ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Android-додаток для керування дроном і аналізу відеопотоку дронів DJI з детекцією мін на базі YOLO»

на здобуття освітнього ступеня бакалавра зі спеціальності
121 Інженерія програмного забезпечення
освітньо-професійної програми «Технологій цифрового розвитку»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Микола СТАРЦЕВ

(підпис)

Виконав: здобувач вищої освіти групи ТЦР-42
Микола СТАРЦЕВ

Керівник: Вадим ЧИТУЛЯН

старший викладач кафедри

Рецензент:

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Технологій цифрового розвитку

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

_____ Вікторія ЖЕБКА

« ____ » _____ 2026 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Старцеву Миколі Сергійовичу

1. Тема кваліфікаційної роботи: «Android-додаток для керування дроном і аналізу відеопотоку дронів DJI з детекцією мін на базі YOLO» керівник кваліфікаційної роботи старший викладач кафедри, Вадим ЧИТУЛЯН.
2. Строк подання кваліфікаційної роботи «25» травня 2026 р.
3. Вихідні дані до кваліфікаційної роботи: офіційна документація DJI Mobile SDK v4, TensorFlow Lite, Ultralytics YOLOv8, Android Jetpack, наукові джерела з детекції об’єктів та дистанційного зондування, аерофотознімки з БПЛА DJI Mavic 2.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
    1. Аналіз предметної галузі та існуючих підходів до автоматизованої детекції мін за допомогою БПЛА.
    2. Проектування архітектури мобільного застосунку та ML-компонента.

3. Програмна реалізація системи виявлення вибухонебезпечних предметів.
4. Тестування та аналіз результатів роботи системи.

5. Перелік графічного матеріалу: *презентація*
    1. Мета, об'єкт та предмет дослідження.
    2. Задачі кваліфікаційної роботи.
    3. Аналіз аналогів.
    4. Вимоги до android-застосунку.
    5. Програмні засоби реалізації.
    6. ML-конвеєр детекції.
    7. Результати навчання моделі.
    8. Архітектура застосунку.
    9. Діаграма варіантів використання (оператор бпла).
    10. Тестування продуктивності.
    11. Екранні форми
    12. Демонстрація роботи застосунку
    13. Апробація результатів дослідження
    14. Висновки
6. Дата видачі завдання «20» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	01.10-15.10.2025	
2	Аналіз існуючих аналогів та методів детекції об’єктів	16.10-31.10.2025	
3	Огляд архітектур нейронних мереж YOLOv8, технологій мобільної розробки та DJI SDK	01.11-20.11.2025	

4	Проектування архітектури Android-застосунку та ML-компонента	21.11-15.12.2025	
5	Програмна реалізація застосунку MineDetector	16.12-28.02.2026	
6	Тестування системи на реальних аерофотознімках	01.03-31.03.2026	
7	Оформлення роботи: вступ, висновки, реферат	01.04-15.04.2026	
8	Розробка демонстраційних матеріалів	16.04-30.04.2026	
9	Попередній захист роботи	18.05-31.05.2026	

Здобувач вищої освіти _____ Микола СТАРЦЕВ

(підпис)

Керівник

кваліфікаційної роботи _____ Вадим ЧИТУЛЯН

(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 51 стор., 18 табл., 24 рис., 21 джерело.

Мета роботи - розробка мобільного Android-застосунку для автоматизованого виявлення протипіхотних та протитанкових мін на місцевості за допомогою безпілотного літального апарату DJI Mavic 2 та нейронної мережі YOLOv8, розгорнутої локально на мобільному пристрої у форматі TFLite.

Об'єкт дослідження - процес автоматизованого виявлення вибухонебезпечних предметів на відеопотоці з БПЛА.

Предмет дослідження - методи та програмні засоби розробки мобільного застосунку для детекції мін у реальному часі з використанням згорткових нейронних мереж сімейства YOLO та інтеграції з БПЛА серії DJI Mavic 2.

Короткий зміст роботи: У роботі проаналізовано проблему мінної загрози в Україні та існуючі рішення для дистанційного виявлення мін. Обґрунтовано вибір архітектури YOLOv8n та технологій мобільної розробки. Зібрано та анотовано датасет аерофотознімків (129 зображень, 6 класів мін). Навчено модель YOLOv8n з використанням трансферного навчання та конвертовано у формат TFLite. Розроблено Android-застосунок мовою Kotlin з архітектурою MVVM, інтеграцією DJI Mobile SDK v4, системою геолокації на карті Mapbox та обробкою відеопотоку в реальному часі. Проведено тестування на полігоні з макетами мін та оцінено продуктивність на мобільних пристроях.

Сферою використання є автоматизація процесу попереднього обстеження забрудненої території для саперних підрозділів, Державної служби з надзвичайних ситуацій та гуманітарних організацій з розмінування.

КЛЮЧОВІ СЛОВА: МОБІЛЬНИЙ ЗАСТОСУНОК, ВИЯВЛЕННЯ МІН, YOLOV8, TENSORFLOW LITE, БПЛА, DJI, КОМП’ЮТЕРНИЙ ЗІР, ANDROID, KOTLIN.

ЗМІСТ

ВСТУП

Актуальність: за даними Організації Об'єднаних Націй, станом на 2024 рік в Україні забруднено понад 156 000 квадратних кілометрів території вибухонебезпечними предметами (ВНП), що становить приблизно 26% від загальної площі держави та робить Україну однією з найбільш замінованих країн світу [1]. Ця площа перевищує територію таких європейських країн як Англія або Греція. За оцінками міжнародних організацій, повне очищення забрудненої території може зайняти від 30 до 757 років при використанні виключно традиційних ручних методів розмінування [2].

Ручне розмінування є надзвичайно небезпечним, повільним та дорогим процесом. Один сапер обстежує в середньому 20-50 квадратних метрів на день, при цьому кожна помилка може коштувати життя. За даними Державної служби України з надзвичайних ситуацій (ДСНС), протягом 2022-2024 років було знешкоджено понад 500 000 вибухонебезпечних предметів, проте обсяг забруднення залишається критичним [3]. Вартість розмінування одного квадратного метра складає від 1 до 5 доларів США, що при масштабах забруднення вимагає мільярдних інвестицій.

Аерофотозйомка з висоти 5-15 метрів дозволяє оглядати значні площі за короткий час, а алгоритми глибокого навчання здатні автоматично ідентифікувати потенційно небезпечні об'єкти на зображеннях. Така комбінація збільшує швидкість обстеження місцевості у 50-200 разів порівняно з піхотним обходом та практично усуває ризик для саперів на етапі попередньої розвідки.

Нейронні мережі YOLO виконують детекцію за один прохід через мережу, що робить їх достатньо швидкими для роботи в реальному часі [4]. Архітектура YOLOv8 від компанії Ultralytics пропонує оптимальний баланс між швидкістю та точністю детекції, особливо у варіанті nano (YOLOv8n), оптимізованому для мобільних пристроїв [5]. Формат TFLite дає можливість запускати модель безпосередньо на Android-пристрої без хмарних сервісів [6], що є критично важливим для польових умов.

Об'єкт дослідження - процес автоматизованого виявлення вибухонебезпечних предметів на аерофотознімках, отриманих з безпілотних літальних апаратів.

Предмет дослідження - методи та програмні засоби розробки мобільного застосунку для детекції мін у реальному часі з використанням згорткових нейронних мереж сімейства YOLO та інтеграції з БПЛА серії DJI Mavic 2.

Мета роботи - розробка мобільного Android-застосунку для автоматизованого виявлення протипіхотних та протитанкових мін на місцевості за допомогою безпілотного літального апарату DJI Mavic 2 та нейронної мережі YOLOv8, розгорнутої локально на мобільному пристрої у форматі TFLite, для допомоги саперам у попередній розвідці місцевості та подальшому розмінуванні територій.

Актуальність теми додатково підтверджується статистикою Державної служби з надзвичайних ситуацій (ДСНС): щомісяця в Україні фіксується до 200 випадків виявлення вибухонебезпечних предметів цивільним населенням. Традиційні методи розмінування потребують від 2 до 5 годин на гектар, тоді як застосування БПЛА з автоматизованою детекцією дозволяє скоротити цей час до 15-20 хвилин первинного обстеження.

Існуючі комерційні рішення, такі як SpotlightAI ONSITE та Dropla BLUE EYES, орієнтовані на військові замовлення та мають закритий вихідний код, що унеможливлює їх адаптацію для волонтерських організацій та цивільних сапернів. Розробка відкритого Android-застосунку з підтримкою доступних дронів серії DJI Mavic створює передумови для масового впровадження технології автоматизованої детекції мін.

Для досягнення поставленої мети в бакалаврській роботі виконано наступні задачі:

1. Провести аналіз предметної області, дослідити проблему мінної загрози в Україні та існуючі рішення для дистанційного виявлення мін.
2. Обґрунтувати вибір архітектури нейронної мережі (YOLOv8n) та технологій розробки мобільного застосунку.

3. Зібрати та анотувати датасет аерофотознімків мін, зроблених з БПЛА DJI Mavic 2.
4. Навчити модель YOLOv8n на зібраному датасеті з використанням трансферного навчання та конвертувати її у формат TFLite.
5. Розробити Android-застосунок мовою Kotlin з архітектурою MVVM та інтеграцією DJI Mobile SDK v4.
6. Реалізувати повний конвеєр обробки відеопотоку у реальному часі: декодування H.264/H.265, ML-детекція, трекінг, візуалізація.
7. Реалізувати систему геолокації з відображенням на інтерактивній карті Mapbox з офлайн-підтримкою.
8. Провести тестування системи на полігоні з макетами мін та оцінити продуктивність на мобільних пристроях.

Методи дослідження: методи глибокого навчання для детекції об'єктів (YOLOv8 з anchor-free архітектурою), трансферне навчання з попередньо навченої моделі (COCO pretrained), аугментацію даних для компенсації малого датасету, Non-Maximum Suppression (NMS) для фільтрації дублікатів детекцій, алгоритм ByteTrack для багатооб'єктного трекінгу між кадрами відео [7], а також методи оптимізації ML-інференсу для мобільних платформ (квантизація float16, GPU-делегат TFLite, pre-allocated буфери).

