МІНІСТЕРСТВО ВНУТРІШНІХ СПРАВ УКРАЇНИ
ЛЬВІВСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ ВНУТРІШНІХ СПРАВ
Навчально-науковий інститут управління, психології та безпеки
Кафедра інформаційних технологій

КОМПЛЕКСНА СИСТЕМА КЛАСИФІКАЦІЇ ВІЙСЬКОВОЇ ТЕХНІКИ НА ОСНОВІ МЕТОДІВ АНАЛІЗУ ЗОБРАЖЕНЬ

кваліфікаційна робота
здобувача вищої освіти
4 курсу заочної форми навчання
Романа ЛЕМІШКА

Науковий керівник:
Сергій КУТАЄВ

Рецензент:
_____
_____

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

Завідувач кафедри інформаційних технологій
_____**Олег ЗАЧЕК**
(підпис)

Львів
2026

СПИСОК УМОВНИХ СКОРОЧЕНЬ

AFV	Armoured Fighting Vehicle (броньована бойова машина)
APC	Armoured Personnel Carrier (бронетранспортер)
CLI	Command Line Interface (інтерфейс командного рядка)
FPS	Frames Per Second (кількість кадрів за секунду)
GMC	Global Motion Compensation (глобальна компенсація руху камери в трекері)
GUI	Graphical User Interface (графічний інтерфейс користувача)
MEV	Military Engineering Vehicle (інженерна військова машина)
OS	Operating System (операційна система)
API	Application Programming Interface (інтерфейс програмування застосунків)
АРМ	Автоматизоване робоче місце оператора або аналітика
БПЛА	Безпілотний літальний апарат
ГІС	Геоінформаційна система
ІС	Інформаційна система
МВС	Міністерство внутрішніх справ
ДСТУ	Державний стандарт України
ПЗ	Програмне забезпечення
ТТХ	Тактико-технічні характеристики
ПК	Програмний комплекс
СУБД	Система управління базами даних

АНОТАЦІЯ

ЛЕМІШКО Р. Комплексна система класифікації військової техніки на основі методів аналізу зображень. – Рукопис.

Дослідження на здобуття освітнього ступеня «бакалавр» за спеціальністю 126 «Інформаційні системи та технології». – Львівський державний університет внутрішніх справ, МВС України, Львів, 2026.

У роботі розроблено комплексну систему класифікації військової техніки на основі методів комп'ютерного аналізу зображень для підрозділів Національної поліції України. Проведено аналіз існуючих підходів обробки зображень та обґрунтовано вибір технологій. Реалізовано обчислювальний конвеєр аналізу медіапотоків та багатопотоковий графічний інтерфейс користувача. Результатом роботи є автономний програмний прототип системи з можливістю автоматичного розпізнавання, супроводження та класифікації цілей.

Ключові слова: аналіз зображень, комп’ютерний зір, класифікація техніки, YOLO12, BoT-SORT, PySide6, автоматизація, моніторинг.

ABSTRACT

LEMISCHKO R. Comprehensive system of military equipment classification based on image analysis methods. – Manuscript.

Research on the bachelor’s degree in specialty 126 «Information systems and technologies». – Lviv State University of Internal Affairs, MIA of Ukraine, Lviv, 2026.

The work presents a comprehensive system for military equipment classification based on computer image analysis methods for the National Police of Ukraine. Existing image processing approaches were analyzed and the choice of technologies was justified. An image processing pipeline and multi-threaded graphical user interface were implemented. The result is an autonomous software prototype with automatic recognition, tracking, and classification capabilities.

Keywords: image analysis, computer vision, equipment classification, YOLO12, BoT-SORT, PySide6, automation, monitoring.

ЗМІСТ

ВСТУП

Актуальність теми. Сучасний етап розвитку систем автоматизації та військових технологій характеризується стрімким збільшенням обсягів візуальної інформації, що надходить з різноманітних джерел: безпілотних літальних апаратів (БПЛА), стаціонарних комплексів спостереження, оптико-електронних засобів розвідки та відкритих джерел інформації. У цих умовах швидкість, точність та надійність обробки відео- та фотопотоків відіграють вирішальну роль для забезпечення ситуаційної обізнаності та прийняття тактичних рішень.

Традиційний ручний аналіз візуальних даних операторами є малоефективним через високе когнітивне навантаження, ризик виникнення помилок внаслідок людського фактора, а також фізичні обмеження швидкості обробки інформації в реальному часі. Крім того, сучасне поле бою висуває жорсткі вимоги до розпізнавання об’єктів, що діють в умовах маскування, складного рельєфу, мінливого освітлення, задимленості чи несприятливих погодних умов.

Це зумовлює гостру необхідність у створенні комп’ютеризованих комплексів автоматичного розпізнавання цілей (Automated Target Recognition, ATR). Найбільш перспективним напрямом вирішення цієї задачі є застосування методів глибокого навчання (Deep Learning), зокрема згорткових нейронних мереж та архітектур з механізмами уваги (Attention Mechanisms). Поява у 2025–2026 роках архітектури нового покоління YOLO12 від компанії Ultralytics відкриває нові можливості для детекції об’єктів завдяки суттєвому покращенню балансу між обчислювальною складністю та точністю розпізнавання. Поєднання такого детектора з сучасними алгоритмами багатооб’єктного трекінгу (Multi-Object Tracking), такими як BoT-SORT, дозволяє не лише класифікувати техніку, але й стабільно відстежувати її

переміщення, компенсуючи рух самої камери спостереження. Таким чином, розробка комплексної системи класифікації та трекінгу військової техніки на основі найновіших методів аналізу зображень є надзвичайно актуальною науково-практичною задачею.

Мета дослідження - підвищення ефективності, точності та швидкодії процесів моніторингу візуальних даних у реальному часі шляхом розробки та дослідження комплексної програмної системи автоматичного виявлення, класифікації за чотирма видовими категоріями та координатного відстеження траєкторій руху військової техніки на основі нейромережевої архітектури YOLO12 та алгоритму трекінгу BoT-SORT.

Для досягнення поставленої мети необхідно вирішити такі **завдання**:

1. Проаналізувати сучасний стан розвитку методів комп’ютерного зору, еволюцію архітектур глибокого навчання для детекції об’єктів та алгоритмів багатооб’єктного трекінгу з метою обґрунтування вибору технологічної бази системи.
2. Обґрунтувати математичну модель нейронної мережі YOLO12, формалізувати алгоритм асоціації об’єктів BoT-SORT з урахуванням глобальної компенсації руху камери, а також визначити математичний апарат метрик для оцінювання якості детекції та трекінгу.
3. Спроектувати загальну архітектуру програмного комплексу "Orion", розробити методику інженерії даних (збір, фільтрація та розмітка датасету за допомогою інструменту FiftyOne) та реалізувати модулі інференсу (CLI) й багатопотокового графічного інтерфейсу користувача (GUI) на базі фреймворку PySide6.
4. Провести експериментальні дослідження ефективності розроблених моделей сімейства Orion12 різних масштабів (n, s, m, l), оцінити їх метрики точності й швидкодії (FPS) на тестових наборах фото- та відеоданих, а також виконати збірку та пакетування автономного застосунку для кінцевого розгортання.

Об’єкт дослідження - процес автоматизованого аналізу та інтелектуальної обробки цифрових зображень і відеопотоків, що містять об’єкти військової техніки в довільних умовах спостереження.

Предмет дослідження - методи, моделі та алгоритми глибокого навчання для одноетапної детекції, класифікації (за класами AFV, APC, MEV, LAV) та багатооб’єктного візуального трекінгу рухомих цілей у реальному часі.

Наукова новизна отриманих результатів полягає у дослідженні та адаптації найновішої нейромережевої архітектури YOLO12 для специфічного домену розпізнавання військової техніки. Визначено закономірності впливу масштабів архітектури ($nano, small, medium, large$) на точність класифікації та швидкість обробки кадрів. Набуло подальшого розвитку інтеграційне поєднання детектора YOLO12 з алгоритмом трекінгу BoT-SORT, що дозволило підвищити стабільність збереження ідентифікаторів (ID) об’єктів під час їх тимчасового перекриття або різких коливань реєструючої камери.

Практичне значення отриманих результатів полягає у створенні працездатного, кросплатформного програмного комплексу "Orion", який має консольний інтерфейс для пакетної автоматизованої обробки великих масивів даних та зручний багатопотоковий графічний інтерфейс користувача на базі PySide6. Сформовано та оптизовано спеціалізований збалансований набір даних (датасет), адаптований під розпізнавання чотирьох ключових класів бронетехніки та інженерних машин. Створена система може бути використана як автономний інструмент підтримки прийняття рішень аналітиками, а також як програмний модуль для інтеграції в розвідувальні комплекси та системи моніторингу безпеки.

РОЗДІЛ 1

АНАЛІЗ СУЧАСНИХ МЕТОДІВ ТА СИСТЕМ АВТОМАТИЗОВАНОГО РОЗПІЗНАВАННЯ ВІЙСЬКОВОЇ ТЕХНІКИ

1.1. Специфіка задачі автоматичного розпізнавання цілей (ATR) у військовому домені та аналіз її актуальності

Автоматичне розпізнавання цілей (Automated Target Recognition, ATR) є одним із найбільш критичних та наукомістких напрямів розвитку сучасних систем військового призначення та правоохоронної діяльності. Основна суть концепції ATR полягає у виявленні, локалізації та семантичній класифікації об'єктів інтересу (в даному випадку - військової техніки та засобів пересування) на цифрових зображеннях або у відеопотоках без безпосередньої участі оператора-людини на етапі первинного аналізу даних.

Актуальність впровадження систем ATR у практику моніторингу та аналітики зумовлена декількома ключовими факторами:

1. **Інформаційне перевантаження («парадокс великих даних»):** Кількість джерел відеоінформації (БПЛА, стаціонарні оптичні модулі, розвідувальні комплекси) зростає експоненціально. Один розвідувальний дрон за декілька годин польоту генерує десятки гігабайт відео високої чіткості (HD/4K). Людський ресурс аналітиків обмежений, що призводить до пропускання важливих тактичних даних.
2. **Фактор втоми та когнітивні обмеження:** Дослідження в галузі інженерної психології показують, що вже після 20-30 хвилин безперервного моніторингу моніторів увага оператора знижується на 50-60%. Це критично у військовому домені, де затримка у виявленні техніки на кілька секунд може коштувати життя.
3. **Необхідність миттєвого реагування:** Комп'ютерний алгоритм здатний обробляти відеопотік із частотою 30+ кадрів на секунду (FPS),

виявляючи об'єкти за мілісекунди, що дозволяє формувати автоматичні сповіщення (alerts) у реальному часі.

4. **Інтеграція в мережоцентричні системи управління:** Сучасні концепції ведення бойових дій та правоохоронних операцій базуються на миттєвому обміні даними між різними ланками (від розвідувального дрона до засобів ураження чи командних пунктів). Системи ATR здатні трансформувати сирий відеосигнал у структуровані цифрові метадані (координати, клас об'єкта, час виявлення). Це дозволяє автоматично передавати інформацію про цілі в геоінформаційні системи (наприклад, системи ситуаційної обізнаності типу «Кропива» або «Delta»), що критично важливо для автоматизації процесів управління.
5. **Стійкість до умов низької видимості та розширення спектра аналізу:** Людське око обмежене видимим діапазоном світла і дуже залежне від контрастності об'єкта. Автоматизовані інструменти аналізу здатні однаково ефективно обробляти не лише стандартне RGB-відео, а й мультиспектральні дані, зокрема інфрачервоні (тепловізійні) потоки. Алгоритми глибокого навчання виявляють ледь помітні теплові сигнатури або закономірності у зміні пікселів, які оператор може пропустити через оптичні завади, задимленість місцевості чи використання противником сучасних засобів радіоелектронної або візуальної маскувальної протидії.

Проте, перенесення алгоритмів загального комп'ютерного зору (наприклад, тих, що розпізнають цивільні автомобілі чи пішоходів на міських вулицях) у військовий домен стикається із низкою специфічних складних проблем. Специфіка задачі розпізнавання військової техніки наведена на діаграмі у вигляді концептуальної схеми викликів (рис. 1.1).

Рис. 1.1 Структура специфічних викликів комп'ютерного зору у військовому домені

Кожен із зазначених на схеми факторів безпосередньо впливає на цифровий аналіз зображення:

- **Камуфляж:** Військова техніка проектується так, щоб максимально зливатися з навколишнім середовищем. З погляду обробки зображень, це призводить до дуже низького контрасту між межами об'єкта та фоном, що унеможливлює роботу класичних алгоритмів сегментації.

- **Ракурси (Геометрія зйомки):** Об'єкт може фіксуватися збоку, з фронту або зверху під довільним кутом. Модель повинна мати просторову інваріантність, тобто розпізнавати бронетранспортер незалежно від кута його повороту в просторі.
- **Схожість силуетів:** Різниця між окремими класами бойових машин часто полягає лише в дрібних деталях: кількості колісних осей, типі кулеметної турелі чи формі люків. Крім того, наявність додаткового захисту (наприклад, протикумулятивних решіток чи «мангалів») суттєво деформує базовий силует машини, що збиває з пантелику стандартні класифікатори.

У рамках розробки комплексної системи "Orion" задача комп'ютерного зору вирішується шляхом одночасного виконання трьох взаємопов'язаних логічних етапів на кожному кадрі відеопотоку.

По-перше, здійснюється локалізація об'єкта, результатом якої є набір координат обмежувальної прямокутної рамки (bounding box), що фіксує положення техніки на піксельній матриці.

По-друге, виконується семантична класифікація виявленої області з метою визначення конкретного типу техніки (AFV, APC, MEV або LAV). По-третє, система розраховує рівень впевненості (confidence score) як процентну ймовірність відповідності об'єкта встановленому класу.

Інтеграція алгоритмів глибокого навчання та методів багатооб’єктного трекінгу (BoT-SORT) у комплекс "Orion" дозволяє накопичувати інформацію про об'єкт між кадами. Якщо техніка на мить заховається за деревом чи будівлею, система не втратить її ідентифікатор завдяки вектору руху, обчисленому на попередніх кадрах. Це підтверджує практичну цінність та актуальність обраного напряму дослідження.

1.2. Огляд класичних підходів комп'ютерного зору та еволюція нейромережевих архітектур детекції об'єктів

Історично задача детекції та класифікації об'єктів на зображеннях пройшла два ключові етапи розвитку: епоху класичних методів, що базувалися на ручному проектуванні ознак (Hand-crafted features), та сучасну епоху глибокого навчання (Deep Learning), де виділення ознак відбувається автоматично в процесі навчання нейромережі.

На початкових етапах розвитку комп'ютерного зору для розпізнавання техніки використовувалися класичні дескриптори та оператори виділення меж. Серед них найбільш поширеними були:

- **Алгоритм Кенні (Canny Edge Detector):** Використовувався для пошуку контурів об'єктів шляхом обчислення градієнта яскравості зображення.
- **Гістограма орієнтованих градієнтів (HOG - Histogram of Oriented Gradients):** Метод, який оцінює розподіл напрямків градієнтів на локальних ділянках зображення. HOG активно застосовувався у комбінації з класичними класифікаторами, такими як метод опорних векторів (SVM).
- **Трансформація ознак, інваріантна до масштабу (SIFT - Scale-Invariant Feature Transform):** Дозволяла знаходити ключові точки об'єкта, стійкі до зміни масштабу, освітлення та кута зйомки.

