFormats.MIME_TYPES[ExportFormats.XLSX],
            )

        elif format == ExportFormats.PDF:
            # For simplicity, we'll just return JSON for now
            # In a real implementation, you'd use a library like ReportLab or WeasyPrint
            return jsonify({"error": "PDF format not implemented yet"}), 501

        else:
            return jsonify({"error": f"Unsupported export format: {format}"}), 400

    except Exception as e:
        logger.error(
            f"Error exporting job data for job {job_id}: {str(e)}", exc_info=True
        )
        return jsonify({"error": f"Failed to export data: {str(e)}"}), 500


@app.route("/settings", methods=["GET", "POST"])
def settings():
    """Handle application settings page"""
    try:
        if request.method == "POST":
            # Process form submission
            for key in request.form:
                if key.startswith("setting_"):
                    setting_key = key[8:]  # Remove 'setting_' prefix
                    setting_value = request.form[key]

                    # Update setting in database
                    setting = Settings.query.filter_by(key=setting_key).first()
                    if setting:
```

```
                        setting.value = setting_value
                    else:
                        # Create setting if it doesn't exist
                        new_setting = Settings(key=setting_key, value=setting_value)
                        db.session.add(new_setting)

            # Commit all changes to database
            db.session.commit()
            flash("Settings updated successfully", "success")
            return redirect(url_for("settings"))

        # GET request: retrieve and display current settings
        all_settings = Settings.query.all()

        # Convert to dict for easier template access
        settings_dict = {s.key: s.value for s in all_settings}

        # Load default settings if needed
        if not all_settings:
            settings_dict = {
                "detection_threshold": str(
                    ProcessingSettings.DEFAULT_CONFIDENCE_THRESHOLD
                ),
                "max_processing_jobs": "3",
                "retention_days": "30",
                "notification_email": "",
                "enable_notifications": "false",
            }

        # Get active streams for display
        active_streams = stream_service.get_all_active_streams()

        return render_template(
            "settings.html",
            settings=settings_dict,
            active_streams=active_streams,
            page_name="settings",
        )

    except Exception as e:
        logger.error(f"Error handling settings: {str(e)}", exc_info=True)
        if request.method == "POST":
            return jsonify({"success": False, "error": str(e)})
        flash(f"An error occurred: {str(e)}", "error")
        return redirect(url_for("index"))


@app.route("/api/jobs/<job_id>/cancel", methods=["POST"])
def api_cancel_job(job_id):
```

```
    """API endpoint to cancel a running job or stream"""
    try:
        # Get the job to ensure it exists
        job = db.session.get(VideoJob, job_id)
        if not job:
            return jsonify({"success": False, "error": "Job not found"}), 404

        # Log the request for debugging
        logger.info(
            f"Cancel request received for job {job_id} (type: {job.source_type}, status:
{job.status})"
        )

        if job.source_type == SourceType.STREAM:
            success, error_message = stream_service.stop_stream(job_id)

            if success:
                updated_job = db.session.get(VideoJob, job_id)
                if updated_job.status != JobStatus.CANCELLED:
                    updated_job.status = JobStatus.CANCELLED
                    updated_job.end_time = datetime.now(timezone.utc)
                    db.session.commit()
                    logger.info(
                        f"Force updated job status to cancelled for job {job_id}"
                    )

                return jsonify(
                    {
                        "success": True,
                        "message": "Stream stopped successfully",
                        "job_id": job_id,
                    }
                )
            else:
                return (
                    jsonify(
                        {
                            "success": False,
                            "error": error_message or "Failed to stop stream",
                        }
                    ),
                    400,
                )
        else:
            success = processing_service.cancel_job(job_id)
            if success:
                return jsonify(
                    {
                        "success": True,
```

```
                        "message": "Job cancelled successfully",
                        "job_id": job_id,
                    }
                )
            else:
                try:
                    if job.status in [JobStatus.QUEUED, JobStatus.PROCESSING]:
                        job.status = JobStatus.CANCELLED
                        job.end_time = datetime.now(timezone.utc)
                        db.session.commit()
                        logger.info(
                            f"Force cancelled job {job_id} via direct database update"
                        )