Наукова новизна - система працює повністю офлайн: ML-модель запускається прямо на телефоні, дрон керується через DJI SDK без інтернету, що відрізняє її від хмарних аналогів. Розроблено спеціалізовану багаторівневу систему фільтрації хибних спрацювань для аерофотозйомки: фільтр темних зон, покласні пороги впевненості, геометричні фільтри (площа, пропорції, відступ від краю) та детектор артефактних сіткових патернів.

Практичне значення. Застосунок MineDetector може бути використаний саперними підрозділами ЗСУ, ДСНС та гуманітарними організаціями для попереднього обстеження місцевості. Результати детекції відображаються в реальному часі, що дозволяє оператору оперативно фіксувати підозрілі ділянки та передавати інформацію саперам.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз проблеми мінної загрози в Україні

Після повномасштабного вторгнення у 2022 році мінна загроза в Україні різко зросла - значна частина території виявилась забруднена мінами, касетними боєприпасами та іншими вибухонебезпечними предметами (ВНП). За даними Landmine Monitor 2024, Україна посідає перше місце у світі за площею мінного забруднення (близько 156 000 км², або 26% території держави) [2].

За даними ДСНС, протягом 2022-2024 років обстежено понад 35 000 км² та знешкоджено понад 500 000 ВНП [3], проте це лише невелика частка від загального обсягу забруднення. Мінна загроза безпосередньо впливає на безпеку цивільного населення, сільське господарство, відновлення інфраструктури та повернення переміщених осіб додому.

На рисунку 1.1 наведено карту мінного забруднення території України станом на 2024 рік, що ілюструє масштаб проблеми.

Рис. 1.1 Карта мінного забруднення території України станом на 2024 рік [2]

Характеристики обраних типів мін для детекції наведено у таблиці 1.1.

Таблиця 1.1

Характеристики типів мін, обраних для детекції

ID	Клас	Тип	Розміри
0	MON-50	Осколкова направлена	226x156x66 мм
1	PFM-1	Протипіхотна фугасна	120x62x18 мм
2	PMN-1	Протипіхотна натискна	112 мм діам.
3	PMN-2	Протипіхотна натискна	120 мм діам.
4	POM-3	Протипіхотна стрибаюча	60 мм діам.
5	TM-62	Протитанкова натискна	320 мм діам., 9 кг

1.2 Аналіз існуючих аналогів для автоматизованого виявлення мін

PFM-1 («пелюстка») є особливо небезпечною через малий розмір (62 мм), пластиковий корпус, що не виявляється металошукачами, та масове застосування - до 432 штук за один авіаційний скид. POM-3 «Медальйон» оснащена сейсмічним датчиком реакції на крок людини до 16 метрів та елементом самоліквідації.

Ручне розмінування щупом і металошукачем забезпечує лише 20-50 м² на день при постійному ризику для сапера. Механізоване підходить лише для відкритих рівнин і не гарантує 100% очищення. Кінологічне обмежене кліматом, витривалістю тварин (до 2 годин) та тривалою підготовкою.

БПЛА дозволяють збільшити продуктивність обстеження у 50-200 разів. DJI Mavic 2 з часом польоту 31 хвилина охоплює 10-20 гектарів за виліт, тоді як група з 3 саперів - лише 0.05-0.15 гектара за той самий час, при цьому ризик для людей практично відсутній.

1.2.1 Огляд існуючих рішень для дистанційного виявлення мін

Сучасні підходи до дистанційного виявлення мін з використанням безпілотних платформ можна класифікувати за типом сенсора та методом обробки даних.

Мультиспектральна зйомка (NIR, LWIR) дозволяє виявляти порушення ґрунту та аномалії рослинності над закопаними об'єктами; теплові камери фіксують різницю температур між міною та ґрунтом у 0.5-2°C. Доступні рішення - DJI Phantom 4 Multispectral та Mavic 3 Multispectral (від 3 500 USD) - орієнтовані на сільське господарство і потребують додаткової калібрації. Основний недолік - залежність від кліматичних умов та часу доби.

Георадар (GPR) виявляє підповерхневі об'єкти на глибині до 30 см, проте потребує висоти польоту 1-2 м, що створює ризик підриву від штирьових підривачів. Маса модуля 2-5 кг обмежує сумісні БПЛА мультикоптерами від 10 кг.

Комп'ютерний зір (Visual Detection). Використання RGB-камер у поєднанні з алгоритмами глибокого навчання є найбільш доступним підходом - достатньо споживчого дрона з камерою (від 1 500 USD). Обмеження: працює лише з мінами на поверхні або частково замаскованими.

Порівняльний аналіз існуючих програмних рішень для детекції мін наведено у таблиці 1.2.

Таблиця 1.2

Рішення	Сенсор	Обробка	Автономність	Вартість
SpotlightAI ONSITE	RGB	Локальна (Windows ноутбук)	Так (офлайн)	Комерційна
Dropla BLUE EYES	RGB + thermal	Edge-пристрій (2.1 кг)	Так (офлайн)	Комерційна
Blackthorn.ai	RGB	Хмарна (CNN)	Ні (потрібен інтернет)	Комерційна
SENSYS MAGNETO	Магнітометр	Локальна (Windows)	Так (офлайн)	Комерційна
MineDetector (наш)	RGB (DJI)	Локальна TFLite	Так	~1 500 USD

Вибір шести типів мін обумовлений звітами HALO Trust та Danish Demining Group за 2023-2024 роки: понад 85% виявлених в Україні мін належать саме до цих

типів. PFM-1 та PMN-1 становлять найбільшу загрозу через масове дистанційне розсіювання.

Особливу складність становлять міни малого розміру (PFM-1: 120×62×18 мм) та міни з корпусом, що зливається з ґрунтом (PMN-2: темно-зелений, 120 мм). Це вимагає від моделі високої чутливості до дрібних деталей на різноманітних текстурах поверхні.

На рисунку 1.2 наведено порівняння підходів до дистанційного виявлення мін за ключовими параметрами.

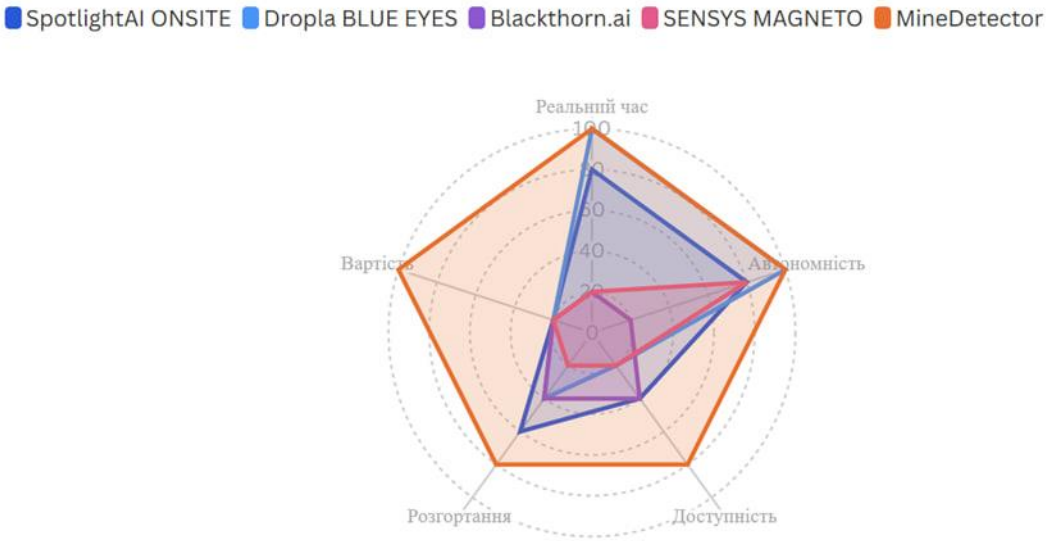


Рис. 1.2 Порівняння підходів до дистанційного виявлення мін

Існуючі рішення мають суттєві обмеження: SpotlightAI та Dropla потребують дорогого обладнання, Blackthorn.ai залежить від хмари, SENSYS MAGNETO орієнтований лише на магнітометричний пошук. MineDetector пропонує доступне рішення на базі споживчого дрона DJI Mavic 2 з локальною обробкою без мережевого з'єднання.

1.3 Аналіз методів комп'ютерного зору для детекції об'єктів

Детекція об'єктів - задача пошуку всіх об'єктів заданих класів на зображенні з обведенням їх рамками. Формально, для зображення I розміром W×H детектор

повертає D = {(b_i, c_i, p_i)}, де b_i = (x, y, w, h) - координати рамки, c_i - клас, p_i - впевненість [4].

Сучасні підходи до детекції поділяються на двостадійні та одностадійні детектори.

Двостадійні детектори (Faster R-CNN, Cascade R-CNN [8]) спочатку знаходять потенційні ділянки через RPN, потім уточнюють координати. Висока точність (mAP@50 80%+ на COCO [9]), але 5-15 FPS на серверному GPU унеможливлює мобільне застосування. Одностадійні детектори (SSD [10]) виконують детекцію за один прохід, що значно підвищує швидкість.

Серія YOLO постійно розвивалась: YOLOv4 запропонував оптимізації для підвищення точності без втрати швидкості [16], а YOLOv7 встановив нові показники для детекції в реальному часі [18]. Повну еволюцію серії наведено у таблиці 1.3.

Таблиця 1.3

Еволюція архітектур серії YOLO

Версія	Рік	Backbone	Ключові нововведення	mAP50 (COCO)
YOLOv1	2016	Darknet	Перший one-stage детектор	63.4%
YOLOv3	2018	Darknet-53	Multi-scale, FPN neck	57.9%
YOLOv4	2020	CSPDarknet53	Bag of Freebies, Mish activation	65.7%
YOLOv5	2020	CSPDarknet	PyTorch, auto-augment, export pipeline	68.9%
YOLOv7	2022	E-ELAN	Compound scaling, RepConv	73.1%
YOLOv8	2023	CSPDarknet (C2f)	Anchor-free, decoupled head, DFL	75.2%

YOLOv8 від Ultralytics обрано для даного проекту як найновішу архітектуру з оптимальним балансом швидкості та точності. Модель пропонує варіанти за розміром: nano (n, 3.2M параметрів), small (s, 11.2M), medium (m, 25.9M), large (l, 43.7M), extra-large (x, 68.2M). Для мобільного розгортання обрано YOLOv8n.