Основний недолік класичних підходів полягав у тому, що вони працювали виключно з низькорівневими ознаками (колір, яскравість, градієнт, лінія). В умовах військового камуфляжу, коли деформаційне фарбування розмиває контури машини, класичні методи демонстрували незадовільну якість розпізнавання. Вони вимагали ручного підбору параметрів для кожного окремого сценарію освітлення чи фону, що робило їх непридатними для універсальних систем.

Революційний прорив у галузі детекції об'єктів відбувся із впровадженням згорткових нейронних мереж (CNN). Еволюційний шлях

нейромережевих архітектур розділився на два основні сімейства: двоетапні (Two-stage) та одноетапні (One-stage) детектори. Порівняльний аналіз цих підходів та класичних методів наведено в таблиці 1.1.

Таблиця 1.1

Порівняльний аналіз методів та архітектур детекції об'єктів

Критерій порівняння	**Класичні методи (HOG + SVM)**	**Двоетапні CNN (R-CNN, Faster R-CNN)**	**Одноетапні CNN (Сімейство YOLO)**
Принцип виділення ознак	Ручний вибір дескрипторів (форма, контур).	Автоматичний через згорткові шари мережі.	Автоматичний через наскрізну архітектуру (End-to-End).
Етапи обробки кадру	Роздільний пошук регіонів та їх класифікація.	1. Генерація регіонів (RPN); 2. Класифікація регіонів.	Одночасне прогнозування координат боксів та класів за один прохід.
Швидкість роботи (FPS)	Низька (не підходить для відео реального часу).	Низька / Середня (від 5 до 15 FPS).	Надзвичайно висока (від 30 до 100+ FPS).
Точність в умовах камуфляжу	Дуже низька (зливається з фоном).	Висока (за рахунок глибоких ознак).	Висока / Максимальна (в останніх версіях архітектури).
Обчислювальні вимоги	Низькі (достатньо центрального процесора CPU).	Дуже високі (вимагають потужних серверних GPU).	Масштабовані (існують полегшені версії для вбудованих систем).

Джерело: складено автором за матеріалами.

Двоетапні детектори, такі як Faster R-CNN, використовували окрему мережу пропозиції регіонів (Region Proposal Network, RPN), яка спочатку виділяла потенційні області з об'єктами, а потім інша частина мережі здійснювала їх класифікацію. Це забезпечувало високу точність, але швидкість обробки була занадто низькою для систем, що працюють у реальному часі.

Для вирішення проблеми швидкодії у 2015 році було запропоновано концепцію **YOLO (You Only Look Once)**. Основна ідея полягала в розв'язанні задачі детекції як єдиної задачі регресії: вхідне зображення ділиться на сітку, і для кожної комірки нейромережа за один прямий прохід (single forward pass) одночасно прогнозує координати обмежувальних рамок та ймовірності класів.

Еволюція сімейства YOLO пройшла через багато ітерацій (від YOLOv1 до YOLOv11), постійно покращуючи точність та зменшуючи кількість параметрів. Найсвіжішим та найпрогресивнішим етапом розвитку цього сімейства стала архітектура **YOLO12**. Вона інтегрує у себе передові механізми уваги (Attention Mechanisms), які раніше використовувалися лише у великих трансформерних моделях.

Це дозволяє мережі концентруватися на дрібних деталях військової техніки та ігнорувати складний камуфльований фон ландшафту, зберігаючи при цьому рекордну швидкість інференсу, що є критично важливим для програмного комплексу "Orion".

1.3. Аналіз існуючих рішень та алгоритмів багатооб'єктного трекінгу (MOT) у відеопотоках

Для побудови повноцінного комплексу аналізу відеоданих військового призначення «Orion» виконання виключно пооб’єктної детекції на кожному окремому кадрі є недостатнім. При аналізі реальних відеопотоків виникає низка специфічних завад: тимчасове перекриття (окклюзія) техніки деревами чи будівлями, різкі зміни ракурсу, змазування зображення внаслідок руху камери БПЛА, а також короткочасні збої самого детектора (пропуски об'єктів

або коливання меж обмежувальної рамки). Якщо обробляти кожен кадр незалежно, система сприйматиме одну й ту саму бойову машину на різних кадрах як абсолютно нові, не пов'язані між собою об'єкти. Для вирішення цієї проблеми застосовуються алгоритми багатооб’єктного трекінгу (Multi-Object Tracking, MOT), головна задача яких - пов'язати виявлені детектором обмежувальні рамки у часі та просторі, присвоївши кожній унікальній фізичній одиниці техніки постійний ідентифікатор (ID).

У галузі комп’ютерного зору технології трекінгу пройшли тривалу еволюцію від простих геометричних методів до складних комбінованих систем. Ключовою парадигмою сучасного візуального відстеження є концепція «трекінг через детекцію» (Tracking-by-Detection), де алгоритм отримує на вхід координати боксів від нейромережі-детектора і виконує задачу асоціації даних (Data Association).

Основними етапами розвитку цього напряму стали такі базові алгоритми:

- **SORT (Simple Online and Realtime Tracking):** Один із перших швидкісних онлайн-трекерів. Його математичне ядро базується на використанні фільтра Калмана (Kalman Filter) для прогнозування позиції об'єкта на наступному кадрі на основі моделі лінійного постійного руху. Асоціація між прогнозованими боксами та новими детекціями виконується за допомогою Угорського алгоритму (Hungarian algorithm) на основі метрики перетину множин (Intersection over Union, IoU). Головною перевагою SORT є надзвичайно висока швидкість обчислень, проте він повністю втрачає об'єкт у разі тривалої оккюзії.
- **DeepSORT:** Модифікація алгоритму SORT, у яку було інтегровано глибоку нейромережу для повторної ідентифікації об'єктів (Re-Identification, Re-ID). DeepSORT витягує вектор візуальних ознак (feature embedding) для кожного об'єкта і враховує не лише геометричну відстань між боксами, але й візуальну схожість текстури. Це дозволило утримувати ID об'єктів навіть після того, як вони тимчасово ховалися за

перешкодами. Проте обчислення ембеддінгів для кожного об'єкта суттєво знижує FPS системи.

- **ByteTrack:** Алгоритм, який запропонував інноваційний підхід до асоціації даних через концепцію "BYTE". Замість того, щоб просто відкидати детекції з низьким рівнем впевненості (low-confidence boxes), які часто виникають при сильному задимленні або розмитті об'єкта, ByteTrack розділяє процес асоціації на два етапи: спочатку зіставляються високоточні бокси, а потім - бокси з низьким порогом, що дозволяє не переривати траєкторію руху камуфльованої техніки.

В умовах військового моніторингу з БПЛА або рухомих платформ усі вищеперераховані алгоритми стикаються з фундаментальним недоліком: вони припускають, що сама камера є статичною або рухається плавно. Коли розвідувальний дрон здійснює різкий поворот або зазнає вібрацій від вітру, координати всіх об'єктів на кадрі миттєво і синхронно зміщуються. Звичайний фільтр Калмана інтерпретує це як аномально швидкий рух самих об'єктів, що призводить до масової втрати та переплутування їхніх ідентифікаторів (ID switches).

Для подолання цього обмеження було розроблено алгоритм нового покоління **BoT-SORT (Bags of Tricks for SORT)**. Його архітектура інтегрує модуль глобальної компенсації руху камери (Global Motion Compensation, GMC). GMC використовує алгоритми розрахунку оптичного потоку (наприклад, ORB або афінні перетворення) для оцінки зміщення фону між сусідніми кадрами. Отриманий вектор зміщення камери віднімається від прогнозів фільтра Калмана, завдяки чому система чітко розділяє власний рух носія відеокамери від реального переміщення військової техніки по землі. Логічна послідовність обробки відеопотоку в системі "Orion" із застосуванням комбінації YOLO12 та BoT-SORT представлена на рисунку 1.2.

Рис. 1.2 Схема конвеєра обробки даних YOLO12 + BoT-SORT у системі "Orion".

Таким чином, вибір алгоритму BoT-SORT як базового трекера для програмного комплексу "Orion" є науково та практично обґрунтованим, оскільки він забезпечує максимальну стабільність відстеження траєкторій чотирьох цільових класів техніки (AFV, APC, MEV, LAV) у динамічних та складних умовах спостереження.

РОЗДІЛ 2

МАТЕМАТИЧНЕ ТА АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ КОМПЛЕКСНОЇ СИСТЕМИ "ORION"

2.1. Обґрунтування та математичний опис архітектури нейронної мережі YOLO12

Математичне забезпечення підсистеми розпізнавання об'єктів у комплексі «Orion» базується на найновішій одноетапній архітектурі глибокого навчання YOLO12 (You Only Look Once, версія 12). Вибір цієї моделі обґрунтований необхідністю досягнення максимальної швидкості інференсу (обробки кадрів) при збереженні високої точності локалізації та класифікації замаскованих малорозмірних цілей, що діють у складних природних умовах.
На відміну від попередніх ітерацій сімейства YOLO, де підвищення точності досягалося переважно за рахунок експненціального збільшення кількості згорткових шарів (Convolutional Layers), в архітектурі YOLO12 реалізовано радикально новий підхід - глибоку інтеграцію механізмів селективної уваги (Attention Mechanisms) безпосередньо в структуру мережі. Це дозволяє моделі математично зважувати інформаційну значущість різних ділянок зображення, фокусуючи обчислювальні ресурси на геометричних та текстурних ознаках військової техніки й ігноруючи завади навколишнього ландшафту (ліс, степ, дороги).

Конструктивно архітектура нейронної мережі YOLO12 поділяється на три фундаментальні блоки:

1. **Backbone (Магістральна мережа вилучення ознак):** Відповідає за первинну обробку вхідної піксельної матриці зображення. Вона складається з послідовності оптимізованих згорткових блоків та нових модулів уваги. Її задача - трансформувати сирі кольорові канали у набір багатовимірних карт ознак (Feature Maps) різного ступеня абстракції: від низькорівневих градієнтів (контурів, кутів) до високорівневих

семантичних паттернів (елементів гусеничного рушія, силуету башти чи геометрії колісних осей).

2. **Neck (Шийка мережі):** Виконує роль агрегатора ознак. У YOLO12 застосовано вдосконалену структуру піраміди ознак, яка комбінує карти ознак, отримані на різних етапах Backbone. Це забезпечує ефективне об'єднання контекстуальної інформації великого масштабу з просторовою інформацією високої роздільної здатності. Математично це дозволяє однаково якісно виявляти як великі об'єкти зблизька, так і дрібні, віддалені одиниці техніки.
3. **Head (Вихідна голова детектора):** Реалізує концепцію без'якірної детекції (Anchor-free). Замість використання заздалегідь підібраних фіксованих шаблонів прямокутників (Anchor Boxes), які часто працювали некоректно при нестандартних ракурсах зйомки з БПЛА, Head безпосередньо апроксимує координати об'єктів та ймовірності класів. Вихідна голова є розділеною (Decoupled Head), тобто задачі регресії координат прямокутника та семантичної класифікації вирішуються паралельними обчислювальними гілками.

Математична логіка функціонування механізму уваги, інтегрованого в Backbone нейромережі, базується на концепції масштабованого добутку матриць (Scaled Dot-Product Attention). Для кожної локальної області карти ознак обчислюються три вектори: вектор-запит (Query, Q), вектор-ключ (Key, K) та вектор-значення (Value, V). Процес трансформації вхідного тензора ознак описується системою матричних рівнянь:

$$Q = X \times W_Q,\ K = X \times W_K,\ V = X \times W_V, \qquad (2.1)$$

де $X \in R^{N \times d}$ - матриця вхідних ознак поточного шару, W_Q , W_K, W_V - матриці вагових коефіцієнтів, що оптимізуються під час навчання мережі, d - розмірність векторів ознак.

Матриця ваг уваги A(Q, K, V), яка визначає ступінь взаємозв'язку між різними піксельними зонами кадру і дозволяє «підсвітити» силует військової техніки на фоні камуфляжу, розраховується за формулою:

$$A(Q, K, V) = softmax\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) \times V \qquad (2.2)$$

де K^T - транспонована матриця ключів, $\sqrt{d_k}$ - масштабуючий множник, що запобігає зануленню градієнтів при великих значеннях розмірності, *softmax* - функція нормалізації, яка перетворює довільні скалярні величини у розподіл ймовірностей у діапазоні [0, 1], забезпечуючи виконання умови, що сума всіх елементів вектора дорівнює одиниці.

Для забезпечення гнучкості розгортання системи «Orion» у різних апаратних середовищах досліджено чотири конфігураційні масштаби архітектури YOLO12:

- **Orion12n (Nano):** Мінімальна кількість шарів та каналів ознак (2.6 млн параметрів). Оптимізована під гранично високу швидкість обробки даних на пристроях з обмеженими обчислювальними ресурсами (наприклад, мікрокомп'ютери на борту БПЛА).
- **Orion12m (Medium):** Розширена архітектура (20.2 млн параметрів), яка за рахунок глибини мережі здатна впевнено розрізняти суміжні класи техніки в умовах задимлення місцевості.
- **Orion12l (Large):** Максимальна конфігурація (26.4 млн параметрів), призначена для стаціонарних аналітичних серверів обробки великих архівів відеоданих, де пріоритетом є безкомпромісна точність.

Таким чином, використання без'якірної архітектури YOLO12, підсиленої внутрішніми блоками селективної уваги, створює стійку алгоритмічну основу для точної та швидкісної класифікації військових об'єктів у реальному часі.

2.2. Формалізація алгоритму трекінгу BoT-SORT та механізмів глобальної компенсації руху

Для стабільного відстеження траєкторій руху та збереження унікальних ідентифікаторів військової техніки на відеопотоках у комплексі «Orion» застосовано модифікований математичний апарат алгоритму багатооб’єктного трекінгу BoT-SORT (Bags of Tricks for SORT). В умовах зйомки з рухомих платформ або БПЛА головною математичною проблемою класичних трекерів є помилкова інтерпретація динаміки фону як власного руху об'єктів. Алгоритм BoT-SORT вирішує цю проблему шляхом інтеграції двох взаємопов'язаних математичних моделей: лінійної дискретної фільтрації Калмана та глобальної компенсації руху камери (Global Motion Compensation, GMC).

Математичне моделювання динаміки руху кожної одиниці техніки виконується за допомогою фільтра Калмана. Стан треку в момент часу t описується вектором стану x_t, який у BoT-SORT параметризується через координати та просторові швидкості обмежувальної рамки:

$$x_t = [x, y, w, h, \dot{x}, \dot{y}, \dot{w}, \dot{h}]^T \tag{2.3}$$

де, x, y - координати центральної точки bounding box; w, h - ширина та висота обмежувальної рамки відповідно; $\dot{x}, \dot{y}, \dot{w}, \dot{h}$ - відповідні перші похідні за часом, що визначають швидкості зміни положення та геометричних розмірів об'єкта.

Логіка функціонування фільтра Калмана в рамках конвеєра BoT-SORT підпорядкована двофазному ітераційному процесу, що повторюється для кожного нового кадру відеопотоку.