                        return jsonify(
                            {
                                "success": True,
                                "message": "Job cancelled by direct database update",
                                "job_id": job_id,
                            }
                        )
                    else:
                        return (
                            jsonify(
                                {
                                    "success": False,
                                    "error": f"Job is in {job.status} state which cannot be
cancelled",
                                }
                            ),
                            400,
                        )
                except Exception as e:
                    logger.error(f"Error during direct job cancellation: {str(e)}")
                    return (
                        jsonify(
                            {
                                "success": False,
                                "error": f"Failed to cancel job: {str(e)}",
                            }
                        ),
                        500,
                    )

    except Exception as e:
        logger.error(f"Error cancelling job/stream {job_id}: {str(e)}", exc_info=True)
        return jsonify({"success": False, "error": str(e)}), 500
```

```
@app.route("/api/stream-info", methods=["GET"])
def api_stream_info():
    """API endpoint to get information about an active stream"""
    try:
        stream_id = request.args.get("id")
        if not stream_id:
            return jsonify({"error": "Stream ID is required"}), 400

        # Get job data
        job = db.session.get(VideoJob, stream_id)
        if not job:
            return jsonify({"error": "Stream not found"}), 404

        # Check if job is a stream
        if job.source_type != SourceType.STREAM:
            return jsonify({"error": "Job is not a stream"}), 400

        # Get additional stream info
        stream_info = stream_service.get_stream_info(job.source_path)
        logger.info("Stream info", stream_info)
        # Combine job data with stream info
        result = {
            "id": job.id,
            "status": job.status,
            "url": job.source_path,
            "active": job.status == JobStatus.PROCESSING,
            "startTime": job.start_time.isoformat() if job.start_time else None,
            "framesProcessed": job.total_frames or 0,
            "duration": job.duration or 0,
        }

        # Get recent detections
        recent_detections = []
        try:
            job_details = result_service.get_result_details(
                stream_id, page=1, per_page=5
            )
            if job_details and "detections_by_frame" in job_details:
                # Flatten recent detections
                for frame_id, detections in job_details["detections_by_frame"].items():
                    for detection in detections:
                        recent_detections.append(
                            {
                                "frame": frame_id,
                                "class": detection.get("class_name", "unknown"),
                                "confidence": detection.get("confidence", 0),
                                "timestamp": datetime.now().isoformat(),  # Placeholder
                            }
                        )
```

```
                # Sort by most recent (assuming highest frame_id is most recent)
                recent_detections.sort(key=lambda x: x["frame"], reverse=True)
                # Limit to 10 most recent
                recent_detections = recent_detections[:10]
        except Exception as e:
            logger.warning(f"Error getting recent detections: {str(e)}")

        result["recentDetections"] = recent_detections

        return jsonify(result)
    except Exception as e:
        logger.error(f"Error getting stream info: {str(e)}", exc_info=True)
        return jsonify({"error": str(e)}), 500


@app.route("/api/jobs/<job_id>", methods=["DELETE"])
def api_delete_job(job_id):
    """API endpoint to delete a job and all associated files"""
    try:
        # Get the job to ensure it exists
        job = db.session.get(VideoJob, job_id)
        if not job:
            return jsonify({"error": "Job not found"}), 404

        # Cancel the job if it's running
        if job.status == JobStatus.PROCESSING:
            processing_service.cancel_job(job_id)

        # Use the service to delete the job and its files
        success = job_service.delete_job(job_id)

        if success:
            return jsonify(
                {
                    "success": True,
                    "message": "Job deleted successfully",
                    "job_id": job_id,
                }
            )
        else:
            return jsonify({"error": "Failed to delete job"}), 500

    except Exception as e:
        logger.error(f"Error deleting job {job_id}: {str(e)}", exc_info=True)
        return jsonify({"error": str(e)}), 500


def initialize_database():
```

```
"""Initialize database tables and default settings"""
with app.app_context():
    try:
        # Check if tables exist
        inspector = db.inspect(db.engine)
        existing_tables = inspector.get_table_names()

        if not existing_tables or "video_jobs" not in existing_tables:
            logger.info("Database tables not found. Creating tables...")
            db.create_all()
            logger.info("Database tables created successfully.")