Архітектура YOLOv8 складається з трьох частин - backbone, neck та head, як зображено на рисунку 1.3. Backbone побудований на принципах залишкового навчання [15], що дозволяє ефективно тренувати глибокі мережі.

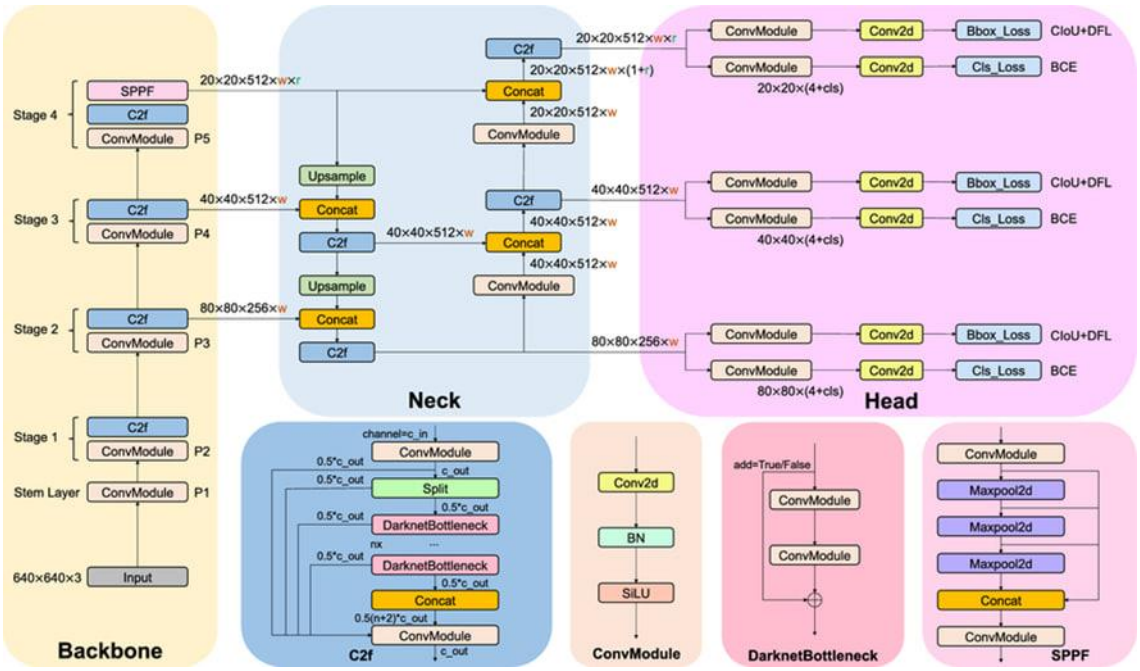


Рис. 1.3 Архітектура нейронної мережі YOLOv8n: backbone, neck та detection head

У YOLOv8 реалізовано anchor-free підхід: на відміну від попередніх версій, де потрібно було вручну підбирати розміри anchor boxes під датасет, мережа тепер самостійно прогнозує центр і розміри рамки, що особливо важливо для об'єктів різних форм і пропорцій.

Голова мережі (head) теж зазнала змін - тепер класифікація та регресія координат виконуються окремими підмережами. Раніше обидві задачі вирішувались разом, через що градієнти конфліктували між собою. Розділення дозволило підвищити точність локалізації [5].

Замість блоків C3 з YOLOv5 розробники впровадили C2f (Cross-Stage Partial with 2 convolutions and Flow). Завдяки цьому модель навчається ефективніше - приріст приблизно 15-20% порівняно з попереднім поколінням [5].

Окремої уваги заслуговує функція втрат Distribution Focal Loss. Вона прогнозує не одне значення координати, а розподіл ймовірностей навколо неї. Для об'єктів нестандартної форми, як-от MON-50 з прямокутним корпусом, це дає помітно точнішу локалізацію.

1.4 Обґрунтування вибору технологій розробки

Технології для MineDetector підбирались під конкретні обмеження: додаток має працювати без інтернету, підключатись саме до DJI Mavic 2, обробляти відео нейромережею прямо на телефоні.

TensorFlow Lite (TFLite) [6] - офіційне рішення Google для мобільного ML. GPU-прискорення через OpenCL/OpenGL ES 3.1 дає приріст у 3-5 разів порівняно з CPU. Формат float16 скорочує розмір моделі з 12 до 6 МБ при втраті точності менше 0.5% mAP.

БПЛА SDK: DJI Mobile SDK v4 - єдиний офіційний SDK для серії Mavic 2 (Pro, Zoom, Enterprise) [12]. Забезпечує доступ до камери (live video, photo/video capture), підвісу (gimbal rotation), телеметрії (GPS, altitude, speed, battery), навігації (waypoints, virtual sticks) та медіа (SD card access). DJI Mobile SDK v5 не підтримує Mavic 2.

Архітектура застосунку побудована за патерном MVVM (Model-View-ViewModel) з Repository шаром, що є рекомендованим підходом Google для Android-розробки [13]. ViewModel зберігає стан екрану навіть при повороті пристрою, а StateFlow та LiveData забезпечують автоматичне оновлення інтерфейсу при зміні даних. Repository шар відповідає за роботу з даними та приховує від ViewModel деталі того, звідки ці дані приходять - з локальної бази чи з мережі.

Для локального збереження даних використовується бібліотека Android Room, яка є обгорткою над SQLite від Google [13]. Room перевіряє всі SQL-запити на етапі компіляції, що дозволяє виявити помилки ще до запуску застосунку. Робота з базою відбувається через Kotlin корутини [11], завдяки чому операції читання та запису не блокують інтерфейс. У базі зберігаються результати детекцій та метадані знімків з камери дрона.

Для відображення карти обрано Mapbox Android SDK, головною перевагою якого є повноцінна офлайн-підтримка [14]. Оператор може заздалегідь завантажити карту потрібного регіону та працювати без інтернету, що є критичним для польових умов. Mapbox також підтримує супутниковий режим відображення та

кластеризацію маркерів. Google Maps було відхилено саме через відсутність повноцінного офлайн-режиму.

Повний стек технологій наведено у таблиці 1.4.

Таблиця 1.4

Компонент	Технологія	Версія	Обґрунтування вибору
Мова	Kotlin	1.9+	Null-safety, корутини, офіц. мова Android
ML-інференс	TFLite	2.14+	GPU-делегат, ~6 МБ модель
Навчання	Ultralytics YOLOv8	8.0+	SOTA anchor-free, Python API
БПЛА SDK	DJI Mobile SDK v4	4.16+	Єдиний SDK для Mavic 2
Архітектура	MVVM + Repository	-	Стандарт Google, тестованість
БД	Room (SQLite)	2.6+	Compile-time SQL, coroutines
Карта	Mapbox Android	10.0+	Офлайн, кастомізація стилів
Мережа	Retrofit + OkHttp	2.9+	REST API, інтерцептори
Зображення	Coil	2.5+	Coroutines-native, кешування
Відео	ExoPlayer (Media3)	1.2+	Стабільний плеєр від Google
Збірка	Gradle + KSP	8.2+	Kotlin Symbol Processing

1.5 Постановка задачі

На основі аналізу предметної області та існуючих рішень сформульовано вимоги до системи.

Функціональні вимоги:

— детекція 6 типів мін (MON-50, PFM-1, PMN-1, PMN-2, POM-3, TM-62) у реальному часі на відеопотоці з камери БПЛА;

— підключення до DJI Mavic 2 (Pro, Zoom, Enterprise) через DJI Mobile SDK v4 з підтримкою USB;

— відображення live відеопотоку у режимі FPV з обмежувальними рамками, назвою класу та впевненістю;

— телеметрія в реальному часі: висота, швидкість, батарея, GPS, супутники, відстань від Home;

— керування підвісом камери (pitch -90°..+30°);

— тестовий режим детекції на фото з галереї без дрона;

— перегляд та завантаження знімків з камери дрона;

— налаштування параметрів камери (ISO, витримка, баланс білого).

Нефункціональні вимоги:

— повна автономність - робота без інтернет-з'єднання, всі обчислення локально на пристрої;

— швидкість ML-інференсу не менше 10 FPS на пристроях з GPU-сумісним SoC;

— підтримка Android 7.0+ (API 24), цільова версія Android 14 (API 34);

— розмір APK не більше 100 МБ з урахуванням DJI SDK нативних бібліотек та ML-моделі;

— стабільна робота при тривалих польотах понад 30 хвилин без крашів, витоків пам'яті та ANR;

— graceful degradation: автоматичний перехід GPU → CPU при відсутності GPU-підтримки;

— crash logging у файл для діагностики на пристроях без ADB (DJI Smart Controller).

2 ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Архітектура мобільного застосунку

Застосунок MineDetector побудовано на патерні MVVM з Repository-шаром та реактивним UI через StateFlow. Система має чотири логічні шари (рис. 2.1).

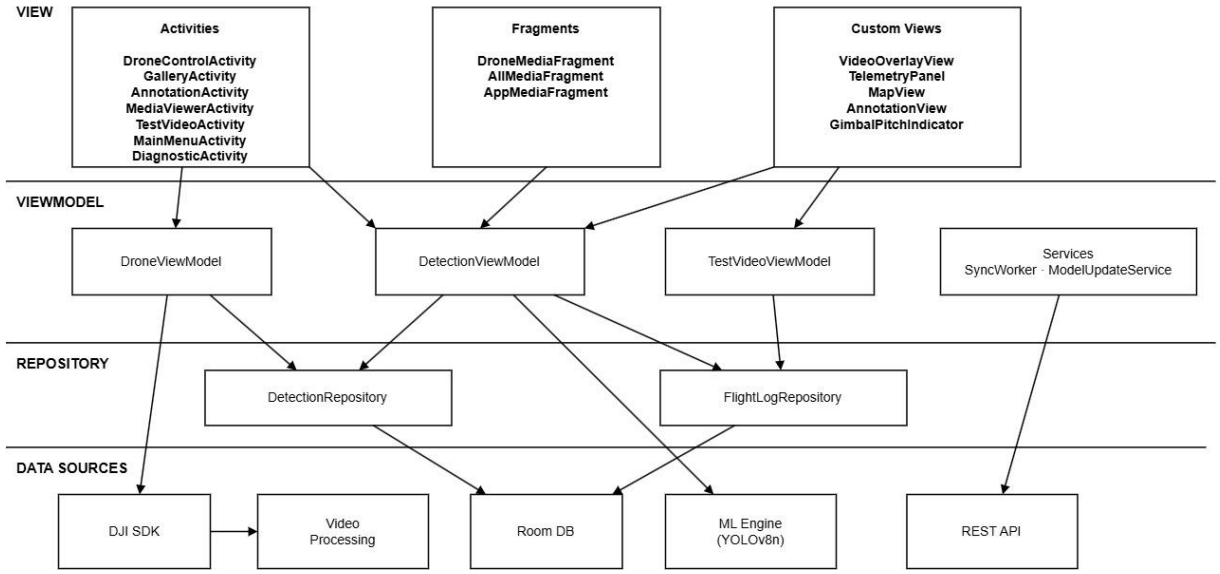


Рис. 2.1 Архітектура застосунку MineDetector (MVVM + Repository)

UI Layer (10 Activity, 3 Fragment, 5 View), ViewModel Layer (3 ViewModel), Repository Layer (Room DB) та Data/Hardware Layer (TFLite, DJI SDK, API) утворюють 62 файли Kotlin у 13 пакетах (рис. 2.2).