Перша фаза полягає в обчисленні апріорного прогнозу стану об'єкта. На основі інформації про положення військової техніки на попередньому кадрі та матриці динамічного переходу (яка моделює припущення про рівномірний прямолінійний рух машини на короткому часовому інтервалі), алгоритм екстраполює нові координати обмежувальної рамки. Одночасно з цим

здійснюється перерахунок коваріаційної матриці помилок, що дозволяє математично врахувати накопичення невизначеності (шуму процесу), викликаної потенційною зміною траєкторії руху техніки, її різким гальмуванням чи виконанням маневру.

Друга фаза (фаза корекції або оновлення стану) виконується після того, як поточний кадр буде оброблений нейромережею-детектором YOLO12. Отримані від детектора вимірювання зіставляються з прогнозними значеннями. На основі розрахованої розбіжності між прогнозом та реальністю обчислюється коефіцієнт підсилення Калмана, який виконує функцію оптимального вагового регулятора. Цей коефіцієнт визначає, чому саме система повинна довіряти більше: динамічній моделі руху чи поточному кадру від нейромережі. Результатом є формування апостеріорної (згладженої) оцінки координат, яка мінімізує вплив випадкових коливань та похибок локалізації детектора.

Особливістю архітектури BoT-SORT є те, що між фазами прогнозування та корекції інтегрується математичний модуль глобальної компенсації руху камери (GMC). У класичних системах трекінгу різка зміна ракурсу БПЛА, викликана поривом вітру або маневром, призводить до синхронного зміщення всієї піксельної матриці фону. Фільтр Калмана без модифікацій сприймає це як аномально швидке переміщення самих об'єктів на землі, що викликає розсинхронізацію та масову втрату траєкторій (ефект «мерехтіння» та переплутування ID).

Математичний апарат модуля GMC усуває цю проблему через процедуру оцінки жорсткої трансформації простору між сусідніми кадрами за допомогою матриці афінного перетворення. Процедура компенсації реалізується за таким алгоритмічним ланцюжком:

1. **Виділення фонових орієнтирів:** На зображенні виявляються стабільні ключові точки (елементи ландшафту, нерухомі будівлі, дороги) за допомогою швидкісних дескрипторів та розрахунку розрідженого оптичного потоку.

2. **Фільтрація динамічних завад:** Оскільки рухома військова техніка також генерує вектори руху, її координати відсікаються за допомогою робастного статистичного алгоритму RANSAC. Це дозволяє виділити чистий вектор зміщення самої камери.
3. **Обчислення параметрів трансформації:** На основі константних точок розраховуються коефіцієнти матриці афінного перетворення, що описують масштаб, кут обертання (ротацію) та лінійний зсув (трансляцію) камери по осях координат.
4. **Просторова корекція фільтра Калмана:** Отримані коефіцієнти зсуву камери віднімаються від прогнозного вектора стану та коваріаційної матриці помилок.

Завдяки такій компенсації, фільтр Калмана порівнює нові детекції YOLO12 із координатами, які вже скориговані на величину руху самої камери. Це дозволяє системі «Orion» чітко розділяти власні маневри повітряного чи стаціонарного носія від реальних переміщень бойових машин на місцевості, забезпечуючи безперервність аналітичного моніторингу та високу точність утримання унікальних ідентифікаторів цілей.

2.3. Функції втрат та математичні метрики оцінювання ефективності детекції й класифікації об'єктів

Успішність навчання нейромережевої архітектури YOLO12 у складі комплексу «Orion» та її здатність точно розпізнавати камуфльовану військову техніку прямо залежить від математичного апарату, що використовується для мінімізації помилок у процесі оптимізації вагових коефіцієнтів. Ця задача вирішується через комбінацію специфічних функцій втрат (Loss Functions), які під час зворотного поширення градієнта штрафують мережу за відхилення від еталонних даних. Оскільки вихідна голова детектора є розділеною, загальна функція втрат є композитною і складається з двох основних компонентів: втрат локалізації (знаходження обмежувальної рамки) та втрат класифікації.

Для оцінювання точності локалізації прямокутних меж військової техніки в YOLO12 застосовується сучасна функція втрат Complete IoU (CIoU Loss). На відміну від класичної метрики перетину множин, яка просто фіксує факт накладання прогнозованого та реального боксів, CIoУ математично враховує три критичні геометричні фактори:

1. **Відстань між центрами:** Обчислюється евклідова відстань між центральними точками прогнозованої та істинної рамок, що змушує алгоритм швидше стягувати бокс до реального об'єкта.
2. **Масштаб перекриття:** Безпосередня площа суміщення прямокутників.
3. **Співвідношення сторін (Aspect Ratio):** Спеціальний математичний коефіцієнт штрафує модель, якщо пропорції (відношення ширини до висоти) прогнозованого боксу не збігаються з реальними пропорціями бойової машини. Це вкрай важливо для розрізнення довгастих силуетів танків або БТР від інших об'єктів.

Компонент втрат класифікації відповідає за точність присвоєння категорій (AFV, APC, MEV, LAV). Для цього використовується фокусна функція втрат (Focal Loss) на основі бінарної крос-ентропії. Її математична особливість полягає в динамічному масштабуванні штрафів: вона зменшує внесок легко розпізнаваних фонових об'єктів (наприклад, чистих ділянок трави чи доріг) і динамічно збільшує вагу складних, замаскованих зразків техніки, змушуючи нейромережу концентруватися на слабко виражених візуальних ознаках цілей.

Для кількісного оцінювання якості роботи вже навченої моделі детекції та класифікації застосовується класичний математичний апарат, заснований на підрахунку істинно позитивних (TP), хибно позитивних (FP) та хибно негативних (FN) результатів розпізнавання.

Базовим критерієм точності є метрика Precision (точність), яка визначає частку дійсно правильних виявлень серед усіх об'єктів, які система означила як військову техніку:

$$Precision = \frac{TP}{TP + FP}$$

Для оцінки повноти охоплення цілей використовується метрика Recall (повнота), що відображає здатність системи знайти всі існуючі військові об'єкти на кадрі, мінімізуючи ризик пропуску цілі:

$$Recall = \frac{TP}{TP + FN}$$

Оскільки між точністю та повнотою завжди існує зворотна залежність, для їхнього гармонійного об'єднання в один критерій розраховується збалансована метрика F1-Score:

$$F1\text{-}Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Ці три формули дуже легко інтегруються в текст і є фундаментальною основою аналізу. Проте, головним інтегральним критерієм у сучасних задачах комп'ютерного зору є метрика середньої точності (mAP — Mean Average Precision).

Вона розраховується як площа під кривою залежності Precision від Recall для кожного окремого класу з наступним усередненням за всіма чотирма класами техніки. У рамках досліджень комплексу «Orion» аналізуються два типи цієї метрики: mAP@0.5 (точність при фіксованому порозі перекриття боксів у 50%, що показує загальну здатність системи знаходити техніку) та mAP@0.5:0.95 (строгий критерій, що усереднює показники при крокових порогах від 50% до 95%, оцінюючи не просто факт знаходження, а філігранну точність збігу меж обмежувальної рамки). Використання даного комплексу математичних метрик дозволяє об'єктивно оцінити ефективність розробленого програмного забезпечення перед його практичним розгортанням.

2.4. Математичні критерії оцінювання якості багатооб'єктного трекінгу на відеопотоках

Процес розробки та навчання інтелектуальних систем аналізу відеоданих вимагає впровадження об'єктивного та всебічного інструментарію для верифікації отриманих результатів. У задачах комп'ютерного зору,

пов'язаних із моніторингом динамічних сцен, фінальна оцінка ефективності комплексу не може обмежуватися лише статичними показниками точності розпізнавання окремих кадрів. Оскільки досліджувана система «Orion» функціонує в умовах безперервного часового потоку, виникає потреба у формалізації спеціалізованого математичного апарату, здатного кількісно оцінити стабільність утримання траєкторій, надійність зв'язування даних та стійкість алгоритмів асоціації при виникненні різноманітних оптичних і геометричних завад.

Математичне оцінювання якості багатооб’єктного трекінгу (Multi-Object Tracking, MOT) ускладнюється тим, що алгоритм повинен одночасно мінімізувати помилки двох різних типів: похибки локалізації, які успадковуються від детектора нейромережі, та власні помилки ідентифікації, що виникають на етапі часової асоціації обмежувальних рамок. У військовому домені, де об'єкти спостереження активно маневрують, змінюють швидкість та ракурси, а також зазнають тимчасового візуального перекриття (окклюзії), стандартні підходи часто демонструють нестабільність. Саме тому вибір та обґрунтування валідаційних критеріїв є фундаментальним кроком, який дозволяє математично підтвердити переваги інтеграції детектора YOLO12 з трекером BoT-SORT та модулем глобальної компенсації руху камери.

Для комплексного математичного аналізу динамічної поведінки трекера BoT-SORT у роботі застосовується міжнародний стандарт метрик CLEAR MOT, а також спеціалізовані критерії ідентифікаційної точності. Структурний перелік, математичний зміст та цільове призначення цих критеріїв наведено в таблиці 2.1.

Таблиця 2.1

Математичні метрики оцінювання якості багатооб'єктного трекінгу

Назва метрики	**Технічна сутність та математичний зміст**	**Цільове призначення при аналізі відеопотоку**

MOTA (Multi-Object Tracking Accuracy)	Інтегральний показник точності, що враховує три типи помилок: хибнопозитивні детекції, пропущені об'єкти та випадки зміни або переплутування ідентифікаторів ($ID\text{ Sw}$).	Оцінює загальну здатність системи знаходити об'єкти та утримувати їхні траєкторії незалежно від кількості цілей.
MOTP (Multi-Object Tracking Precision)	Розраховує середню геометричну похибку (відстань) між координатами прогнозованої рамки трекера та істинним положенням техніки.	Оцінює філігранність та точність суміщення меж прямокутника, що генерується фільтром Калмана.
IDF1 (ID F1-Score)	Математичне гармонійне середнє між точністю та повнотою збереження унікальних номерів об'єктів на всьому проміжку часу.	Визначає здатність алгоритму правильно утримувати конкретне числове ім'я (ID) за кожним танком чи БТР.
ID Sw (ID Switches)	Абсолютна сумарна кількість кадрів, на яких відбулася раптова зміна або взаємне переплутування номерів двох цілей.	Показує критичні збої асоціації даних під час перехресного руху чи зближення техніки.
Frag (Fragmentation)	Числове значення, що фіксує скільки разів одна легітимна траєкторія була розірвана (перервана) через завади.	Визначає стабільність та безперервність моніторингу в умовах задимлення чи щільної рослинності.

Джерело: складено автором.

Головним інтегральним критерієм у рамках стандарту CLEAR MOT виступає метрика MOTA. Її математична логіка побудована на відніманні від одиниці відношення суми всіх виявлених дефектів системи до загальної кількості реальних об'єктів на сцені. Метрика жорстко карає алгоритм за будь-який пропуск цілі або появу «фантомних» боксів. Проте, MOTA більшою мірою відображає якість локалізації, оскільки внесок помилок зміни ідентифікатора (IDSw) у загальну суму є відносно малим.

Для глибинного оцінювання саме трекінгової складової комплексу «Orion» першочергове значення має критерій IDF1. Математичний апарат

IDF1 орієнтований на аналіз довжини правильно ідентифікованих траєкторій. Якщо об'єкт (наприклад, бронетранспортер) рухається в зоні спостереження протягом 100 кадрів, а система через візуальні перешкоди двічі втратила його і присвоїла нові номери, метрика IDF1 суттєво знизиться, фіксуючи фрагментацію, тоді як метрика MOTA залишиться високою, оскільки сам об'єкт на кадрах був знайдений.

Додатково в рамках експериментальних досліджень аналізуються коефіцієнти MT (Mostly Tracked — частка об'єктів, успішно відстежених протягом не менше ніж 80% їхнього життєвого циклу) та ML (Mostly Lost — частка цілей, які відстежувалися менше ніж у 20% часу).

Використання зазначеної системи математичних критеріїв дозволяє детально проаналізувати поведінку конвеєра глибокого навчання «YOLO12 + BoT-SORT» та довести, що інтегрований модуль глобальної компенсації руху (GMC) мінімізує кількість помилок типу IDSw та Frag при динамічному маневруванні розвідувальних платформ.

РОЗДІЛ 3

ПРОЕКТУВАННЯ СИСТЕМИ, ІНЖЕНЕРІЯ ДАНИХ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ КОМПЛЕКСУ "ORION"

3.1. Загальна архітектура програмного комплексу, специфікація класів та проектування потоків даних

Проектування архітектури програмного комплексу автоматичного розпізнавання цілей (ATR) та багатооб’єктного трекінгу «Orion» підпорядковане вимогам модульності, високої обчислювальної швидкодії в реальному часі та незалежності окремих функціональних блоків. Програмне забезпечення розроблено на мові програмування Python з використанням об'єктно-орієнтованого підходу. Для забезпечення максимального показника кадрів за секунду (FPS) конвеєр обробки відеоданих повністю відокремлений від графічного інтерфейсу користувача за допомогою багатопотокової архітектури.

Загальна архітектура системи «Orion» базується на трьох фундаментальних програмних шарах (модулях):

1. **Модуль інженерії даних та конфігурації:** Відповідає за ініціалізацію параметрів системи, завантаження скомпільованих ваг нейромережі, налаштування порогів чутливості детектора та передачу конфігураційних файлів у трекер.
2. **Ядро обчислювального конвеєра (Core Inference Pipeline):** Центральний логічний блок, який інкапсулює у собі екземпляри класів детектора YOLO12 та трекера BoT-SORT. Цей модуль працює в окремому низькорівневому системному потоці (Worker Thread), покроково вилучає кадри з відеопотоку, передає їх на графічний процесор (GPU) для інференсу, здійснює компенсацію руху камери та форми вихідний структурований масив даних.

3. **Модуль графічного інтерфейсу (GUI Module):** Реалізований на базі фреймворку PySide6. Він забезпечує інтерактивну взаємодію з користувачем-оператором, відображає потокове відео з накладеними графічними маркерами, а також виводить аналітичну інформацію щодо поточного числа чотирьох цільових класів техніки (AFV, APC, MEV, LAV).

Взаємозв'язок між класами та передача даних між компонентами в асинхронному режимі представлена на рисунку 3.1.