            # Initialize default settings
            if (
                "settings" in inspector.get_table_names()
                and not Settings.query.first()
            ):
                default_settings = [
                    Settings(
                        key="detection_threshold",
                        value=str(ProcessingSettings.DEFAULT_CONFIDENCE_THRESHOLD),
                    ),
                    Settings(key="max_processing_jobs", value="3"),
                    Settings(key="retention_days", value="30"),
                    Settings(key="notification_email", value=""),
                    Settings(key="enable_notifications", value="false"),
                    Settings(key="default_export_format", value=ExportFormats.JSON),
                    Settings(
                        key="stream_fps",
                        value=str(ProcessingSettings.DEFAULT_STREAM_FPS),
                    ),
                    Settings(
                        key="max_reconnect_attempts",
                        value=str(ProcessingSettings.MAX_RECONNECT_ATTEMPTS),
                    ),
                ]
                db.session.add_all(default_settings)
                db.session.commit()
                logger.info("Default settings initialized.")
        else:
            logger.info("Database tables already exist.")

            # Check for and add missing columns
            columns = [col["name"] for col in inspector.get_columns("video_jobs")]
            missing_columns = []

            if "result_video_path" not in columns:
                missing_columns.append("result_video_path VARCHAR(255)")
```

```
                if "preview_video_path" not in columns:
                    missing_columns.append("preview_video_path VARCHAR(255)")

                # Add all missing columns at once
                if missing_columns:
                    logger.info(
                        f"Adding missing columns to video_jobs table: {missing_columns}"
                    )

                    for col_def in missing_columns:
                        col_name = col_def.split(" ")[0]
                        try:
                            # Use raw SQL for simplicity
                            db.session.execute(
                                f"ALTER TABLE video_jobs ADD COLUMN {col_def}"
                            )
                            logger.info(f"Column {col_name} added successfully.")
                        except Exception as e:
                            # Column might have been added in a concurrent process
                            logger.warning(f"Error adding column {col_name}: {str(e)}")

                    db.session.commit()

            # Create storage directories if they don't exist
            storage_service._ensure_directories_exist()

        except Exception as e:
            logger.error(f"Error initializing database: {str(e)}", exc_info=True)
            raise