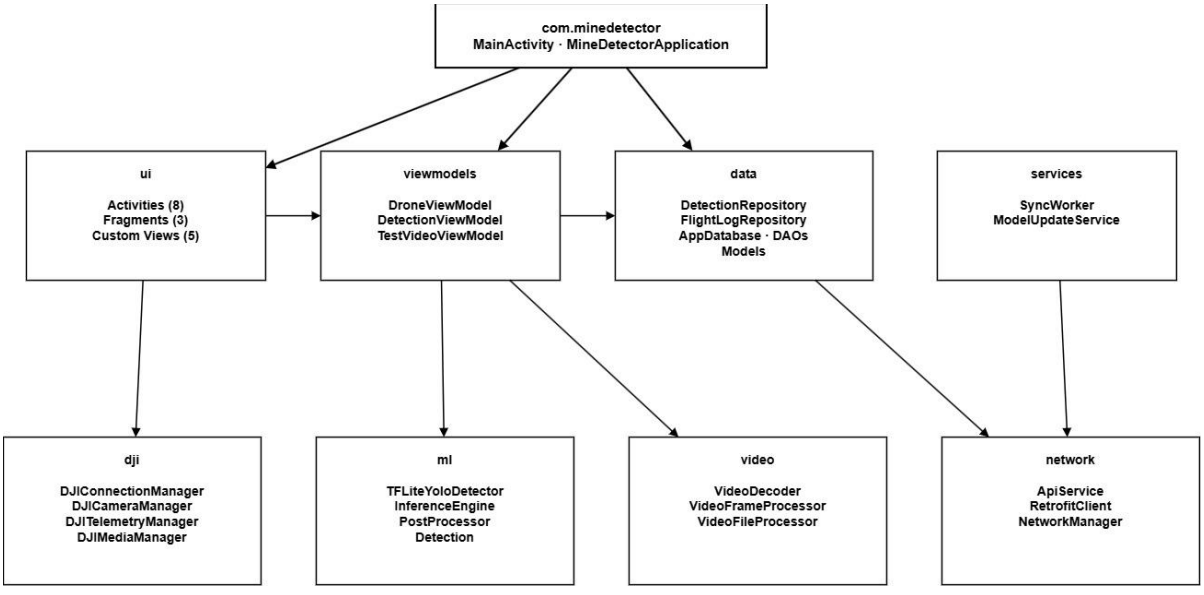


Рис. 2.2 Діаграма структури пакетів проекту MineDetector

Кожен пакет відповідає за окрему функціональність (табл. 2.1).

Таблиця 2.1

Структура пакетів та модулів проекту

Пакет	Файлів	Призначення	Ключові класи
com.minedetector.ui	8	Екрани застосунку	DroneControlActivity, GalleryActivity
com.minedetector.ui.components	6	Кастомні View	VideoOverlayView, TelemetryPanel
com.minedetector.ui.fragments	4	Фрагменти галереї	DroneMediaFragment, MediaAdapter
com.minedetector.viewmodels	3	ViewModels	DroneViewModel, DetectionViewModel
com.minedetector.ml	6	ML-конвеєр	InferenceEngine, TFLiteYoloDetector
com.minedetector.dji	6	Робота з DJI SDK	DJIConnectionManager, DJICameraManager
com.minedetector.data.local	7	Локальна БД	AppDatabase, DetectionDao
com.minedetector.data.models	5	Моделі даних	Telemetry, DroneState, MediaItem
com.minedetector.data.repository	2	Репозиторії	DetectionRepository, FlightLogRepository
com.minedetector.network	3	Мережевий шар	ApiService, RetrofitClient
com.minedetector.services	2	Фонові сервіси	SyncWorker, ModelUpdateService
com.minedetector.utils	4	Утиліти	Constants, PermissionHelper

У MVVM View показує UI, ViewModel тримає стан, Repository надає дані. Менеджери DJI SDK реалізовано як singleton через Application.

2.2 Проектування бази даних

Локальна база даних "minedetector.db" реалізована через Android Room та складається з трьох таблиць. ER-діаграму бази даних наведено на рисунку 2.3.

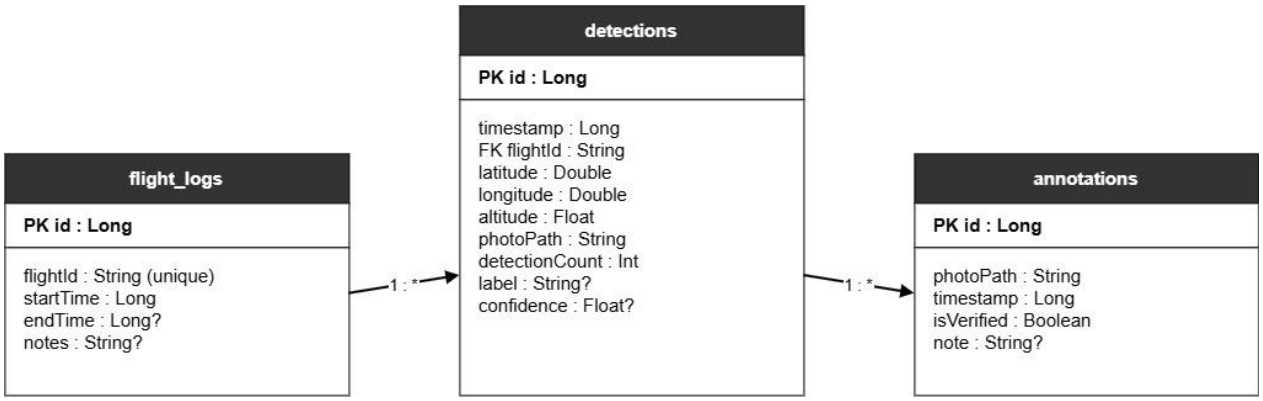


Рис. 2.3 ER-діаграма бази даних MineDetector (3 таблиці)

База даних містить три таблиці: detections (мітка часу, знімок, клас міни, впевненість), flight_logs (час початку та завершення польоту) та annotations (ручна перевірка результатів), пов'язані через flightId та photoPath. Структуру наведено у таблицях 2.2-2.4.

Таблиця 2.2

Структура таблиці detections

Поле	Тип	Індекс	Опис
id	Long (PK, auto)	-	Унікальний ідентифікатор
timestamp	Long	Так	Мітка часу (Unix ms)
flightId	String	Так	UUID польоту для групування
latitude	Double	-	GPS широта (WGS84)
longitude	Double	-	GPS довгота (WGS84)
altitude	Float	-	Висота над точкою зльоту (м)
photoPath	String	Так	Шлях до збереженого кадру
detectionCount	Int	-	Кількість об'єктів на кадрі
label	String?	-	Клас міни

Таблиця 2.3 flight_logs зберігає інформацію про кожен виліт дрона, зокрема унікальний ідентифікатор, час початку та завершення польоту, а також примітки оператора.

Таблиця 2.3

Структура таблиці flight_logs

Поле	Тип	Індекс	Опис
id	Long (PK, auto)	-	Унікальний ідентифікатор
flightId	String	Unique	UUID польоту
startTime	Long	Так	Час початку
endTime	Long?	-	Час завершення (null = в процесі)
notes	String?	-	Примітки оператора

Таблиця 2.4 annotations використовується для ручної перевірки результатів детекції - оператор може позначити знімок як перевірений та залишити коментар.

Таблиця 2.4

Структура таблиці annotations

Поле	Тип	Індекс	Опис
id	Long (PK, auto)	-	Унікальний ідентифікатор
photoPath	String	Так	Шлях до фото
timestamp	Long	Так	Мітка часу
isVerified	Boolean	Так	Чи перевірено вручну
note	String?	-	Коментар анотатора

AppDatabase реалізує Singleton - база створюється один раз і використовується у всьому застосунку. Для спрощення розробки застосовано fallbackToDestructiveMigration() замість міграційних скриптів.

2.3 Проектування ML-компонента

ML-компонент реалізує повний конвеєр обробки зображень від Bitmap до списку детекцій. Блок-схему ML-конвеєра наведено на рисунку 2.4.