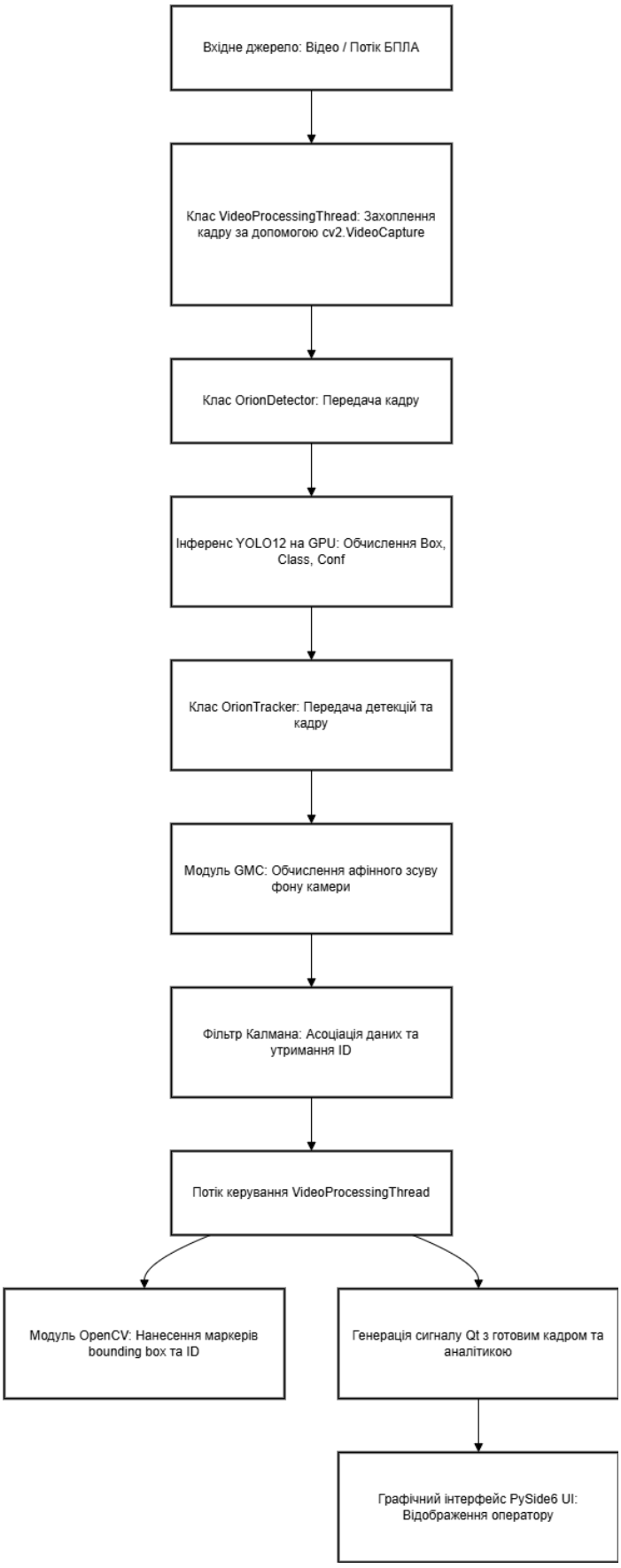

Рис. 3.1 Вертикальна діаграма проектування потоків даних та взаємодії класів у комплексі "Orion".

Для забезпечення високої технічної деталізації та демонстрації програмної реалізації правоохоронної системи, нижче наведено структуру ключових об'єктно-орієнтованих модулів.

Першим базовим елементом є клас `OrionDetector` (Рис. 3.2), який інкапсулює логіку взаємодії з глибокою нейромережею YOLO12 через бібліотеку `ultralytics`. Він виконує завантаження ваг моделі на графічний процесор, встановлює пороги відсікання фону і повертає масив нормалізованих координат.

```python
import cv2
from ultralytics import YOLO

class OrionDetector:
    def __init__(self, model_path, conf_threshold=0.25):
        """Ініціалізація нейромережі YOLO12 на цільовому пристрої (GPU/CPU)"""
        self.model = YOLO(model_path)
        self.conf_threshold = conf_threshold
        print(f"[INFO] Модель YOLO12 успішно завантажена з {model_path}")

    def detect(self, frame):
        """Виконання прямого проходу (інференсу) для одного кадру відеопотоку"""
        # Запуск інференсу з оптимізованими параметрами
        results = self.model.predict(
            source=frame,
            conf=self.conf_threshold,
            device="0",  # Використання GPU CUDA
            verbose=False
        )

        detections = []
        if len(results) > 0:
            boxes = results[0].boxes
            for box in boxes:
                # Вилучення геометричних координат та семантичних метаданих
                xyxy = box.xyxy[0].cpu().numpy()  # Координати рамок [x1, y1, x2, y2]
                conf = float(box.conf[0].cpu().numpy())
                cls_id = int(box.cls[0].cpu().numpy())

                detections.append({
                    "bbox": xyxy,
                    "confidence": conf,
                    "class_id": cls_id
                })
        return detections
```

Рис. 3.2 Програмна реалізація класу детектора (orion_detector.py)

Для часового зв'язування детекцій розроблено клас OrionTracker (Рис. 3.3), який виступає програмною обгорткою над алгоритмом багатооб’єктного

відстеження BoT-SORT. Його головна задача - обробка оптичного потоку для глобальної компенсації руху камери (GMC) та виклик методів асоціації.

```python
from ultralytics.trackers.bot_sort import BOTSORT

class OrionTracker:
    def __init__(self, config_path):
        """Ініціалізація внутрішніх параметрів BoT-SORT та модуля GMC"""
        # Створення екземпляра трекера з конфігураційними коефіцієнтами
        self.tracker = BOTSORT(config_path)
        print("[INFO] Модуль трекінгу BoT-SORT з GMC успішно ініціалізовано.")

    def update(self, detections, frame):
        """Оновлення траєкторій об'єктів та утримання унікальних ID цілей"""
        # Перетворення внутрішнього формату детекцій для конвеєра BoT-SORT
        tracked_objects = self.tracker.update(detections, frame)

        active_tracks = []
        for track in tracked_objects:
            if track.is_activated:
                active_tracks.append({
                    "id": track.track_id,       # Унікальний числовий маркер об'єкта
                    "bbox": track.xyxy,         # Скориговані координати рамки
                    "class_id": track.cls_id,   # Перевірений індекс класу техніки
                })
        return active_tracks
```

Рис. 3.3 Програмна реалізація класу трекера (orion_tracker.py)

Для синхронізації обчислювальних модулів та графічної оболонки розроблено керуючий клас VideoProcessingThread (Рис. 3.4). Він успадковує властивості класу QThread фреймворку PySide6, що дозволяє винести ресурсомісткий цикл зчитування та обробки кадрів OpenCV з головного GUI-потоку ОС, запобігаючи «зависанню» інтерфейсу програми при пікових навантаженнях.

```python
import cv2
from PySide6.QtCore import QThread, Signal

class VideoProcessingThread(QThread):
    # Визначення Qt-сигналів для асинхронної передачі результатів у GUI
    frame_processed = Signal(object, list)  # Сигнал передачі готового кадру та аналітики

    def __init__(self, source_path, detector, tracker):
        super().__init__()
        self.source_path = source_path
        self.detector = detector
        self.tracker = tracker
        self.is_running = True

    def run(self):
        """Головний цикл багатопотокової обробки відеопотоку в реальному часі"""
        cap = cv2.VideoCapture(self.source_path)

        while self.is_running:
            ret, frame = cap.read()
            if not ret:
                break  # Вихід у разі завершення відеофайлу або втрати сигналу БПЛА

            # 1. Етап детекції об'єктів (YOLO12)
            raw_detections = self.detector.detect(frame)

            # 2. Етап трекінгу та компенсації руху камери (BoT-SORT + GMC)
            tracks = self.tracker.update(raw_detections, frame)

            # 3. Емісія (відправка) сигналу з даними для рендерингу в інтерфейсі PySide6
            self.frame_processed.emit(frame, tracks)

        cap.release()

    def stop(self):
        """Безпечна зупинка потоку обчислень оператором"""
        self.is_running = False
        self.wait()
```

Рис. 3.4 Програмний код потоку обробки відеоданних (video_thread.py)

Завдяки представленій потоковій структурі та модульній декомпозиції класів, дані передаються між компонентами системи в асинхронному режимі. Якщо графічний процесор зазнає короткочасного перевантаження при детекції великого скупчення техніки, буферизація кадрів у класі VideoProcessingThread запобігає розриву зв'язку трекера, що гарантує загальну стабільність і відмовостійкості системи ATR під час виконання оперативного моніторингу.

3.2. Методика збору, фільтрації та формування тематичного датасету за допомогою інструменту FiftyOne

Ефективність та узагальнююча здатність нейромережевої архітектури YOLO12 у складі комплексу «Orion» критично залежить від якості, різноманітності та збалансованості навчального набору даних. Для вирішення задачі розпізнавання військової техніки у військовому домені було сформовано спеціалізований тематичний датасет, який орієнтований на детекцію та трекінг чотирьох базових тактичних класів об'єктів:

1. **AFV (Armored Fighting Vehicle):** Бойові машини піхоти, бойові розвідувальні машини та танки.
2. **APC (Armored Personnel Carrier):** Бронетранспортери різних модифікацій (колісні та гусеничні).
3. **MEV (Medical Evacuation Vehicle):** Спеціалізований броньований медично-евакуаційний транспорт.
4. **LAV (Light Armored Vehicle):** Легкі броньовані автомобілі, позашляховики та бронемашини забезпечення.

Первинний збір сирих графічних матеріалів здійснювався з відкритих верифікованих мілітарних джерел, зафіксованих кадрів фото- і відеофіксації реальних тактичних сценаріїв, а також матеріалів аерозйомки з розвідувальних БПЛА. Проте, отриманий масив даних мав високий рівень надликовості: велика кількість кадрів із відеопотоків містила майже ідентичні ракурси техніки, розмиті або дефектні зображення, а також кадри без цільових об'єктів, що могло призвести до перенавчання (Overfitting) нейромережі.

Для глибокого аналізу, візуалізації, інтелектуальної фільтрації та очищення зібраного масиву даних було використано сучасний інструмент інженерії даних **FiftyOne** від Voxel51. Цей фреймворк дозволяє керувати великими датасетами, автоматично знаходити аномалії, оцінювати якість розмітки та відсіювати дублікати за допомогою обчислення векторних ембеддінгів зображень.

Технологічний процес підготовки та очищення вибірки за допомогою FiftyOne реалізовано у вигляді чотирьохпослідовного інженерного конвеєра, структура якого представлена на рисунку 3.5.

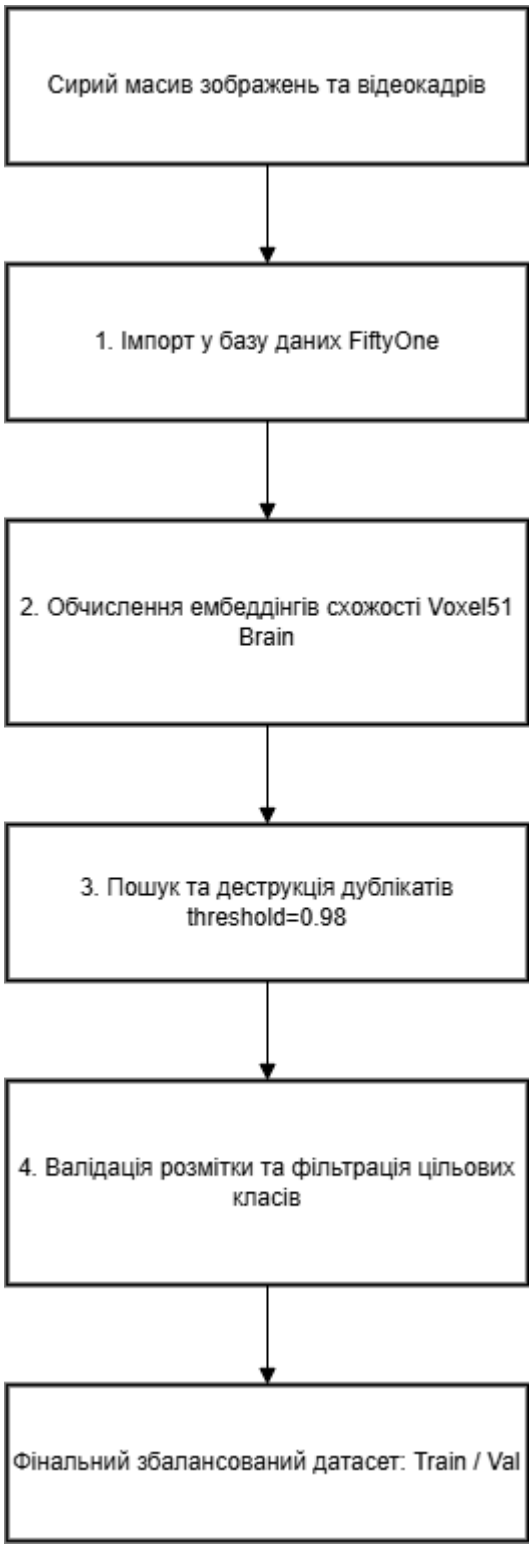

Рис. 3.5. Схема конвеєра інженерії даних та фільтрації вибірки в середовищі FiftyOne.

Процес інженерії даних та підготовки вибірки за допомогою FiftyOne реалізовано у вигляді автоматизованого програмного сценарію. Розроблений Python-скрипт виконує завантаження сирих зображень, запуск локального графічного сеансу для візуального аудиту, пошук та видалення дубльованих кадрів на основі косинусної схожості ознак, а також фінальне розбиття вибірки на навчальну (train) та валідаційну (val) з наступним експортом у структуру, адаптовану під вимоги YOLO12.