if __name__ == "__main__":
    initialize_database()
    app.run(debug=True)
```

Додаток Г – Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

“Розробка програмного модуля автоматичного визначення характеристик об’єктів на динамічних зображеннях із застосуванням нейромереж”

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

“Розробка програмного модуля автоматичного визначення характеристик об’єктів на динамічних зображеннях із застосуванням нейромереж”

СТУДЕНТ: ст. гр. 1ПІ-21Б
Хитрич С.С.
КЕРІВНИК: к.т.н., доцент
Рейда О.М.

Вінниця 2025

Рисунок Г.1 – Титульний слайд

Актуальність, мета і завдання дослідження

Актуальність даної розробки зумовлена необхідністю розробки програмного модуля автоматичного визначення характеристик об’єктів на динамічних зображеннях із застосуванням нейромереж для підвищення ефективності автоматизованого аналізу об'єктів на динамічних зображеннях та потокових даних з можливістю локального розгоратння.

Метою і завданням дослідження є підвищення ефективності обробки та аналізу відеоматеріалів за рахунок використання нейромереж у автоматизованих системах позиціювання об'єктів.

Рисунок Г.2 – Актуальність, мета і завдання дослідження

Предмет та об’єкт дослідження

Предмет дослідження – методи та алгоритми реалізації системи комп’ютерного зору й машинного навчання, що використовуються для аналізу параметрів об'єктів у відеопотоці, а також засоби інтеграції відповідних алгоритмів у програмне забезпечення.

Об’єктом дослідження – процес розробки системи автоматизованого визначення параметрів об’єктів у динамічних зображеннях.

Рисунок Г.3 – Предмет та об’єкт дослідження

Задачі

- розробка модуля детекції об’єктів, класифікації, трекінгу, аналізу їх швидкості, прискорення та розмірів;
- розробка графічного інтерфейсу для роботи із програмним забезпеченням, переглядом та експортом результатів;