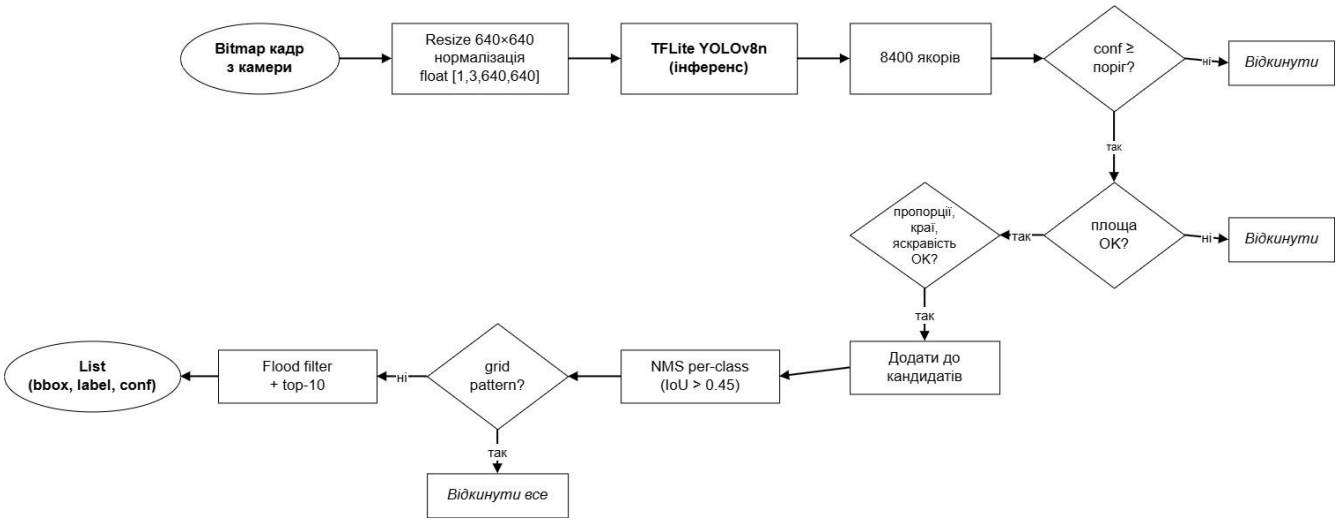


Рис. 2.4 Блок-схема ML-конвеєра: препроцесинг, інференс, постпроцесинг

TFLiteYoloDetector завантажує модель з assets через memory-mapped файл і використовує GPU-делегат з CPU-резервом на 4 потоки.

На вхід подається зображення 640×640 float16, на виході тензор [1, 10, 8400] з 4 координатами рамки та 6 ймовірностями класів для кожного з 8400 вікон. Три масштаби feature map (80×80, 40×40, 20×20) виявляють об'єкти різного розміру.

Для фільтрації хибних спрацювань реалізовано багаторівневу систему з параметрами, підібраними на основі 28 кадрів реального польоту (табл. 2.5).

Таблиця 2.5

Параметри багаторівневої системи фільтрації детекцій

Поріг глобальний	DEFAULT_CONFIDENCE	0.45	Мінімальна впевненість кандидата
Поріг MON-50	classMinThreshold[0]	0.72	Хибні на траві (до 68%)
Поріг PFM-1	classMinThreshold[1]	0.52	Хибні через малий розмір
Поріг PMN-1	classMinThreshold[2]	0.55	Хибні на круглих об'єктах
Поріг PMN-2	classMinThreshold[3]	0.50	Аналогічно PMN-1
Поріг POM-3	classMinThreshold[4]	0.50	Хибні на темних зонах
Поріг TM-62	classMinThreshold[5]	0.55	Великі bbox-хибняки на траві
Площа min	MIN_BOX_AREA_NORM	0.0005	Відкидає дрібні шуми

Продовження таблиці 2.5

Площа max	MAX_BOX_AREA_NORM	0.15	Відкидає > 15% кадру
Пропорції	MAX_ASPECT_RATIO	3.0	Відкидає витягнуті bbox
Краї	EDGE_MARGIN_FRAC	0.06	Відкидає обрізані на краях (6%)
Яскравість	MIN_BOX_BRIGHTNESS	25/255	Фільтр темних зон
NMS	IOU_THRESHOLD	0.45	Придушення дублікатів
Ліміт	MAX_DETECTIONS	10	Макс. детекцій на кадр

Покласові пороги впевненості - ключова частина фільтрації. MON-50 отримав найвищий поріг 0.72, оскільки модель хибно детектує траву як цей клас з впевненістю до 68%. Відфільтровані детекції проходять через NMS та grid-pattern suppression для відкидання сіткових артефактів.

2.4 Проектування інтеграції з DJI SDK

Інтеграція з БПЛА реалізована через систему менеджерів. Діаграму взаємодії DJI SDK менеджерів наведено на рисунку 2.5.

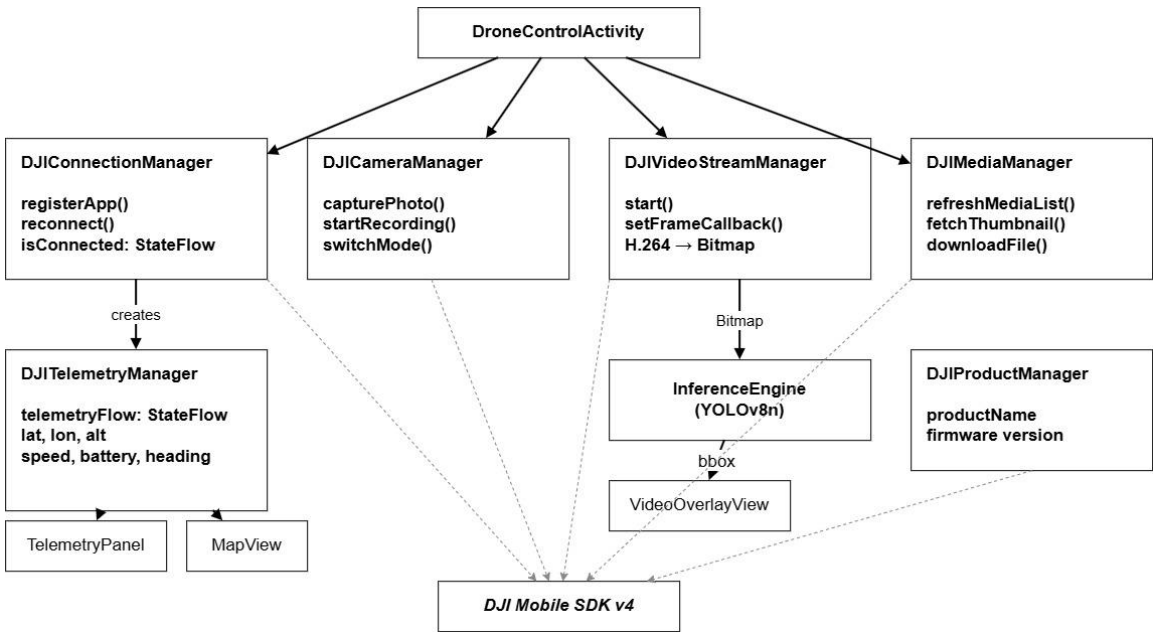


Рис. 2.5 Діаграма взаємодії DJI SDK менеджерів у застосунку MineDetector

Кожен менеджер відповідає за окремий аспект роботи з дроном і працює незалежно від інших. Їх опис наведено у таблиці 2.6.

Таблиця 2.6

Менеджери DJI SDK та їхні функції

Менеджер	Рядків	Підсистема	Ключові функції
DJIConnectionManager	191	Підключення	API key реєстрація, auto-reconnect, onComponentChange
DJIVideoStreamManager	110	Відео	H.264/H.265 → MediaCodec → TextureView → Bitmap
DJITelemetryManager	325	Телеметрія	GPS, altitude, speed, battery → StateFlow
DJICameraManager	77	Камера	Photo/video mode, capture, exposure
DJIMediaManager	332	Медіа	SD card listing, thumbnail/preview, file download
DJIProductManager	51	Продукт	Ідентифікація моделі, firmware version

При підключенні до дрона застосунок автоматично перевіряє доступність функцій камери, оскільки різні моделі серії Mavic 2 мають різний набір підтримуваних API.

2.5 Діаграма варіантів використання

Система передбачає одного актора - оператор БПЛА. Діаграму варіантів використання наведено на рисунку 2.6.

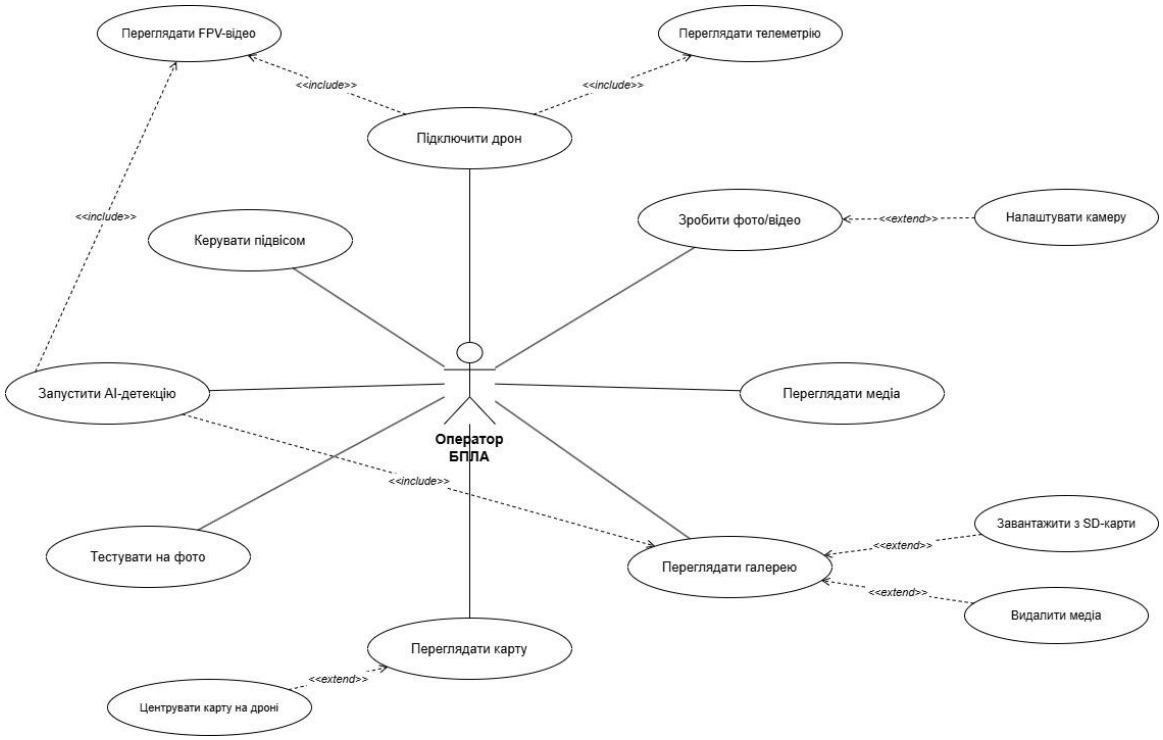


Рис. 2.6 Діаграма варіантів використання (Use Case Diagram) актора «Оператор БПЛА»

Оператор керує БПЛА (FPV, підвіс, телеметрія, фото/відео, медіа з SD), переглядає карту Mapbox з офлайн-режимом, використовує AI-детекцію з bounding boxes, тестує на фото з галереї та налаштовує камеру і діагностику.