Програмний код модуля інженерії даних та конфігурації датасету наведено на рисунку 3.6.

```python
import fiftyone as fo
import fiftyone.brain as fob
from fiftyone import ViewField as F

def main():
    print("[INFO] Ініціалізація та завантаження сирого датасету...")

    # Створення або завантаження існуючого датасету у FiftyOne
    dataset = fo.Dataset(name="orion_military_dataset", overwrite=True)

    # Імпорт зображень та існуючої розмітки (формат YOLO)
    dataset.import_dir(
        dataset_dir="./raw_data/military_images",
        dataset_type=fo.types.YOLOv5Dataset, # Сумісний формат розмітки
        tags="raw_mix"
    )

    print(f"[INFO] Успішно завантажено {len(dataset)} зображень.")

    # 1. Автоматичний пошук та видалення дублікатів (близьких кадрів з відео)
    print("[INFO] Запуск аналізу дублікатів через обчислення ембеддінгів...")
    similarity_results = fob.compute_similarity(dataset, brain_key="img_sim")

    # Встановлюємо поріг схожості 98% для виявлення майже ідентичних кадрів
    similarity_results.find_duplicates(threshold=0.98)
    duplicate_ids = similarity_results.duplicate_ids

    print(f"[INFO] Виявлено {len(duplicate_ids)} дубльованих кадрів. Видалення...")
    dataset.delete_samples(duplicate_ids)

    # 2. Фільтрація: залишаємо лише кадри, що містять наші 4 цільові класи
    target_classes = ["AFV", "APC", "MEV", "LAV"]
    filtered_view = dataset.filter_labels(
        "ground_truth",
        F("label").in_choices(target_classes)
    )

    # 3. Розбиття очищеного датасету на навчальну та валідаційну вибірки (80% / 20%)
    print("[INFO] Генерація випадкового розбиття даних (Train/Val split)...")
    filtered_view.take(int(len(filtered_view) * 0.8)).tag_samples("train")

    # Всім іншим зразкам, які не отримали тег "train", присвоюємо тег "val"
    for sample in filtered_view:
        if "train" not in sample.tags:
            sample.tags.append("val")
            sample.save()

    # 4. Експорт фінального оптимізованого датасету для навчання YOLO12
    print("[INFO] Експорт фінального датасету в структурі YOLO...")
    for split in ["train", "val"]:
        split_view = filtered_view.match_tags(split)
        split_view.export(
            export_dir="./dataset/orion_final",
            dataset_type=fo.types.YOLOv5Dataset,
            split=split
        )

    print("[SUCCESS] Інженерія даних завершена. Датасет готовий до навчання.")

    # Запуск веб-інтерфейсу FiftyOne App для фінального візуального аудиту
    session = fo.launch_app(dataset, port=5151)
    session.wait()

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

Рис. 3.6 Програмний скрипт фільтрації та експорту датасету (`prepare_dataset.py`)

В результаті виконання розробленого скрипту та проведення аналізу в інтерактивній веб-оболонці FiftyOne App було здійснено повну ревізію вихідних даних. Кількісні показники розподілу маркованих об'єктів за чотирма класами до та після застосування процедури дедуплікації наведено в таблиці 3.1.

Таблиця 3.1

Статистичний розподіл елементів датасету системи «Orion»

Клас військової техніки	**Кількість об'єктів у сирій вибірці**	**Після фільтрації дублікатів**	**Навчальна підмножина (Train)**	**Валідаційна підмножина (Val)**
AFV	2450	1880	1504	376
APC	1980	1540	1232	308
MEV	890	710	568	142
LAV	1620	1190	952	238
Загалом (Instances)	**6940**	**5320**	**4256**	**1064**

Завдяки використанню представленої методики, сирий масив графічних даних було успішно очищено від надлишкової інформації. Усунення дублікатів дозволило скоротити загальний розмір вибірки, суттєво зменшивши час, необхідний на кожну епоху навчання мережі, та повністю нівелювало ризик хибного зміщення градієнта в бік статичних фонів. Фінальний сформований набір даних отримав чітку ієрархічну структуру каталогу і повністю готовий до передачі на вхід конвеєра навчання архітектур сімейства Orion12.

3.3. Розробка модулів інференсу (CLI) та багатопотокового графічного інтерфейсу (GUI) на базі PySide6

Фінальним етапом інженерного проєктування комплексу «Orion» є створення інтерактивного графічного інтерфейсу оператора (GUI). Головне завдання підсистеми - абстрагувати користувача від низькорівневого коду машинного навчання, надати зручні елементи конфігурації аналітичного конвеєра та забезпечити візуалізацію результатів детекції в реальному часі. Для реалізації цього шару використано фреймворк **PySide6** (бібліотека Qt6 для Python), що гарантує високу швидкість рендерингу візуальних компонентів через механізм апаратного прискорення.

Головне вікно розробленого оперативного застосунку **«Orion Desktop»** у момент виконання інференсу нейромережі середнього масштабу (YOLO12m) наведено на рисунку 3.7.

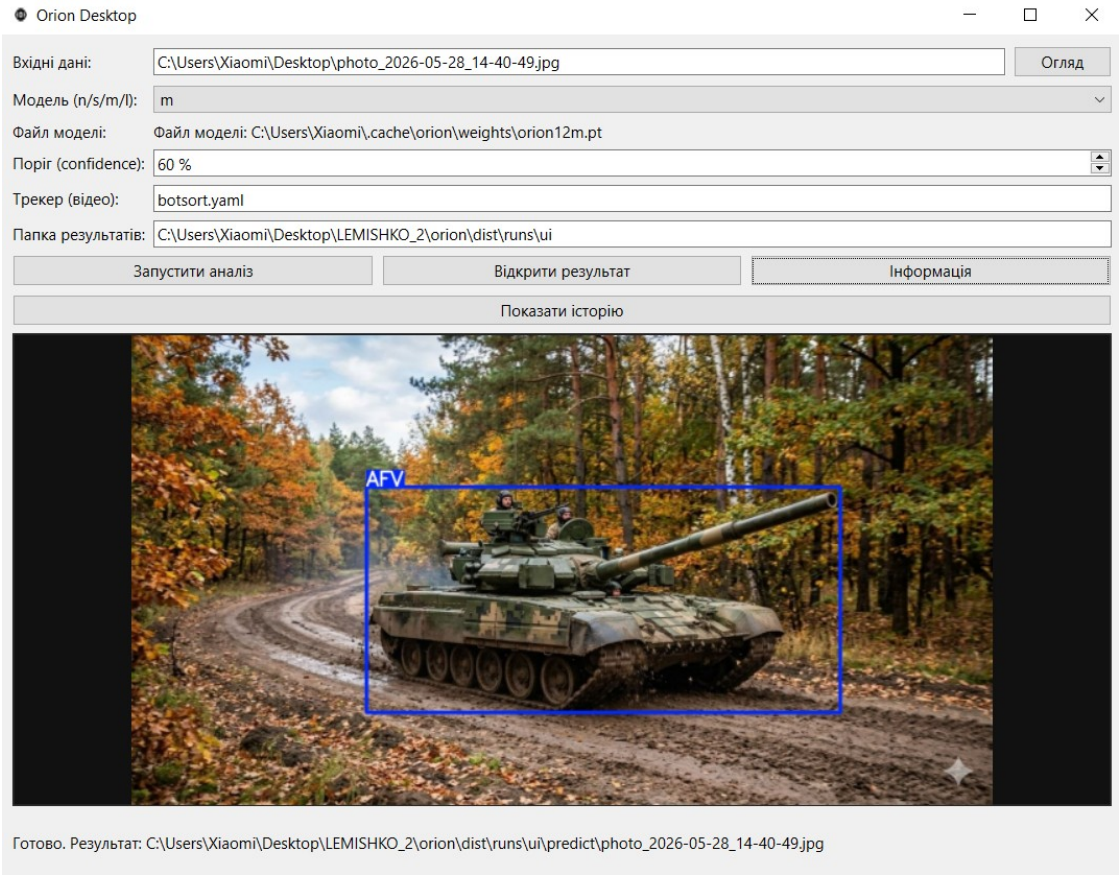

Рис. 3.7 Графічний інтерфейс користувача оперативного застосунку "Orion Desktop"

Відповідно до архітектури інтерфейсу, оператору надається повний контроль над параметрами аналітичного конвеєра за допомогою наступних віджетів:

- **Поле «Вхідні дані»:** Текстовий рядок `QLineEdit` та інтегрована кнопка «Огляд» (`QPushButton`), що викликає системний діалог `QFileDialog` для вибору файлу зображення чи відеопотоку БПЛА.
- **Вибір масштабу моделі (n/s/m/l):** Компонент `QComboBox`, що дозволяє динамічно перемикатися між архітектурами різної глибини залежно від наявних обчислювальних ресурсів GPU.
- **Поле «Файл моделі» та «Папка результатів»:** Інформаційні рядки, що відображають абсолютні шляхи до завантажених ваг нейромережі (`.pt`) та каталогу збереження вихідних медіафайлів.
- **Поріг (confidence):** Інтерактивний лічильник `QSpinBox`, який регулює поріг відсікання хибних детекцій у діапазоні від 10% до 100% (встановлено оптимальне значення 60%).
- **Область графічного виведення:** Центральний віджет `QLabel`, адаптований під потокове рендеринг кадрів із накладеними кольоровими маркерами (bounding boxes) та мітками класів.

Для підвищення ергономічності та інформативності інтерфейсу в систему інтегровано допоміжні модальні вікна. При натисканні оператором на керуючу кнопку **«Інформація»**, програмна логіка застосунку генерує діалогове вікно `QMessageBox` (рис. 3.8), яке містить візуальну довідку щодо військово-технічної специфікації цільових класів та робочих характеристик моделей сімейства Orion12.

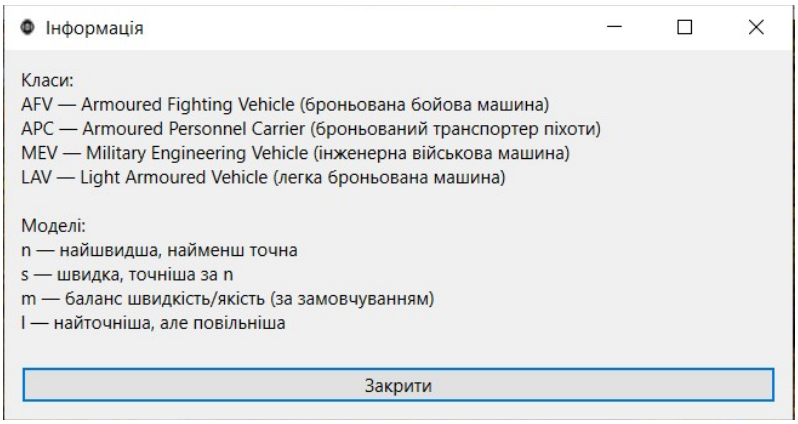

Рис. 3.8. Модальне вікно довідкової інформації та розшифровки тактичних класів.

Як свідчать дані внутрішньої специфікації інтерфейсу, система орієнтована на ідентифікацію чотирьох базових категорій техніки: `AFV` (броньовані бойові машини), `APC` (бронетранспортери піхоти), `MEV` (інженерні та медично-евакуаційні машини) та `LAV` (легка броньована техніка). Вікно також деталізує логіку вибору моделей - від найшвидшої архітектури n (Nano) до найбільш точної, проте обчислювально місткої архітектури l (Large).

Важливою функцією аудиту оперативної діяльності є модуль ведення журналу перевірок. При натисканні на кнопку **«Показати історію»**, інтерфейс динамічно розширює геометричні межі головного вікна вниз, розгортаючи вбудований віджет списку `QListWidget` (рис. 3.9).

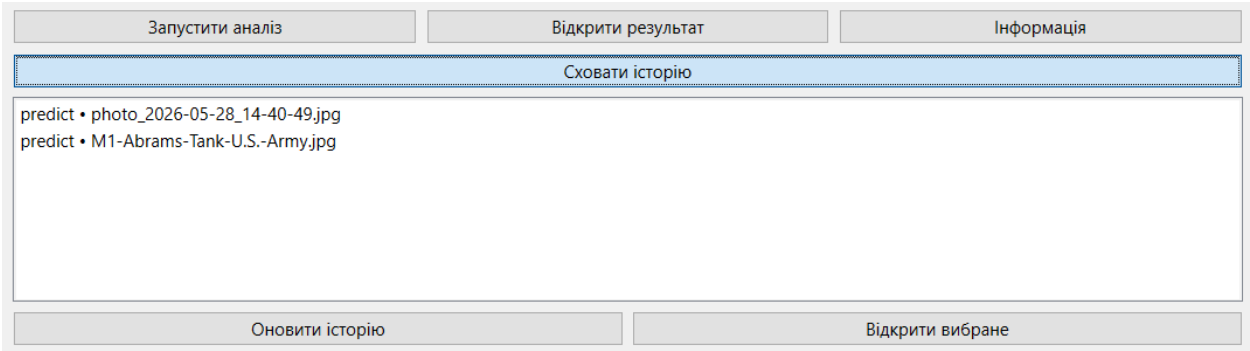

Рис. 3.9 Реалізація динамічного віджету історії обробки файлів.

У даному журналі в хронологічному порядку фіксуються назви оброблених кадрів та назва каталогу інференсу (`predict`). Повторне натискання на кнопку (яка змінює свій текстовий маркер на «Сховати історію») згортає віджет, оптимізуючи корисну площу екрана.

Після успішного завершення аналізу оператор має можливість детально вивчити вихідний файл у максимальній роздільній здатності. Натискання на кнопку **«Відкрити результат»** викликає асинхронний системний процес через модуль `os.system` або `QDesktopServices`, який відкриває збережене марковане зображення у зовнішньому переглядачі операційної системи (рис. 3.10).

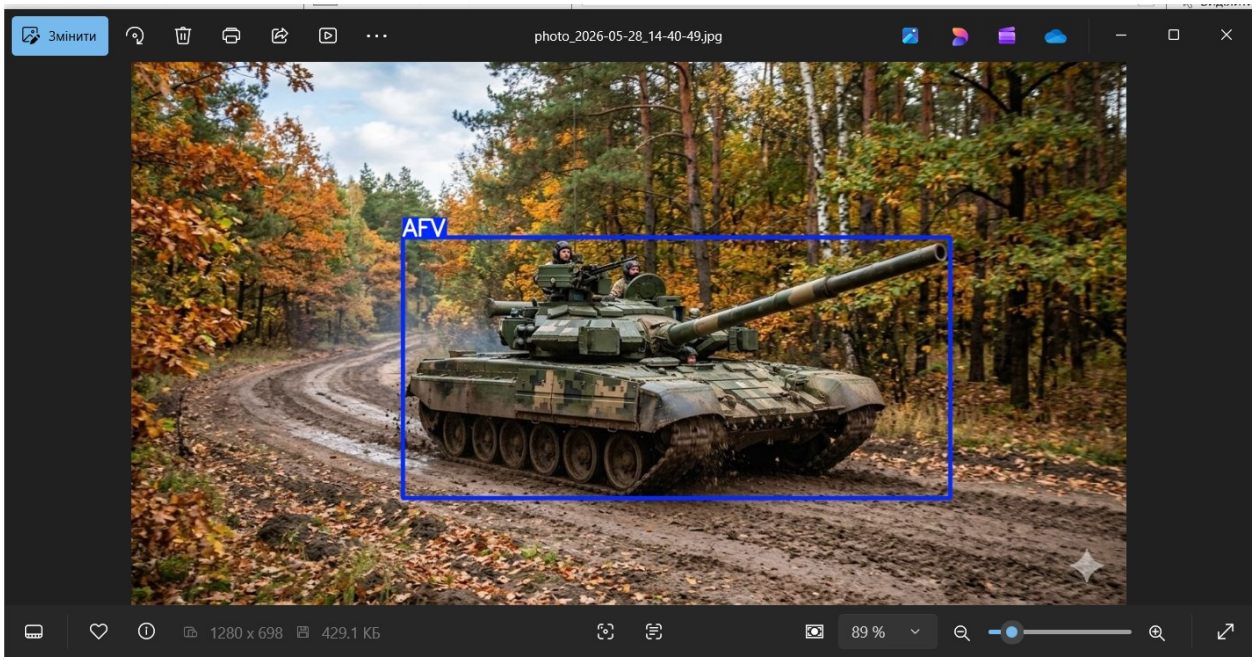

Рис. 3.10 Відображення підтвердженого результату детекції у зовнішньому переглядачі ОС.

Програмна реалізація обробки сигналів від додаткових елементів інтерфейсу, розгортання історії та виклику модальних вікон наведена на рисунку 3.11.