- розробка модуля візуалізації результатів;
- модифікувати метод позиціювання об'єктів у потоці даних для підвищення точності визначення кінематичних параметрів;
- розробка алгоритму збереження\експорту результатів;
- інтеграція модулів та алгоритмів у єдиний застосунок для визначення параметрів на динамічних зображеннях.

Рисунок Г.4 – Задачі

Наукова новизна

Модифіковано метод позиціювання об'єктів у потоці даних, що на відміну від існуючих, використовує «YOLO» модель нейромереж для підвищення точності визначення кінематичних параметрів на основі трекінгу класифікованих об'єктів.

Рисунок Г.5 – Наукова новизна

Практичне значення отриманих результатів

Полягає у можливості використання програмного забезпечення в системах відеоспостереження, аналізу руху транспорту, автоматизованому контролі виробничих процесів та медичній діагностиці на основі відеоаналізу.

Рисунок Г.6 – Практичне значення отриманих результатів

Аналоги

Amazon Rekognition — хмарний сервіс від AWS, який дозволяє аналізувати зображення та відео для виявлення об'єктів, осіб, тексту, активностей, а також відстеження осіб у реальному часі. Підтримує пошук відповідностей облич та модерацію контенту.

Google Cloud Vision API — інструмент від Google, що забезпечує аналіз зображень із визначенням об'єктів, тексту (OCR), емоцій на обличчях, логотипів, а також класифікацією контенту за безпечністю. Працює як REST API з гнучкою інтеграцією.

Microsoft Azure Computer Vision — частина Azure Cognitive Services, яка надає можливості аналізу візуального контенту: опис сцени, виявлення об'єктів, зчитування тексту (включно з рукописним), генерація тегів і створення підписів до зображень.

Рисунок Г.7 – Аналоги

Порівняльна характеристика аналогів

Критерій	Amazon Rekognition	Google Cloud Vision API	Microsoft Azure Computer Vision	Власна розробка
Робота в реальному часі	0	0	0	1
Можливість локального розгортання	0	0	0	1