2.6 Діаграма компонентів системи

Діаграму компонентів системи та їхніх залежностей наведено на рисунку 2.7.

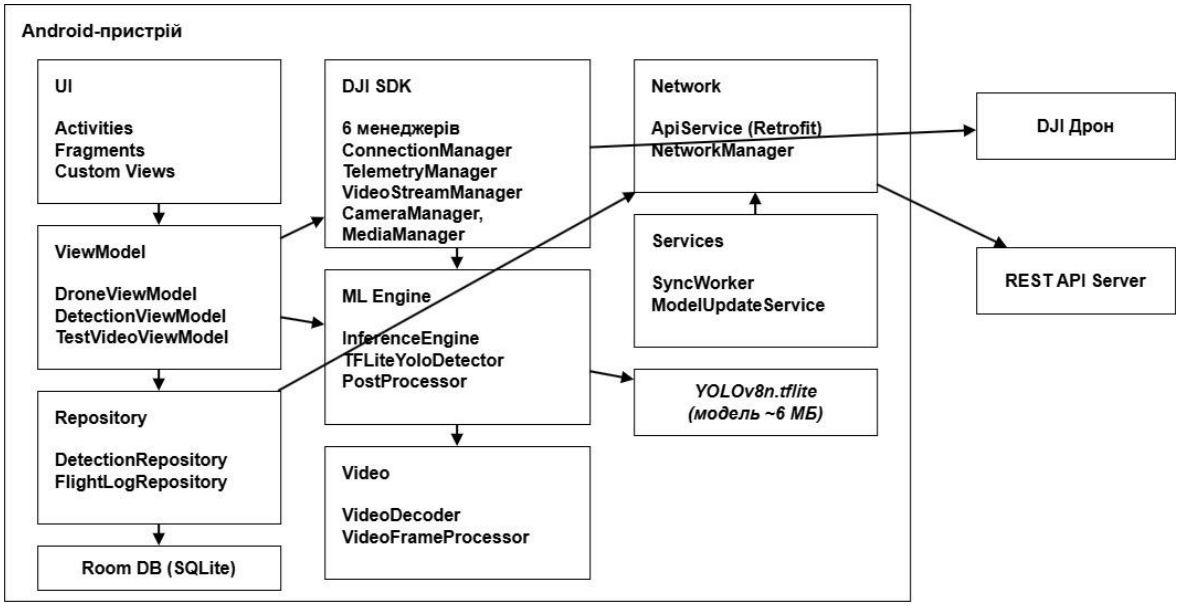


Рис. 2.7 Діаграма компонентів системи MineDetector

Результати детекції одразу відображаються поверх відеопотоку: bounding box з класом та впевненістю без перемикання між екранами.

ML Module: TFLiteYoloDetector (обробка кадру), SimpleByteTracker (трекінг), InferenceEngine (асинхронна обгортка), PostProcessor (злиття детекцій).

DJI Module (SDK v4): ConnectionManager, VideoStreamManager, TelemetryManager, CameraManager, MediaManager.

Data Module: AppDatabase (Room, 3 таблиці), Repositories.

2.7 Діаграма послідовності обробки відеокадру

Діаграму послідовності обробки одного відеокадру від камери дрона до відображення результатів наведено на рисунку 2.8.

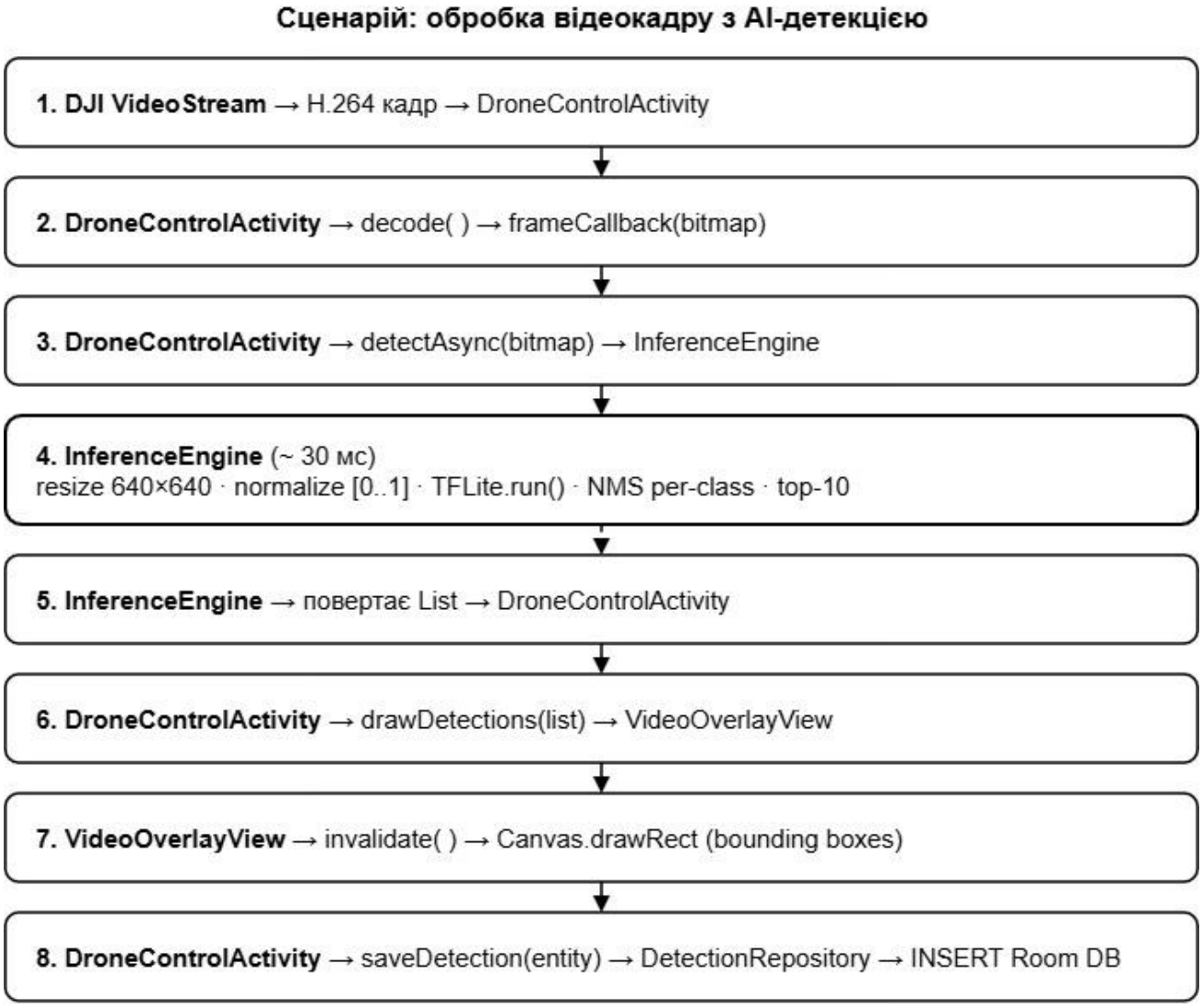


Рис. 2.8 Діаграма послідовності (Sequence Diagram) обробки відеокадру

Послідовність обробки кадру:

1. DJI Camera генерує H.264/H.265 потік та передає через OcuSync 2.0.
2. VideoFeeder.VideoDataListener отримує сирі байти та передає у DJICodecManager.
3. DJICodecManager декодує через апаратний MediaCodec та рендерить на TextureView.
4. onSurfaceTextureUpdated() захоплює Bitmap через getBitmap(reusableBitmap).
5. bitmap.copy() створює незмінну копію для ML-потоку.
6. TFLiteYoloDetector.detectObjects() виконує resize, інференс та постпроцесинг.
7. SimpleByteTracker.update() призначає track ID та стабілізує bounding boxes.
8. VideoOverlayView.drawDetections() малює рамки через Canvas API.
9. Позиція дрона оновлюється на карті Mapbox.

Для забезпечення продуктивності використовується механізм reusable bitmap. Без нього кожен виклик TextureView.getBitmap() створює новий об'єкт у пам'яті, що при 30 FPS генерує значний обсяг сміття та призводить до затримок. Заздалегідь виділений Bitmap передається через getBitmap(reusableBitmap), що знижує навантаження на збирач сміття.

Копіювання кадру через bitmap.copy() перед передачею до ML-потоку необхідне для потокобезпечності. TextureView оновлюється з головного потоку, а обробка нейромережею відбувається на фоновому. Без копіювання зображення може бути перезаписане наступним кадром під час обробки попереднього.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Реалізація ML-детектора (TFLiteYoloDetector)

Клас TFLiteYoloDetector є ядром ML-конвеєра. Код ініціалізації детектора наведено у лістингу Б.1 (Додаток Б), що включає завантаження моделі, створення інтерпретатора з GPU-делегатом та визначення layout вихідного тензора.

Модель завантажується через MappedByteBuffer (файл мапується з APK без копіювання у Java heap), мітки класів з assets/labels.txt. Інтерпретатор використовує GpuDelegate з автоматичним fallback на CPU з 4 потоками [6].

Pre-allocated буфери є ключовою оптимізацією для стабільного FPS без затримок збирача сміття. Їх опис наведено у таблиці 3.1.

Таблиця 3.1

Pre-allocated буфери ML-детектора

Буфер	Тип	Розмір	Призначення
inputBuffer	ByteBuffer (float32)	4.7 МБ	Вхідний тензор [1, 640, 640, 3]
outputBuffer	Array<Array<FloatArray>>	328 КБ	Вихідний тензор [1, 10, 8400]
pixels	IntArray	1.6 МБ	ARGB пікселі 640×640
scaleBitmap	Bitmap ARGB_8888	1.6 МБ	Цільовий bitmap для resize

Детектор автоматично визначає формат вихідного тензора YOLOv8 TFLite при першому запуску.

3.2 Препроцесинг та постпроцесинг зображень

На рисунку 3.1 наведено блок-схему препроцесингу зображень з деталізацією кожного етапу.