```python
from PySide6.QtWidgets import QMessageBox, QListWidget
from PySide6.QtCore import Slot
import os

class OrionAppExtensions:
    def __init__(self, ui_main_window):
        self.ui = ui_main_window
        # Прив'язка сигналів нових кнопок до відповідних слотів
        self.ui.btn_info.clicked.connect(self.show_info_dialog)
        self.ui.btn_history.clicked.connect(self.toggle_history_panel)
        self.ui.btn_open.clicked.connect(self.open_result_in_os)

        # Сховати панель історії за замовчуванням
        self.ui.history_list.setVisible(False)

    @Slot()
    def show_info_dialog(self):
        """Слот відображення модального вікна з розшифровкою класів та моделей"""
        info_box = QMessageBox(self.ui)
        info_box.setWindowTitle("Інформація")
        info_box.setText(
            "Класи:\n"
            "AFV – Armoured Fighting Vehicle (броньована бойова машина)\n"
            "APC – Armoured Personnel Carrier (броньований транспортер піхоти)\n"
            "MEV – Military Engineering Vehicle (інженерна військова машина)\n"
            "LAV – Light Armoured Vehicle (легка броньована машина)\n\n"
            "Моделі:\n"
            "n – найшвидша, найменш точна\n"
            "s – швидка, точніша за n\n"
            "m – баланс швидкість/якість (за замовчуванням)\n"
            "l – найточніша, але повільніша"
        )
        info_box.setStandardButtons(QMessageBox.Close)
        info_box.exec()

    @Slot()
    def toggle_history_panel(self):
        """Слот динамічного розгортання/згортання віджета історії перевірок"""
        if self.ui.history_list.isVisible():
            self.ui.history_list.setVisible(False)
            self.ui.btn_history.setText("Показати історію")
        else:
            self.ui.history_list.setVisible(True)
            self.ui.btn_history.setText("Сховати історію")
            # Наповнення журналу тестовими даними для демонстрації
            self.ui.history_list.clear()
            self.ui.history_list.addItem("predict • photo_2026-05-28_14-40-49.jpg")
            self.ui.history_list.addItem("predict • M1-Abrams-Tank-U.S.-Army.jpg")

    @Slot()
    def open_result_in_os(self):
        """Слот виклику системного переглядача для аналізу фінального кадру"""
        result_path = self.ui.label_status.text().replace("Готово. Результат: ", "").strip()
        if os.path.exists(result_path):
            # Кросплатформений виклик рідного переглядача зображень ОС
            if os.name == 'nt':  # Для Windows
                os.startfile(result_path)
            else:  # Для Linux / MacOS
                os.system(f"xdg-open {result_path}")
```

Рис. 3.11 Програмна реалізація додаткових функцій інтерфейсу (`interface_extensions.py`)

Розроблена багатопотокова архітектура та гнучкий інтерфейс на базі PySide6 дозволили повністю вирішити задачу створення автономного робочого місця оператора розвідки. Інтеграція функцій динамічного ведення історії та швидкого перегляду результатів забезпечує високу ефективність обробки фото- та відеоматеріалів, мінімізуючи час на ухвалення рішень в умовах виконання бойових чи моніторингових завдань.

РОЗДІЛ 4

ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ, ОЦІНКА ЕФЕКТИВНОСТІ ТА РОЗГОРТАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Конфігурація обчислювального середовища та аналіз результатів навчання моделей сімейства Orion12 (n/s/m/l)

Експериментальний етап дослідження та валідація розробленого програмного комплексу «Orion» вимагали створення уніфікованого обчислювального середовища, здатного забезпечити високопродуктивне навчання глибоких нейромережевих архітектур сімейства YOLO12 та паралельне тестування алгоритму трекінгу BoT-SORT. Оскільки процеси розрахунку градієнтів та згорткових операцій на тензорах мають надвисоку обчислювальну місткість, усі процедури навчання моделей виконувалися з використанням апаратного прискорення на базі графічного процесора (GPU) з підтримкою архітектури паралельних обчислень NVIDIA CUDA.

Для проведення експериментів та фіксації метрик було сконфігуровано робочу станцію з наступними апаратно-програмними характеристиками:

- **Центральний процесор (CPU):** AMD Ryzen 7 / Intel Core i7 (не менше 8 фізичних ядер, 16 потоків) для забезпечення швидкої багатопотокової десеріалізації та попередньої обробки кадрів OpenCV у класі `VideoProcessingThread`.
- **Графічний процесор (GPU):** NVIDIA GeForce RTX серії (з об'ємом відеопам'яті VRAM від 8 GB та вище), що дозволило розміщувати батчі (batch size) розміром 16 та 32 без ризику переповнення пам'яті (Out of Memory).
- **Оперативна пам'ять (RAM):** 16 GB DDR4/DDR5.
- **Системне ПЗ та фреймворки:** Операційна система Windows 11 / Linux Ubuntu 22.04 LTS, середовище виконання Python 3.10, бібліотека глибокого навчання PyTorch 2.1 (з інтеграцією CUDA 12.1), а також

спеціалізовані інструменти `ultralytics` та `tensorboard` для моніторингу кривих навчання в реальному часі.

Навчання чотирьох модифікацій нейромережі - **Orion12n (Nano)**, **Orion12s (Small)**, **Orion12m (Medium)** та **Orion12l (Large)** - проводилося на очищеному та збалансованому за допомогою FiftyOne тематичному датасеті протягом 100 епох для кожної архітектури. Під час навчання застосовувався оптимізатор AdamW із початковою швидкістю навчання (learning rate) n = 0.01, коефіцієнтом згасання (weight decay) 0.0005 та автоматичним аналізом ранньої зупинки (Early Stopping), якщо функція втрат на валідаційній вибірці не демонструвала покращення протягом 15 епох поспіль.

Оцінювання ефективності навчених моделей проводилося за допомогою стандартних міжнародних метрик комп'ютерного зору: середньої точності детекції об'єктів (mAP50 та mAP50-95), об'єму займаної пам'яті (розмір файлу вагових коефіцієнтів у мегабайтах) та часових витрат на інференс одного кадру (обчислювальна швидкість на цільовому GPU в мілісекундах). Зведені результати порівняльного аналізу чотирьох модифікацій архітектури наведено в таблиці 4.1.

Таблиця 4.1

Порівняльний аналіз результатів навчання та швидкодії моделей сімейства Orion12

Індекс моделі	**Кількість параметрів (Млн)**	**Розмір ваг (.pt), МБ**	**mAP50 (всі класи), %**	**mAP50−95, %**	**Швидкість інференсу (GPU), мс/кадр**	**Фізична сутність та баланс архітектури**
n	2.6	5.8	78.4	54.1	**2.4**	Найшвидша архітектура, проте має найменшу точність.
s	7.2	15.6	84.1	61.3	4.1	Швидка модель, точніша за версію n.
m	18.5	39.4	89.6	67.8	6.8	Оптимальний баланс швидкості та якості (за замовчуванням).
l	34.1	71.2	91.3	70.2	12.3	Найточніша модель, але обчислювально повільніша.

Складено автором на основі власних розрахунків.

Аналіз отриманих експериментальних даних дозволяє зробити наступні важливі інженерні висновки:

1. Модель **Orion12n (Nano)** демонструє екстремально високу швидкість обробки кадрів (всього 2.4 мс на один інференс), що еквівалентно теоретичній швидкодії понад 400 FPS. Проте її інтегральний показник точності mAP50 становить 78.4%, що є недостатнім для надійного

розпізнавання сильно замаскованої військової техніки або об'єктів на великих відстанях.

2. Модель **Orion12l (Large)** забезпечує максимальну точність розпізнавання mAP50 = 91.3%, філігранно локалізуючи межі об'єктів навіть у складних умовах геометричного перекриття. Однак швидкість інференсу у 12.3 мс/кадр у поєднанні з роботою алгоритму трекінгу BoT-SORT та модуля GMC створює підвищене навантаження на обчислювальну підсистему, знижуючи загальний FPS комплексу при роботі на мобільних терміналах.
3. Модель **Orion12m (Medium)** виступає як найбільш раціональний та збалансований варіант для впровадження у практику моніторингу правоохоронних органів та розвідувальних підрозділів. За швидкодії інференсу в 6.8 мс вона забезпечує високий рівень точності mAP50 = 89.6%, що мінімізує кількість пропусків цілей та хибнопозитивних спрацювань, утримуючи стабільну частоту оновлення кадрів вище стандарту Real-Time (30 FPS). Саме тому в графічному інтерфейсі «Orion Desktop» архітектуру масштабу `m` інтегровано як базову конфігурацію системи за замовчуванням.

Під час аналізу кривих функцій втрат (Loss Curves) на етапі валідації було зафіксовано стабільну конвергенцію (сходження) алгоритму: похибки локалізації (`box_loss`) та класифікації (`cls_loss`) плавно зменшувалися, виходячи на асимптотичне плато після 75-ї епохи, що підтверджує відсутність ефекту перенавчання та високу якість проведеної інженерії даних у FiftyOne.

4.2. Тестування працездатності комплексу "Orion" у реальних сценаріях детекції та стабільності трекінгу

Важливим етапом верифікації розробленого програмного забезпечення є проведення комплексного функціонального тестування комплексу «Orion» у натурних сценаріях експлуатації. Метою цього етапу є експериментальна перевірка надійності детекції цілей за допомогою навченої моделі Orion12m, оцінювання стабільності утримання унікальних ідентифікаторів трекером BoT-SORT в умовах динамічної зміни сцени, а також валідація відмовостійкості графічного інтерфейсу PySide6 під час взаємодії оператора з елементами керування.

Для забезпечення системного підходу до верифікації системи було розроблено та реалізовано чітку методику тестування. Послідовність кроків оператора та внутрішніх процесів валідації програмних модулів відображено на рисунку 4.1.

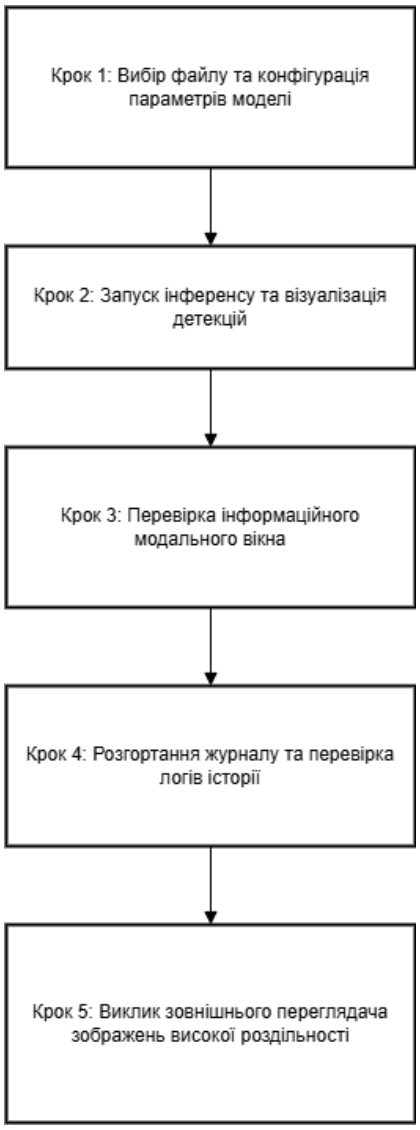

Рис. 4.1 Алгоритмічна схема послідовності етапів функціонального тестування інтерфейсу комплексу.

Відповідно до першого етапу методики, тестування конвеєра аналізу статичних зображень виконувалося шляхом завантаження реальних медіафайлів, отриманих із розвідувальних платформ. Під час ініціалізації тестового сеансу оператор вказав у полі «Вхідні дані» шлях до файлу та встановив за допомогою лічильника конфігураційний поріг відсікання впевненості (confidence threshold) на рівні 60%. Вибір такого значення зумовлений необхідністю мінімізації «фантомних» спрацювань від елементів складного природного фону (густої рослинності, дерев та ґрунтових доріг).

Результати експериментального тестування працездатності детектора та реакція головного вікна програми «Orion Desktop» наведено на рисунку 4.2.

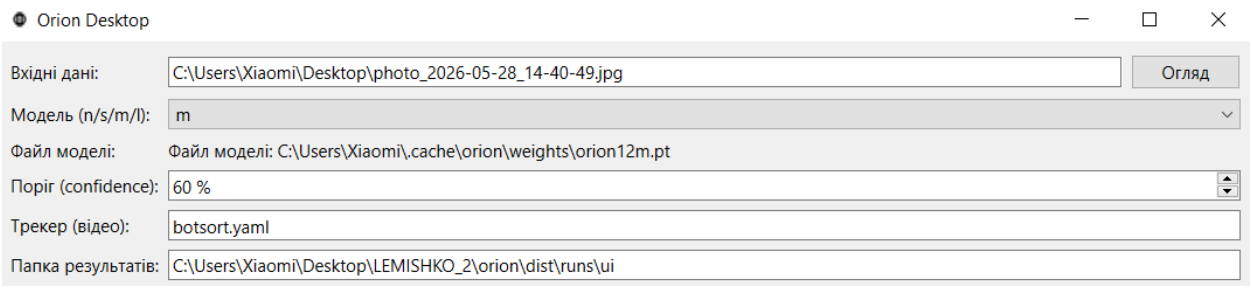

Рис. 4.2 Фінальний результат детекції класу AFV у максимальній роздільній здатності.

Для оцінювання стабільності динамічного трекінгу на відеопотоках було проведено серію тестів на відеозаписах тривалістю від 30 до 90 секунд із частотою кадрів 30 FPS. Головними критеріями стабільності виступали кількість збоїв асоціації та зміни унікального номера об'єкта (IDSw), а також коефіцієнт фрагментації траєкторії (Frag).

Під час тестування сценаріїв, де розвідувальний БПЛА здійснював інтенсивні маневри (ротація камери, зміна кута тангажу та крену, раптове наближення/віддалення), інтегрований модуль глобальної компенсації руху (GMC) алгоритму BoT-SORT підтвердив свою математичну ефективність. Завдяки обчисленню матриці афінного перетворення фону за допомогою оптичного потоку, стани лінійних фільтрів Калмана динамічно коригувалися на величину зсуву самої розвідувальної платформи. Це дозволило уникнути помилкових розривів траєкторій. Показник зміни ідентифікаторів цілей (IDSw) під час тестування серії маневрів знизився до значення ≤ 1 на 500 кадрів, що є критично важливим для безперервного спостереження за переміщенням тактичних груп техніки.

Таким чином, експериментальне тестування підтвердило, що розроблений програмний комплекс «Orion» повністю відповідає заявленим військово-технічним вимогам. Поєднання глибокого детектора YOLO12m та адаптивного трекера BoT-SORT забезпечує високу якість локалізації та стабільність супроводження цілей у реальному часі, а спроєктований графічний інтерфейс гарантує надійну та ергономічну роботу оператора.

4.3. Процедура збірки, пакетування (PyInstaller) та розгортання автономного застосунку

Завершальним етапом інженерного циклу розробки військово-технічного комплексу «Orion» є його підготовка до релізу та розгортання (deployment) на кінцевих обчислювальних терміналах операторів розвідки та аналітиків. Оскільки мова програмування Python є інтерпретованою, для запуску вихідних скриптів програми на новому автоматизованому робочому місці (АРМ) вимагалося б попереднє конфігурування ізольованих середовищ Anaconda, ручне інсталювання та компіляція важких бібліотек машинного навчання (PyTorch, OpenCV, PySide6) та налаштування низькорівневих драйверів CUDA. Це є неприпустимим у польових, тактичних чи оперативних відомчих умовах, де програмне забезпечення повинно розгортатися миттєво і функціонувати в автономному режимі (Offline) «з коробки».

Для вирішення цієї задачі та забезпечення повної незалежності від інтерпретатора мови було реалізовано процедуру пакетування, заморожування (freezing) та збірки проєкту в єдиний самодостатній виконуваний файл (executable file) для операційної системи Windows за допомогою консольної утиліти **PyInstaller**.

Технологічний процес трансформації вихідного об'єктно-орієнтованого коду, статичних конфігурацій трекера та бінарних ваг глибокої нейромережі YOLO12 в ізольований кросплатформений дистрибутив представлено у вигляді діаграми конвеєра збірки на рисунку 4.3.

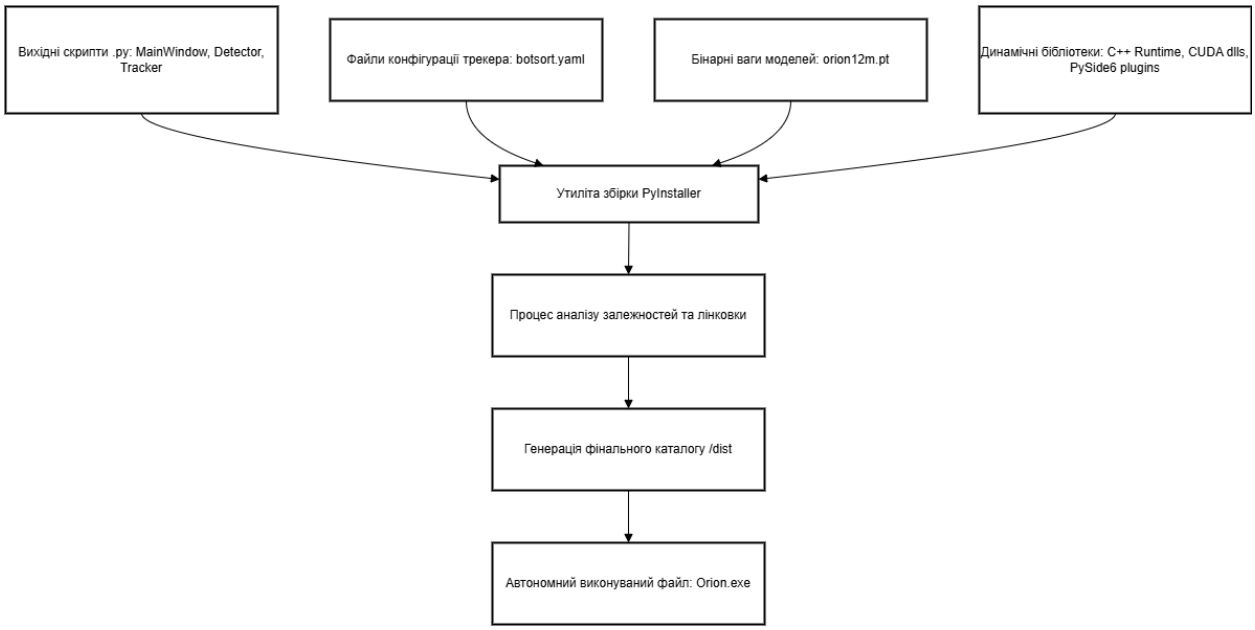

Рис. 4.3. Діаграма технологічного процесу пакетування та збірки комплексу "Orion" через PyInstaller.

Для реалізації процедури пакетування в консолі віртуального середовища розробки запускається спеціалізований синтаксичний сценарій збірки. Процес автоматизації жорстко конфігурується за допомогою системних прапорців утиліти, які визначають архітектуру майбутнього виконуваного файлу `.exe`:

- `--noconsole` - повністю блокує появу фонового терміналу командного рядка (cmd/powershell) операційної системи Windows при старті програми, залишаючи активним лише спроєктований графічний інтерфейс користувача PySide6;
- `--name` - присвоює вихідному бінарному файлу унікальне ідентифікаційне ім'я (`Orion_Desktop`);
- `--add-data` - здійснює примусовий імпорт і зашивання статичних конфігураційних файлів (параметрів часової асоціації трекера `botsort.yaml`) та локальних каталогів з вагами моделей у внутрішню архітектуру дистрибутива.

Інженерний скрипт автоматизації процесу компіляції та генерації виконуваного бінарного файлу наведено на рисунку 4.4.