Гнучкість налаштувань	0	0	1	1
Підтримка кастомних моделей	0	0	1	1
Простота інтеграції через API	1	1	0	1
Підтримка роботи з великими наборами даних	1	0	1	1
Можливість масштабування	0	1	1	1
Сумарний коефіцієнт	2	2	4	7

Рисунок Г.8 – Порівняльна характеристика аналогів

Методи та засоби розробки

Для реалізації застосунку було обрано мову програмування Python 3+ у поєднанні з бібліотекою OpenCV, яка є однією з найпопулярніших і найшвидших для задач комп'ютерного зору.

Основу системи детекції об'єктів склало сімейство моделей YOLO, зокрема версії v5 та v8. YOLOv5 була використана завдяки своїй стабільності та широкій підтримці в спільноті, тоді як YOLOv8 забезпечила вищу точність і продуктивність завдяки новітнім архітектурним рішенням. Обидві моделі навчено на датасеті COCO, що є стандартом де-факто для задач багатокласової детекції об'єктів.

Для забезпечення надійного трекінгу та розрахунку кінетичних параметрів об'єктів було використано алгоритм ByteTrack, який ефективно інтегрується з вихідними детекціями YOLO. Розробку виконано у середовищі Visual Studio Code.

Рисунок Г.9 – Методи та засоби розробки

OpenCV (Open Source Computer Vision Library) — це відкрита бібліотека для комп'ютерного зору, обробки зображень та машинного навчання.

Основні характеристики

- Підтримка багатьох мов програмування (Python, C++, Java).
- Містить понад 2500 оптимізованих алгоритмів для
- обробки зображень (фільтрація, сегментація, перетворення),
- розпізнавання об'єктів і облич,
- трекінгу, виявлення руху,
- калібрування камер та 3D-реконструкції.
- Інтегрується з апаратним прискоренням (GPU, OpenCL).
- Активно підтримується і має широку спільноту

Переваги в проєкті:

- Простота інтеграції з Python.
- Швидкість обробки зображень у реальному часі
- Можливість попередньої обробки перед подачею на модель (наприклад, нормалізація, зміна розміру, виділення ROI).