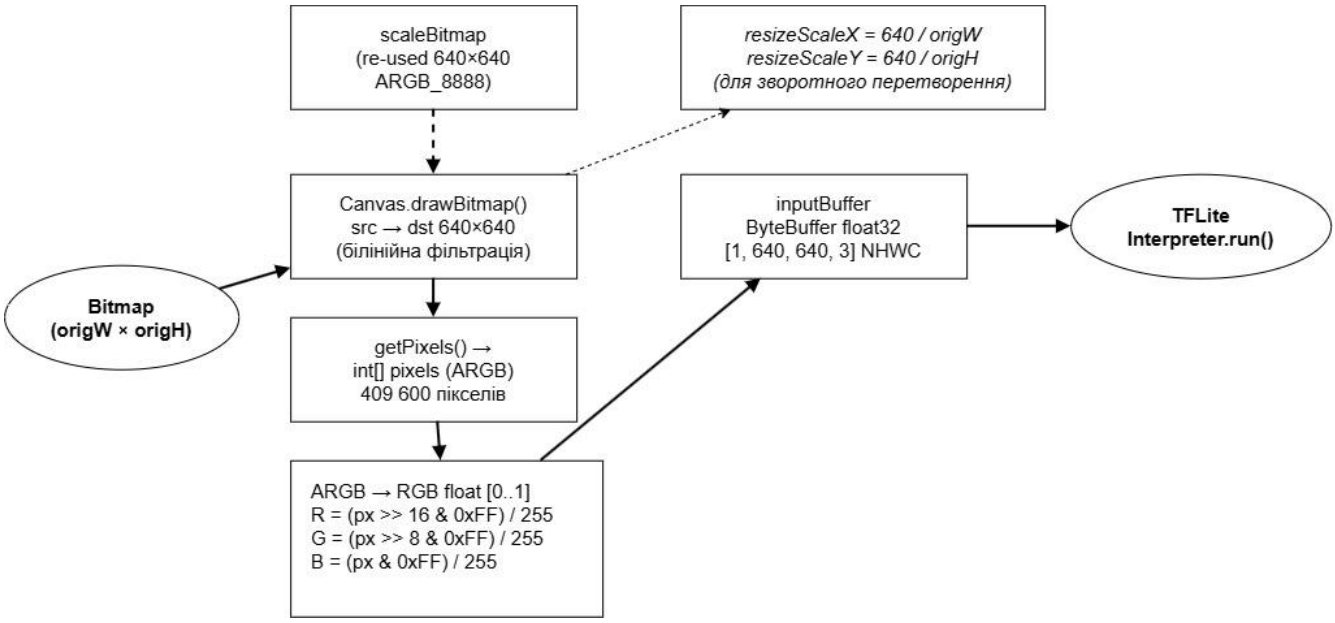


Рис. 3.1 Блок-схема етапів препроцесингу зображень у TFLiteYoloDetector

Препроцесинг виконує три операції без виділення нової пам'яті: масштабування до 640×640 через Canvas, копіювання пікселів у pre-allocated масив та конвертація ARGB → float RGB [0..1]. Коефіцієнти масштабування зберігаються для зворотного перетворення координат.

Постпроцесинг реалізує багаторівневу фільтрацію. Код геометричних фільтрів та NMS наведено у лістингу Б.2 (Додаток Б).

Алгоритм NMS працює окремо для кожного класу: детекції сортуються за впевненістю, кандидати з IoU вище порогу 0.45 відкидаються як дублікати. Поріг підвищено зі стандартного 0.30, щоб не видаляти реальні міни розташовані поруч.

Фільтр темних зон відкидає детекції в областях з низькою яскравістю пікселів - виникають при втраті відеосигналу. Grid-pattern фільтр відкидає всі детекції, якщо після NMS залишається більше 5 у вигляді регулярної сітки - характерний артефакт на однорідних текстурах.

3.3 Трекінг об'єктів (SimpleByteTracker)

SimpleByteTracker реалізує спрощений алгоритм ByteTrack [7] для стабілізації bounding boxes між кадрами відео. Код трекера наведено у лістингу Б.3 (Додаток Б).

На першому етапі детекції з впевненістю вище 0.5 зіставляються з активними треками за IoU, на другому - решта з тимчасово втраченими, що дозволяє відновлювати трекінг після зміни ракурсу. Треки без оновлень протягом 30 кадрів видаляються.

3.4 Інтеграція з DJI Mobile SDK v4

DJI Mobile SDK v4 забезпечує доступ до БПЛА DJI до 2022 року (Mavic 2, Phantom 4, Matrice 200/600) [12]. Інтеграція реалізована через 6 менеджерів, кожен відповідає за окрему підсистему.

DJIConnectionManager керує підключенням; після реєстрації SDK автоматично перевіряється наявність вже підключеного пристрою для коректної роботи при перезапуску застосунку.

DJIVideoStreamManager декодує H.264/H.265 через апаратний MediaCodec на TextureView; для ML-потоку створюється незмінна копія кадру (рис. 3.2).

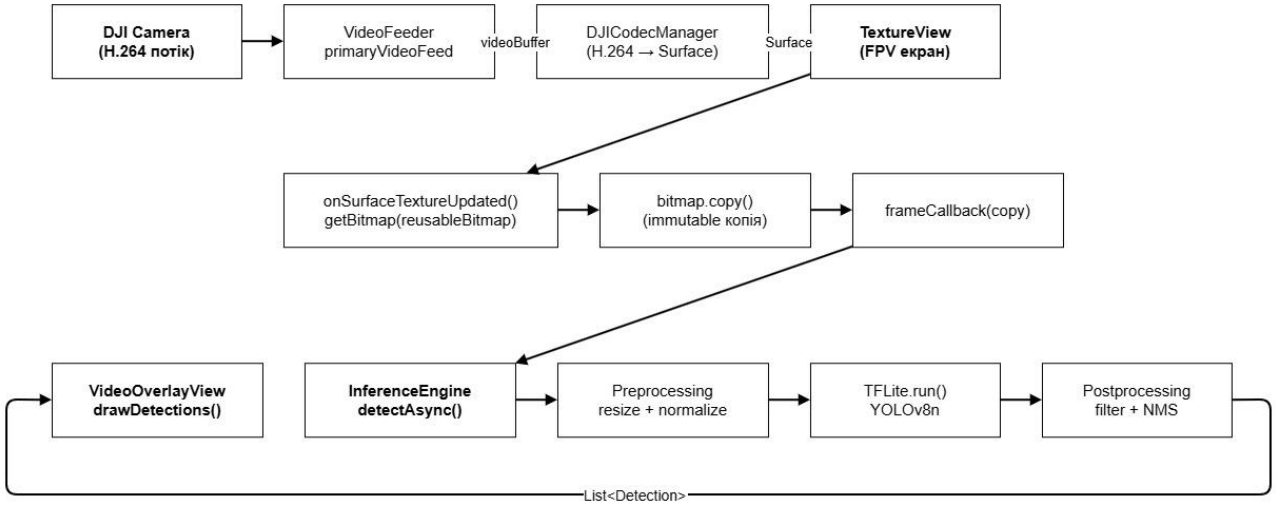


Рис. 3.2 Діаграма потоку відеоданих: камера DJI → декодер → TextureView → ML

DJITelemetryManager передає телеметрію в UI через StateFlow з фільтрацією GPS-дрифту (зміни менше 3 м ігноруються) та захистом від хибного показника батареї 0%.

Обробка кадрів виконується асинхронно - якщо попередній кадр ще обробляється нейромережею, новий пропускається. Це забезпечує обробку тільки актуальних даних без накопичення черги.

3.5 Реалізація відео потоку та декодування

Модуль video містить три класи. VideoFrameProcessor буферизує кадри у черзі і відкидає нові при перевантаженні, запобігаючи переповненню RAM.

VideoFileProcessor витягує кадри з відеофайлу і передає на детекцію у фоновому потоці для тестування моделі без дрона.

VideoDecoder реалізує апаратне декодування через MediaCodec як резерв, коли DJICodecManager недоступний.

3.6 Реалізація телеметрії та GPS

Телеметрія дрона передається через data-клас Telemetry (координати, висота, швидкість, напрямок, батарея, режим) і оновлюється через StateFlow з автоматичним відображенням в інтерфейсі.

Горизонтальна швидкість обчислюється з velocityX і velocityY, відстань від Home за формулою гаверсинусів. Параметри камери та кут gimbal передаються окремими StateFlow.

На MapView позиція дрона відображається синім маркером з лінією напрямку. GPS-координати переводяться в екранні через різницю відносно позиції дрона.

3.7 Реалізація бази даних Room

Room DAO інтерфейси наведено у лістингу Б.6 (Додаток Б). Для кожної таблиці реалізовано CRUD-операції як suspend-функції для асинхронного виконання.

DetectionRepository додає логіку збереження знімків на диск у форматі JPEG та генерацію унікального ідентифікатора польоту.

Міграція реалізована через fallbackToDestructiveMigration() - автоматичне перестворення таблиць при зміні схеми.

3.8 Реалізація картографічного модуля Mapbox

Mapbox Android SDK з офлайн-підтримкою реалізовано через кастомний MapView на базі Canvas API з відображенням позиції БПЛА та напрямку руху.

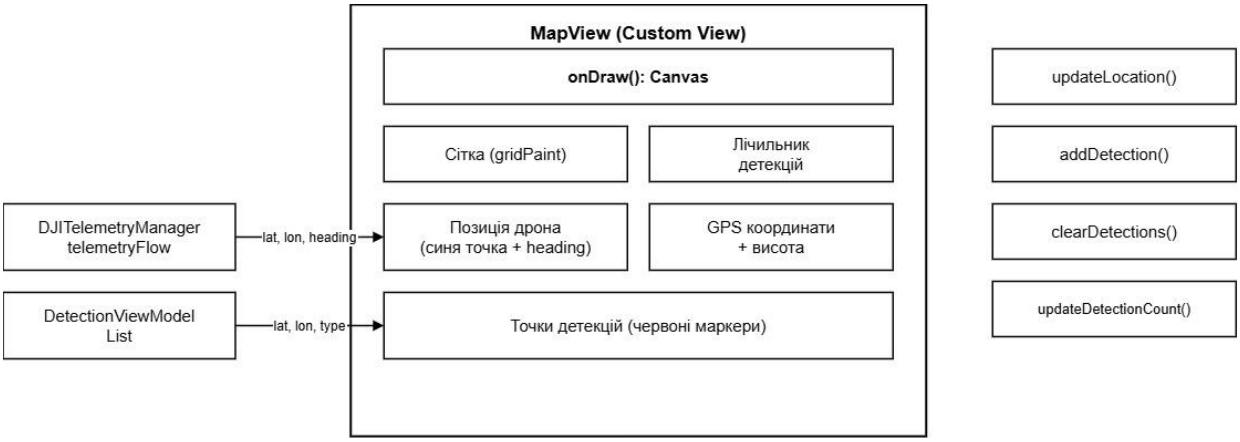


Рис. 3.3 Структура Mapbox інтеграції: MapView, маркери, кластери

MapView відображає позицію дрона на основі телеметрії з підтримкою стандартного та супутникового режимів і офлайн-тайлів. Access Token зберігається у local.properties поза системою контролю версій [14].