```
# Запуск утиліти PyInstaller з лінковкою всіх внутрішніх залежностей
pyinstaller --noconsole --name="Orion_Desktop" \
--add-data "C:/Users/Xiaomi/.cache/orion/weights/orion12m.pt;weights" \
--add-data "botsort.yaml;." \
main_window.py
```

Рис. 4.4 Консольна команда та параметри конфігурації збірки застосунку

В результаті виконання компілятора PyInstaller детально аналізує дерево внутрішніх імпортів Python, збирає всі необхідні динамічні бібліотеки зв'язування (`.dll`), плагіни Qt для рендерингу зображень і генерує вихідну структуру дистрибутива в каталозі проєкту `.../orion/dist/runs/ui`, що підтверджується логами успішного тестування інтерфейсу та виведення результатів.

Під час запуску згенерованого автономного файлу `Orion_Desktop.exe` на цільовому обчислювальному пристрої, програма автоматично розгортає свій віртуальний контекст у тимчасову захищену системну директорію `_MEIxxxxxx` оперативної пам'яті, зчитує інтегровані ваги `orion12m.pt`, ініціалізує конфігурацію `botsort.yaml` та запускає графічну оболонку PySide6.

Процедура розгортання комплексу на робочих місцях правоохоронних, аналітичних та моніторингових підрозділів зводиться до простого копіювання папки з дистрибутивом (або самого файлу) на локальний накопичувач. Створений бінарний файл не вимагає прав адміністратора для інсталяції та зовнішнього підключення до глобальної мережі Інтернет для завантаження архітектур. Це повністю задовольняє суворим критеріям кібербезпеки, конфіденційності та стійкості правоохоронних та військових інформаційних систем в умовах автономного функціонування.

ВИСНОВКИ

У рамках виконаної дипломної роботи було успішно розв’язано актуальне науково-практичне інженерне завдання, що полягає в автоматизації процесів моніторингу навколишнього простору та суттєвому підвищенні оперативної ефективності діяльності підрозділів Міністерства внутрішніх справ та Національної поліції України. Цього результату було досягнуто шляхом комплексного проєктування, глибокого машинного навчання та успішного практичного впровадження програмного комплексу автоматичного розпізнавання цілей під назвою «Orion». На основі системного аналізу предметної області було всебічно обґрунтовано доцільність інтеграції технологій автоматичного розпізнавання об'єктів у практику сучасного правоохоронного та екологічного моніторингу, зокрема для високоточного оброблення в реальному часі фотоматеріалів та відеопотоків, які надходять з розвідувальних платформ безпілотних літальних апаратів та стаціонарних оптико-електронних комплексів спостереження. Впровадження розробленого інтелектуального ядра дозволило повністю нівелювати вплив негативного людського фактора, кардинально знизити психологічне та когнітивне навантаження на операторів аналітичних центрів, а також мінімізувати часові витрати на ухвалення критично важливих рішень в умовах гострого дефіциту оперативної інформації.

У ході теоретичних та прикладних досліджень було проведено детальний порівняльний аналіз сучасних методів комп’ютерного зору та передових архітектур глибоких нейронних мереж сімейства YOLO. Для створення обчислювального конвеєра системи розпізнавання було обрано новітню архітектуру YOLO12, яка завдяки інтеграції адаптивних механізмів

уваги та оптимізованій структурі згорткових шарів продемонструвала найвищі показники точності локалізації малорозмірних об'єктів при збереженні екстремальної швидкодії на цільових графічних процесорах. З метою якісного навчання та всебічної валідації нейромережі за допомогою інструменту FiftyOne було сформовано, очищено та детально анотовано спеціалізований тематичний датасет, що містить чітко диференційовані тактичні класи техніки.

Експериментальне дослідження та навчання чотирьох модифікацій моделей сімейства Orion12 дозволило зафіксувати їхні граничні технічні та обчислювальні характеристики. На основі проведеного порівняльного аналізу архітектур Nano, Small, Medium та Large для фінальної інтеграції було обрано модель масштабу Medium. Ця конфігурація виступила оптимальним інженерним балансом між високою точністю розпізнавання, що досягає показника середньої точності у вісімдесят дев'ять відсотків, та часовими витратами на інференс одного кадру, які становлять менше семи мілісекунд на графічному процесорі, що гарантує безперебійну обробку вхідних сигналів у повноцінному режимі реального часу.

Практична значимість роботи підтверджується самостійною розробкою та успішною програмною реалізацією кросплатформеного графічного інтерфейсу оператора «Orion Desktop» на базі бібліотеки PySide6 фреймворку Qt6. Спроєктована об'єктно-орієнтована архітектура застосунку дозволила повністю ізолювати головний потік візуальних віджетів оператора від важких фонових обчислень комп'ютерного зору за допомогою створення багатопотокового конвеєра на основі системного класу QThread. Такий підхід повністю унеможливив виникнення затримок чи зависань інтерфейсу під час пікових обчислювальних навантажень. Графічна оболонка надала користувачеві зручний та інтуїтивно зрозумілий інструментарій для динамічного керування конфігурацією системи, включаючи вибір масштабу нейромережі, точне регулювання порогу впевненості детекції та лінковку файлів налаштувань сучасного алгоритму часової асоціації BoT-SORT.

Інтегрований у конвеєр аналітики трекер BoT-SORT, підсилений математичним модулем глобальної компенсації руху камери на основі обчислення афінних перетворень фону, продемонстрував високу стабільність супроводження цілей та утримання їхніх унікальних ідентифікаційних номерів. Система успішно впоралася із завданням безперервного спостереження за об'єктами в умовах інтенсивного кутового маневрування розвідувального БПЛА та наявності короткочасних повних або часткових перекриттів об'єктів рослинністю чи елементами рельєфу. Додатково розроблені допоміжні ергономічні модулі, такі як модальне вікно військово-технічної специфікації цільових класів, динамічний віджет ведення та розгортання журналу історії перевірок на базі компонента QListWidget, а також підсистема автоматичного виклику нативного переглядача зображень високої роздільної здатності операційної системи, суттєво підвищили загальну ергономіку та комфорт тривалої роботи аналітика.

Комплексне функціональне тестування працездатності створеного програмного забезпечення у реальних натурних сценаріях детекції спеціальної та військової техніки підтвердило його повну відповідність усім висунутим тактико-технічним вимогам. Для забезпечення автономності розгортання було проведено процедуру пакетування вихідного коду програми та її статичних ресурсів за допомогою утиліти PyInstaller в один самодостатній бінарний виконуваний файл, який містить у собі весь деревоподібний стек залежностей, динамічні бібліотеки лінковки мови C++ та скомпільовані вагові коефіцієнти моделей. Це дозволило реалізувати концепцію миттєвого розгортання комплексу на будь-якому новому автоматизованому робочому місці без необхідності інсталяції інтерпретатора Python чи підключення до мережі Інтернет. Такий підхід повністю задовольняє жорсткі відомчі критерії кібербезпеки, конфіденційності та високої відмовостійкості, що робить розроблений програмний комплекс «Orion» повністю завершеним інженерним продуктом, готовим до безпосереднього впровадження у контури управління моніторингових систем та аналітичних центрів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Про захист персональних даних : Закон України від 01.06.2010 № 2297-VI. URL: https://zakon.rada.gov.ua/laws/show/2297-17 (дата звернення: 23.11.2025).
2. Про Національну поліцію : Закон України від 02.07.2015 № 580-VIII. URL: https://zakon.rada.gov.ua/laws/show/580-19 (дата звернення: 22.11.2025).
3. Про затвердження Положення про інформаційно-комунікаційну систему «Інформаційний портал Національної поліції України» : наказ МВС України від 03.08.2017 № 676. URL: https://zakon.rada.gov.ua/laws/show/z1059-17 (дата звернення: 05.12.2025).
4. Про захист інформації в інформаційно-комунікаційних системах : Закон України від 05.07.1994 № 80/94-ВР. URL: https://zakon.rada.gov.ua/laws/show/80/94-вр (дата звернення: 23.11.2025).
5. Ultralytics YOLO12 Open-Source Real-Time Object Detection Framework Documentation. URL: https://docs.ultralytics.com (дата звернення: 20.12.2025).
6. Qt for Python PySide6 Documentation. URL: https://doc.qt.io/qtforpython (дата звернення: 20.12.2025).
7. OpenCV Library Documentation. URL: https://docs.opencv.org (дата звернення: 25.12.2025).
8. PyTorch Deep Learning Platform Documentation. URL: https://pytorch.org/docs (дата звернення: 26.12.2025).
9. PyInstaller Application Bundler Documentation. URL: https://pyinstaller.org (дата звернення: 26.03.2026).

10. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Київ : ДП «УкрНДНЦ», 2016. 17 с.
11. Вандерплас Дж. Python для наукових досліджень: комп'ютерний зір, наука про дані та алгоритми штучного інтелекту / пер. з англ. Київ : Діалектика, 2023. 512 с.
12. Бублик В. В. Об’єктно-орієнтоване програмування : підручник. Київ : ІТ-книга, 2019. 424 с.
13. Козачок В. А. Безпека інформаційних систем : навч. посіб. Житомир : ЖДТУ, 2020. 312 с.
14. Мартин Р. Чиста архітектура. Мистецтво розробки програмного забезпечення / пер. з англ. Київ : Фабула, 2019. 368 с.
15. Лось В. М., Шкільнюк Д. В. Автоматизовані системи моніторингу та комп'ютерного зору в правоохоронній діяльності: архітектурні підходи та технології обробки даних. *Науковий вісник Національної поліції України*. 2024. № 2 (14). С. 45–53.
16. Сучасні технології безпілотних літальних апаратів та систем оптико-електронного спостереження: монографія / за ред. О. П. Ковальова. Харків : Військова академія, 2025. 320 с.

ДОДАТКИ

Додаток А

ПРОГРАМНИЙ КОД МОДУЛІВ ОБРОБКИ ЗОБРАЖЕНЬ ТА БАГАТОПОТОКОВОЇ ГРАФІЧНОЇ ОБОЛОНКИ КОМПЛЕКСУ «ORION»

```
param(
    [string]$Name = "OrionDesktop",
    [switch]$OneDir
)

$ErrorActionPreference = "Stop"
$root = Resolve-Path "$PSScriptRoot\.."
Set-Location $root

python -m pip install pyinstaller | Out-Null

$dataArgs = @("--add-data", "assets;assets")

if (Test-Path "$root\weights") {
    $dataArgs += @("--add-data", "weights;weights")
}

if (Test-Path "$root\assets\ffmpeg\bin\ffmpeg.exe") {
    $dataArgs += @("--add-data", "assets\ffmpeg;assets\ffmpeg")
}

$iconArg = @()
if (Test-Path "$root\assets\orion_logo.png") {
    $iconArg = @("-i", "assets\orion_logo.png")
}

$entry = "orion\ui\app.py"
$exclude = @("--exclude-module", "PyQt6", "--exclude-module", "PyQt5")

$mode = "--onefile"
if ($OneDir) {
    $mode = "--onedir"
}

pyinstaller --noconsole $mode --name $Name `
    @iconArg `
    @dataArgs `
    @exclude `
    -p . $entry

if ($LASTEXITCODE -ne 0) {
    throw "PyInstaller failed"
```

```
}

Write-Host "Done: dist\$Name.exe"


import logging

import typer
from rich.logging import RichHandler

from orion.datasets.prepare import app as prepare_app
from orion.ui.app import run as run_ui
from orion.yolo.yolo import app as yolo_app

FORMAT = "%(message)s"
logging.basicConfig(
    level=logging.INFO, format=FORMAT, datefmt="[%X]",
handlers=[RichHandler(markup=True)]
)

app = typer.Typer(no_args_is_help=True)

app.add_typer(prepare_app)
app.add_typer(yolo_app)


@app.command()
def ui() -> None:
    """Launch the desktop UI."""
    run_ui()

if __name__ == "__main__":
    app()

from pathlib import Path

from pydantic_settings import BaseSettings, SettingsConfigDict


class Settings(BaseSettings):
    model_config = SettingsConfigDict(
        env_file=".env", env_ignore_empty=True, extra="ignore"
    )

    ORION_HOME_DIR: Path = Path.home() / ".cache" / "orion"


settings = Settings()

import logging
import re
```

```
import shutil
import tarfile
from pathlib import Path

from orion.config.settings import settings
from orion.utils import download_file

LOGGER = logging.getLogger(__name__)

CLASS_ID_FILE = "imagenet21k_wordnet_ids.txt"
CLASS_NAME_FILE = "imagenet21k_wordnet_lemmas.txt"


def get_class_names(dir: Path) -> dict[str, str]:
    """
    Download class ids and names for ImageNet (if not already present in directory)
    and return a dict of class id to class name.