Рисунок Г.10 – OpenCV

YOLO (You Only Look Once) — це архітектура для швидкої та точної детекції об'єктів у зображеннях або відео.

Ключові особливості:

- **Одноетапний підхід** — вся детекція виконується за одну інференцію (на відміну від двоетапних, як Faster R-CNN).
- Висока швидкість, що дозволяє використовувати YOLO в реальному часі.
- Вихід включає координати об'єктів, клас і ймовірність.

Застосування в проєкті:

- YOLOv5 – стабільна модель великою кількістю готових вагів.
- YOLOv8 — більш нова модель, використовується за замовчуванням через кращу продуктивність.

Рисунок Г.11 – Yolo

Діаграма діяльності застосунку

Користувач може обрати обробку відеофайлу або потоку: у першому випадку — завантажує файл і задає параметри, у другому — вводить адресу потоку та налаштування.

Після цього система виконує детекцію об'єктів, аналізує результати й надає доступ до перегляду відео, експорту (JSON, CSV, XLSX) та статистичної візуалізації.

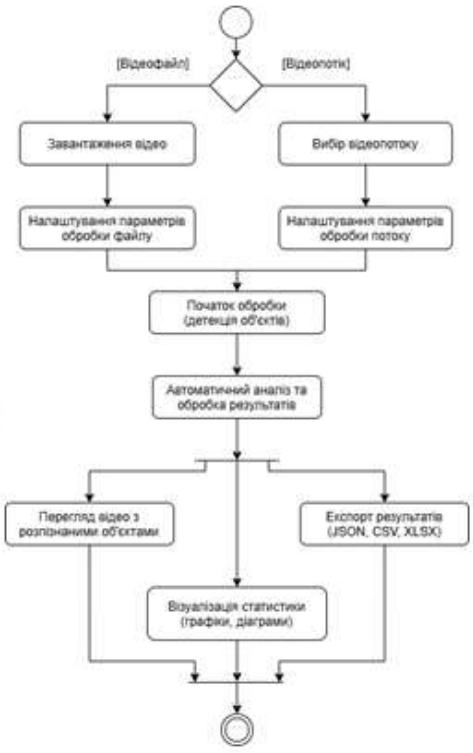


Рисунок Г.12 – Діаграма діяльності застосунку

Структурна діаграма залежностей застосунку

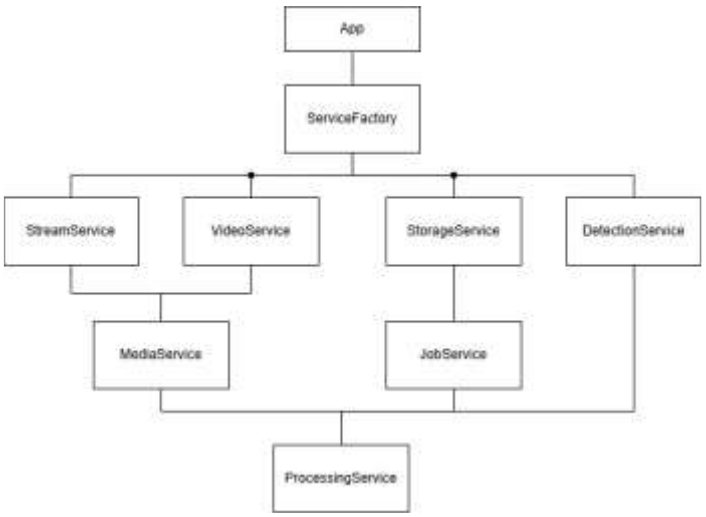


Рисунок Г.13 – Структурна діаграма залежностей застосунку

Блок схема алгоритму аналізу виявлених об'єктів

Процес починається з завантаження параметрів моделі: ініціалізується нейромережа, встановлюються пороги впевненості тощо. Далі запускається система трекінгу для відстеження об'єктів між кадрами.

Після цього починається основний цикл: кадр обробляється через препроцесинг (масштабування, нормалізація), потім виконується детекція об'єктів. Результати фільтруються за порогом впевненості.

Для кожного об'єкта система визначає, чи він новий, оновлює або створює ідентифікатор, виконує класифікацію та обчислює траєкторію, швидкість, прискорення. Наприкінці кадру оновлюється статистика.