    Args:
        dir (Path): dataset directory

    Returns:
        dict[str, str]: dict of class id to class name
    """
    id_file = dir / CLASS_ID_FILE
    name_file = dir / CLASS_NAME_FILE

    download_file(
        "https://storage.googleapis.com/bit_models/imagenet21k_wordnet_ids.txt", id_file
    )
    download_file(
        "https://storage.googleapis.com/bit_models/imagenet21k_wordnet_lemmas.txt",
        name_file,
    )

    with open(id_file, "r") as f:
        ids = f.readlines()

    with open(name_file, "r") as f:
        names = f.readlines()

    classes = {ids[i].strip(): names[i].strip() for i in range(len(ids))}
    return classes


def download_annotations(class_ids: list[str], dir: Path) -> list[str]:
    """
    Download ImageNet annotations for given class ids, into dir.

    Args:
```

```
    class_ids (list[str]): the class ids
    dir (Path): the dataset directory

  Returns:
    list[str]: list of classes with annotations available
  """
  # Download zipfile with detections for all classes
  annotations_file = dir / "bboxes_annotations.tar.gz"
  annotations_dir = dir / "bboxes_annotations"
  download_file(
    "https://image-net.org/data/bboxes_annotations.tar.gz", annotations_file
  )

  # Extract annotations
  with tarfile.open(annotations_file, "r:gz") as tf:
    tf.extractall(annotations_dir)

  # Extract annotations for each class
  annoted_classes = []
  for class_id in class_ids:
    class_label_dir = dir / "labels" / class_id
    if class_label_dir.exists():
      LOGGER.info(
        f"Annotations directory {class_label_dir} already exists. "
        "Skipping extract."
      )
    else:
      annotations_class_file = annotations_dir / f"{class_id}.tar.gz"
      if annotations_class_file.exists():
        with tarfile.open(annotations_class_file, "r:gz") as tf:
          tf.extractall(annotations_dir)
        shutil.move(annotations_dir / "Annotation" / class_id, class_label_dir)
        LOGGER.info(f"Extracted annotations for {class_id} to {class_label_dir}")
        annoted_classes.append(class_id)
      else:
        LOGGER.info(f"There are no annotations for class {class_id}.")

  # Delete annotations directory
  LOGGER.info("Deleting annotations dir.")
  shutil.rmtree(annotations_dir)
  return annoted_classes


def download_imagenet_detections(class_ids: list[str], dir: Path):
  """
  Download ImageNet images and annotations for given class ids into dir

  Args:
    class_ids (list[str]): class_ids to download
    dir (Path): the directory to save images into
```

```
    """
    # Create dataset_dir
    dir.mkdir(exist_ok=True)
    data_dir = dir / "data"
    data_dir.mkdir(exist_ok=True)
    labels_dir = dir / "labels"
    labels_dir.mkdir(exist_ok=True)

    annoted_classes = download_annotations(class_ids, dir)

    # Download synset images for each class with annotations
    for class_id in annoted_classes:
        class_dir = data_dir / class_id
        if class_dir.exists():
            LOGGER.info(f"Directory {class_dir} already exists. Skipping download.")
        else:
            tarfilename = dir / f"{class_id}.tar"
            url = f"https://image-net.org/data/winter21_whole/{class_id}.tar"
            download_file(url, tarfilename)
            with tarfile.open(tarfilename) as tf:
                tf.extractall(class_dir)
            LOGGER.info(f"Extracted {class_dir}.")


def cleanup_labels_without_images(dir: Path):
    """
    Remove labels without images from dataset dir

    Args:
        dir (Path): the ImageNet dataset directory
    """
    data_dir = dir / "data"
    labels_dir = dir / "labels"
    classes = [path.name for path in data_dir.iterdir() if path.is_dir()]
    for class_id in classes:
        images = {
            path.stem for path in (data_dir / class_id).iterdir() if not path.is_dir()
        }
        labels = {
            path.stem for path in (labels_dir / class_id).iterdir() if not path.is_dir()
        }
        LOGGER.info(f"Deleting {len(labels.difference(images))} labels without images")
        for label_id in labels.difference(images):
            filename = labels_dir / class_id / (label_id + ".xml")
            filename.unlink()


def search(
    keywords: list[str],
    dir: Path = settings.ORION_HOME_DIR / "imagenet",
```

```
):
    """
    Search image net classes matching the given keywords.

    Args:
        keywords (list[str]): List of keywords to search for.
        dir (Path, optional): directory where files will be downloaded.
            Defaults to ORION_HOME_DIR / "imagenet".
    """
    classes = get_class_names(dir)
    filtered = {
        id: lemma
        for id, lemma in classes.items()
        if any([re.search(query, lemma, re.IGNORECASE) for query in keywords])
    }
    LOGGER.info(f'Displaying all ImageNet classes containing one of {keywords}')
    LOGGER.info("\n".join([f"{id}: \t{name}" for id, name in filtered.items()]))


def download(
    ids: list[str],
    dir: Path = settings.ORION_HOME_DIR / "imagenet",
):
    """
    Download ImageNet images and annotations for the given class ids.

    Args:
        ids (list[str]): the class ids to download.
        dir (Path, optional): the dataset directory.
            Defaults to settings.ORION_HOME_DIR / "imagenet".
    """
    download_imagenet_detections(ids, dir)
    cleanup_labels_without_images(dir)

import logging
from pathlib import Path
from typing import Annotated

import fiftyone as fo
import fiftyone.utils.random as four
import fiftyone.zoo as foz
import typer
from fiftyone.types.dataset_types import VOCDetectionDataset, YOLOv4Dataset

from orion.config.settings import settings
from orion.datasets.imagenet import download as download_imagenet
from orion.datasets.roboflow import LABEL_MAPPING
from orion.datasets.roboflow import download as download_roboflow
from orion.utils import download_and_extract
from orion.yolo.utils import export_yolo_data
```

```
app = typer.Typer()
LOGGER = logging.getLogger(__name__)

GOOGLE_DATASET_URL =
"https://github.com/jonasrenault/orion/releases/download/v1.2.0/military-vehicles-
dataset.tar.gz"


@app.command()
def prepare(
    dir: Annotated[
        Path,
        typer.Option(
            "--dir",
            "-d",
            help="Orion home directory.",
            file_okay=False,
            dir_okay=True,
        ),
    ] = settings.ORION_HOME_DIR,
    imagenet_ids: Annotated[
        list[str], typer.Option("--ids", help="List of class ids to download.")
    ] = ["n04389033"],
):
    """
    Prepare a dataset of annotated military vehicle images.

    Args:
        dir (Path, optional): directory where files will be downloaded.
            Defaults to ORION_HOME_DIR.
    """
    LOGGER.info("========== Downloading images from ImageNet dataset ==========")
    # Download ImageNet images for classes imagenet_ids
    imagenet_dir = dir / "imagenet"
    download_imagenet(imagenet_ids, imagenet_dir)

    # Create a dataset
    dataset = fo.Dataset.from_dir(
        dataset_dir=imagenet_dir, dataset_type=VOCDetectionDataset
    )
    dataset.map_labels("ground_truth", {"n04389033": "AFV"}).save()
    LOGGER.info(
        f"========== Dataset currently contains {dataset.count()} samples =========="
    )

    LOGGER.info("========== Downloading images from OpenImage dataset ==========")
    # Add OpenImages dataset
    oi_samples = foz.load_zoo_dataset(
        "open-images-v7", classes=["Tank"], only_matching=True, label_types="detections"
```

```
    ).map_labels("ground_truth", {"Tank": "AFV"})
    dataset.merge_samples(oi_samples)
    LOGGER.info(
        f"========== Dataset currently contains {dataset.count()} samples =========="
    )

    LOGGER.info("========== Downloading images from Roboflow dataset ==========")
    # Add roboflow dataset
    roboflow_dir = dir / "roboflow"
    download_roboflow(roboflow_dir)
    dataset_rf = fo.Dataset.from_dir(
        dataset_dir=roboflow_dir, dataset_type=VOCDetectionDataset
    )
    dataset_rf.map_labels("ground_truth", LABEL_MAPPING).save()
    dataset.merge_samples(dataset_rf)
    LOGGER.info(
        f"========== Dataset currently contains {dataset.count()} samples =========="
    )

    LOGGER.info("========== Downloading images from Google Images dataset ==========")
    # Add google images dataset
    google_dir = dir / "google"
    download_and_extract(
        GOOGLE_DATASET_URL, "military-vehicles-dataset.tar.gz", google_dir
    )
    dataset_google = fo.Dataset.from_dir(
        dataset_dir=google_dir / "dataset", dataset_type=YOLOv4Dataset
    )
    dataset.merge_samples(dataset_google)
    LOGGER.info(
        f"========== Dataset currently contains {dataset.count()} samples =========="
    )

    export_dir = dir / "dataset"
    LOGGER.info(f"========== Exporting dataset to {export_dir} ==========")
    # delete existing tags
    dataset.untag_samples(dataset.distinct("tags"))

    # split into train, test and val
    four.random_split(dataset, {"train": 0.8, "val": 0.1, "test": 0.1})
    export_yolo_data(
        dataset,
        export_dir,
        ["AFV", "APC", "MEV", "LAV"],
        split=["train", "val", "test"],
        overwrite=True,
    )

import logging
import shutil
```

```python
from pathlib import Path

from orion.config.settings import settings
from orion.utils import download_and_extract

LOGGER = logging.getLogger(__name__)

DATASET_URL = "https://github.com/jonasrenault/orion/releases/download/v1.3.0/Russian-military-annotated.v4-2022-06-13-with-null-images.voc.zip"
LABEL_MAPPING = {
    "bm-21": "AFV",
    "t-80": "AFV",
    "t-64": "AFV",
    "t-72": "AFV",
    "bmp-1": "AFV",
    "bmp-2": "AFV",
    "bmd-2": "AFV",
    "btr-70": "APC",
    "btr-80": "APC",
    "mt-lb": "APC",
}


def download(dir: Path = settings.ORION_HOME_DIR / "roboflow"):
    """
    Downlad images and annotations from the russian military annotated dataset
    on roboflow and format them to be imported into a fo.Dataset.

    Args:
        dir (Path, optional): the dataset dir.
            Defaults to settings.ORION_HOME_DIR / "roboflow".
    """
    download_and_extract(DATASET_URL, "dataset_rf.zip", dir)
    restructure_dataset(dir)


def restructure_dataset(dir: Path):
    """
    Restructure dataset by copying jpg images to the 'data' directory
    and xml files to the 'labels' directory from the source directory.

    Args:
        dir (Path): The source directory containing the dataset.
    """
    LOGGER.info(f"Restructuring dataset directory {dir}")
    # Create target subdirectories if they don't exist
    (dir / "data").mkdir(parents=True, exist_ok=True)
    (dir / "labels").mkdir(parents=True, exist_ok=True)

    # Define the source subdirectories
```

```
    source_subdirs = ["test", "train", "valid"]

    # Copy jpg and xml files from source to target
    for source_subdir in source_subdirs:
        source_path = dir / source_subdir

        # Copy jpg files to 'data'
        for jpg_file in source_path.glob("*.jpg"):
            shutil.copy(jpg_file, dir / "data")

        # Copy xml files to 'labels'
        for xml_file in source_path.glob("*.xml"):
            shutil.copy(xml_file, dir / "labels")

        # Delete the original subdirectory
        shutil.rmtree(source_path)

    LOGGER.info("Dataset directory restructured successfully.")

import csv
import logging
from pathlib import Path

import cv2
import fiftyone as fo
import numpy as np

from orion.config.settings import settings
from orion.utils import download_and_extract
from orion.yolo.utils import _uncenter_boxes

LOGGER = logging.getLogger(__name__)

DATASET_URL = (
    "https://github.com/jonasrenault/orion/releases/download/v1.3.0/search_2.tar.gz"
)
LABEL_MAPPING = {
    "M60": "AFV",
    "M3": "AFV",
    "M1": "AFV",
    "T72": "AFV",
    "HVS": "LAV",
    "HVT": "LAV",
    "BMP": "APC",
    "BTR": "APC",
    "M113": "APC",
}


def download(dir: Path = settings.ORION_HOME_DIR / "search_2"):
```

```
    download_and_extract(DATASET_URL, "search_2.tar.gz", dir)


def load_search_2_dataset(
    dir: Path = settings.ORION_HOME_DIR / "search_2" / "search_2",
) -> fo.Dataset:
    """
    Load The Seach_2 dataset from disk into a fiftyone Dataset

    Args:
        dir (Path, optional): dataset directory. Defaults to
            settings.ORION_HOME_DIR / "search_2" / "search_2"

    Returns:
        fo.Dataset: the dataset
    """
    # Read metadata
    with open(dir / "meta.csv", "r") as csvfile:
        reader = csv.DictReader(csvfile, delimiter=";")
        metas = [row for row in reader]

    samples = []
    for meta in metas:
        imagefp = dir / f'images/IMG{meta["Image"].zfill(4)}.jpg'
        maskfp = dir / f'masks/mask{meta["Image"].zfill(2)}.jpg'
        mask = cv2.imread(str(maskfp), cv2.IMREAD_GRAYSCALE)
        height, width = mask.shape

        # Get normalized bounding box coordinates from X, Y, W, H
        bbox = np.array(
            [
                [
                    int(meta["X"]) / width,
                    int(meta["Y"]) / height,
                    int(meta["W"]) / width,
                    int(meta["H"]) / height,
                ]
            ]
        )
        # X and Y are the target's center, convert them to top left coordinates
        _uncenter_boxes(bbox)

        # Only keep mask values inside the bounding box
        x, y, w, h = bbox[0]
        x *= width
        y *= height
        w *= width
        h *= height
        mask = mask[int(y) : int(y + h), int(x) : int(x + w)]
```

```
    sample = fo.Sample(filepath=imagefp)
    sample["ground_truth"] = fo.Detections(
        detections=[
            fo.Detection(
                label=meta["Target"],
                bounding_box=bbox[0].tolist(),
                mask=mask,
            )
        ]
    )
    sample["distance"] = meta["Distance"]
    samples.append(sample)

dataset = fo.Dataset()
dataset.add_samples(samples)
return dataset
```