Після обробки всіх кадрів алгоритм зберігає підсумкові дані та позначає виконання задачі як завершене.

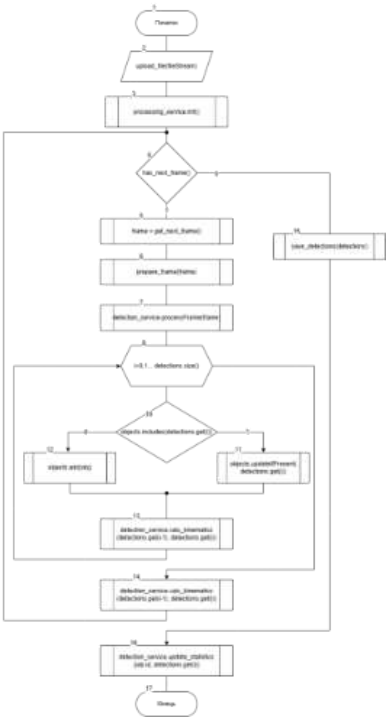


Рисунок Г.14 – Блок схема алгоритму аналізу виявлених об’єктів

Блок схема алгоритму визначення кінетичних параметрів

У системі реалізовано метод для покадрового обчислення кінематичних параметрів виявлених об’єктів: швидкості, прискорення та напрямку руху. Спершу встановлюється відповідність між об’єктами на поточному та попередньому кадрах — за ID або за координатами й класом у разі його відсутності.

Після зіставлення розраховується швидкість як зміщення центру за час, напрямок — як нормалізований вектор, а прискорення — як зміна вектора швидкості. Якщо відповідність не знайдено, параметри ініціалізуються як нульові. Метод інтегрований з обробкою потоків і модулем збереження результатів.

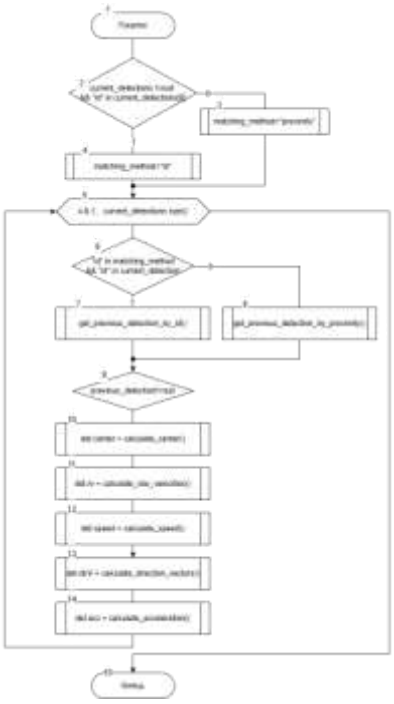


Рисунок Г.15 – Блок схема алгоритму визначення кінетичних параметрів

Рисунки роботи застосунку

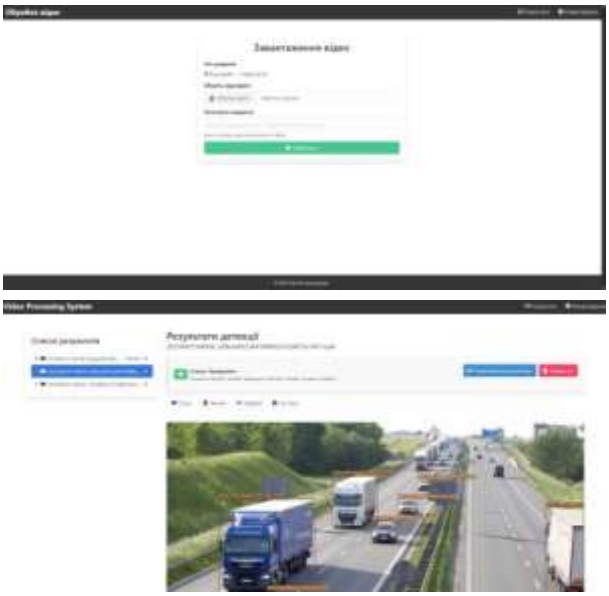


Рисунок Г.16 – Рисунки роботи застосунку

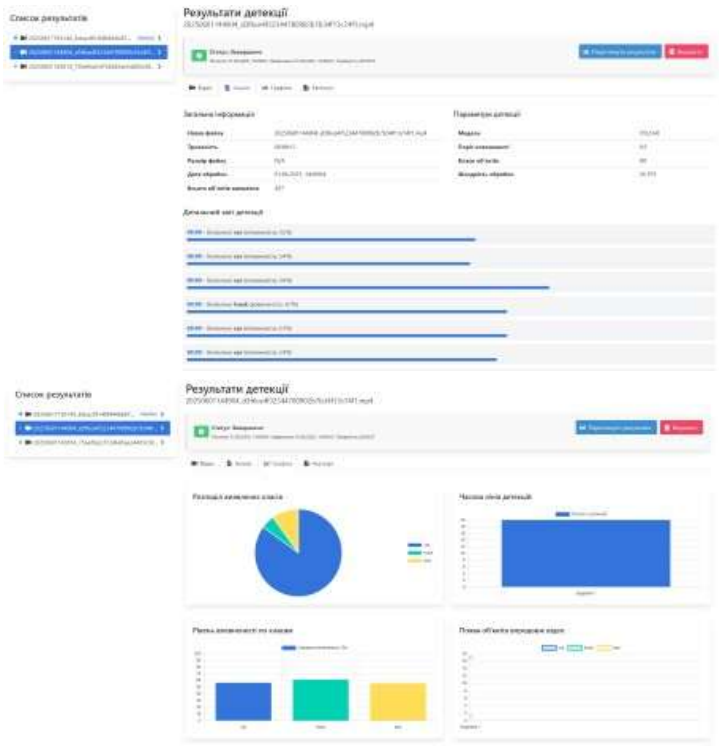


Рисунок Г.17 – Рисунки роботи застосунку

Висновки

- розробка модуля детекції об'єктів, класифікації, трекінгу, аналізу їх швидкості, прискорення та розмірів;
- розробка графічного інтерфейсу для роботи із програмним забезпеченням, переглядом та експортом результатів;
- розробка модуля візуалізації результатів;
- модифікувати метод позиціювання об'єктів у потоці даних для підвищення точності визначення кінематичних параметрів;
- розробка алгоритму збереження\експорту результатів;
- інтеграція модулів та алгоритмів у єдиний застосунок для визначення параметрів на динамічних зображеннях.

Рисунок Г.18 – Рисунки роботи застосунку

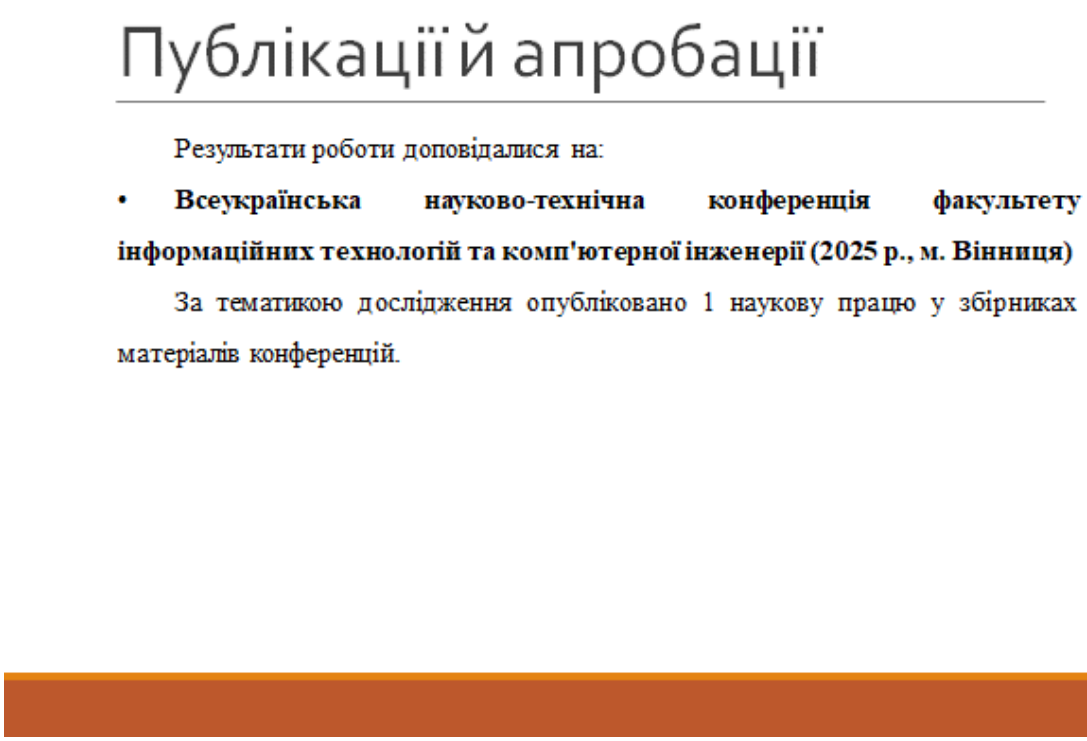


Рисунок Г.19 – Рисунки роботи застосунку

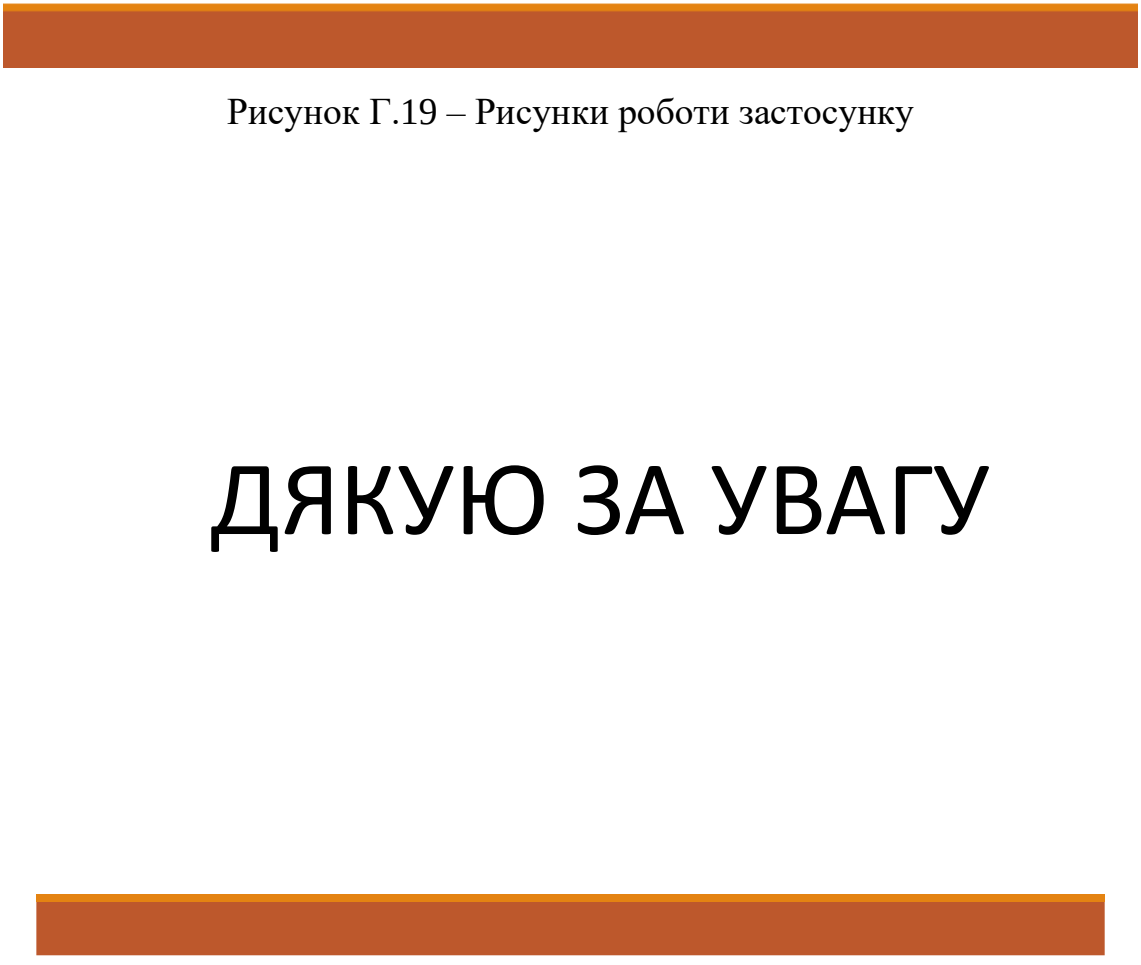


Рисунок Г.20 – Рисунки роботи застосунку