3.9 Реалізація інтерфейсу користувача

DroneControlActivity об'єднує FPV відеопотік, VideoOverlayView, панель телеметрії, індикатор gimbal pitch, карту та кнопки керування (лістинг Б.7).

VideoOverlayView малює bounding boxes через Canvas API з окремим кольором для кожного класу мін та масштабуванням координат під розмір екрану.

DjiGimbalPitchIndicatorView - вертикальний слайдер на Canvas API для керування кутом камери (-90°…+30°) з червоним індикатором (лістинг Б.8).

3.10 Відмовостійкість та crash logging

Код crash logger наведено у лістингу Б.9 (Додаток Б).

Crash logger перехоплює необроблені помилки та зберігає їх у файл на пристрої, що є важливим при тестуванні без підключення через ADB.

Callbacks DJI SDK обгорнуті у try-catch для ізоляції помилок; ML-операції виконуються у фонових корутинах, гарантуючи відгук інтерфейсу.

DiagnosticActivity відображає стан компонентів DJI SDK, спрощуючи діагностику при роботі з різними моделями дронів.

3.11 Конфігурація збірки та залежності

Збірка виконується через Gradle з фільтром arm64-v8a та мініфікацією у release-режимі. DJI API Key зберігається у local.properties поза системою контролю версій. Основні залежності наведено у таблиці 3.2.

Таблиця 3.2

Основні залежності проекту (build.gradle.kts)

Бібліотека	Призначення
DJI Mobile SDK v4 + UX SDK	Керування дроном та UI віджети (FPVWidget)
TensorFlow Lite + GPU Delegate + Support	ML-інференс на мобільному пристрої
Room + KSP compiler	Локальна SQLite база даних з генерацією DAO
Mapbox Android SDK	Інтерактивна карта з офлайн-підтримкою
Retrofit + OkHttp + Gson	REST API клієнт з серіалізацією
ExoPlayer (Media3)	Відтворення відео у галереї та тестовому режимі
Coil	Асинхронне завантаження зображень з кешуванням
PhotoView	Зум та пан для перегляду фото з детекціями
PermissionX	Спрощений запит runtime permissions
WorkManager	Фонова синхронізація та оновлення моделі
Coroutines (core + android)	Асинхронне програмування без callbacks
Lifecycle (ViewModel + LiveData)	MVVM компоненти з lifecycle-awareness
ExifInterface	Зчитування EXIF-метаданих для ротації фото

4 ТЕСТУВАННЯ ТА РЕЗУЛЬТАТИ

4.1 Навчання моделі YOLOv8

Датасет з аерофотознімків DJI Mavic 3 та Matrice 30T зібрано на полігоні з макетами мін, розмітку виконано у CVAT [21] у форматі YOLO txt (рис. 4.1).

Рис. 4.1 Розмітка аерофотознімка у CVAT: bounding box міни PMN-2 на фоні рослинності

Характеристики датасету наведено у таблиці 4.1. Включення негативних зразків (близько 40% фото без мін) виявилось критичним - без них модель генерувала хибні спрацювання на будь-якому фоні.

Таблиця 4.1

Характеристики датасету для навчання

Параметр	Значення	Коментар
Train	107 зображень (95 анотованих)	~91.6% з анотаціями
Validation	22 зображення (20 анотоване)	~95.5% з анотаціями

Продовження таблиці 4.1

Всього	129 зображень, 115 анотацій	Компактний спеціалізований датасет
Негативні зразки	~51 фото (~40%)	Критично для зменшення FP
Інструмент	CVAT	Ручна розмітка bounding box
Формат	YOLO txt	class cx cy w h (normalized)
Висота	5-15 м	Оптимум 8-15 м
Джерело	DJI Mavic 2, тренувальний полігон	Реальні умови зйомки

Конфігурацію навчання наведено у таблиці 4.2. Трансферне навчання на базі YOLOv8n з вагами COCO забезпечує прийнятну точність навіть на невеликому датасеті.

Таблиця 4.2

Конфігурація навчання (mine_detector_v3)

Параметр	Значення	Обґрунтування
model	yolov8n.pt (COCO)	Трансферне навчання з 80 класів COCO
epochs	200	З early stopping patience=40
batch	8	Обмежено пам'яттю GPU (8 ГБ)
imgsz	640	Стандарт YOLO, баланс точн./швидк.
optimizer	AdamW	Краща генералізація для малих DS
lr0 / lrf	0.002 / 0.01	Знижений LR для fine-tuning
cos_lr	True	Cosine annealing scheduler
warmup_epochs	3	Плавний старт навчання
weight_decay	0.0008	L2 регуляризація проти overfitting
amp	True	Mixed precision (FP16 на GPU)

Функція втрат YOLOv8 включає box loss, cls loss та dfl loss. Стабільне зниження всіх компонентів під час навчання свідчить про відсутність перенавчання. Для збільшення різноманітності тренувальних даних застосовано аугментації, параметри у таблиці 4.3.

Таблиця 4.3

Аугментація даних при навчанні

Параметр	Значення	Фізичне обґрунтування
degrees	40.0	Міни лежать під будь-яким кутом
flipud / fliplr	0.5 / 0.5	Інваріантність до орієнтації
scale	0.2	Різна висота дрона (±20% масштаб)
translate	0.1	Зсув об'єкта в кадрі
perspective	0.0002	Мінімальна перспективна деформація
hsv_h / hsv_s / hsv_v	0.015 / 0.3 / 0.3	Різне освітлення та час доби
mosaic	0.4	Комбінація 4 зображень (40%)
mixup	0.0	Вимкнено - шкодить малим датасетам
copy_paste	0.0	Потребує сегментаційних масок

Конвертація через ONNX до TFLite (onnx2tf) забезпечує коректну нормалізацію координат для Android-постпроцесингу. Розмір моделі - ~6 МБ.

4.2 Результати навчання та метрики

Навчання виконано на NVIDIA GPU протягом 200 епох; конвергенція - приблизно на 160-й епосі. Графіки втрат та метрик наведено на рисунку 4.2.

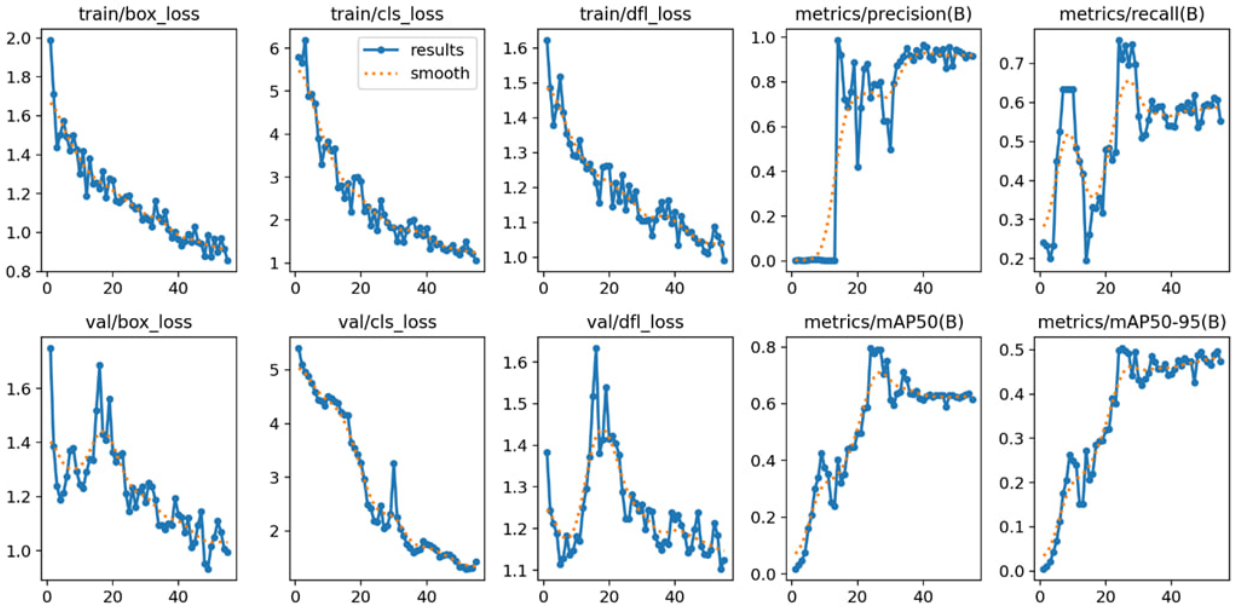


Рис. 4.2 Графіки функції втрат та метрик навчання YOLOv8n

Loss стабільно знижується протягом перших 160 епох з подальшою стабілізацією, mAP@50 досягає 80.4%. Результати на валідаційному наборі наведено у таблиці 4.4.

Таблиця 4.4

Загальні метрики на валідаційному наборі

Метрика	Значення	Інтерпретація
mAP@50	80.4%	Середня точність при IoU >= 0.50
mAP@50-95	53.8%	Середня точність при IoU 0.50-0.95
Precision	80.4%	Частка правильних серед усіх детекцій
Recall	87.5%	Частка знайдених серед усіх наявних мін

Recall (87.5%) вищий за Precision (80.4%), що є бажаним для задачі детекції мін - пропущена міна є значно небезпечнішою за хибне спрацювання. Матрицю плутанини (confusion matrix) наведено на рисунку 4.3.

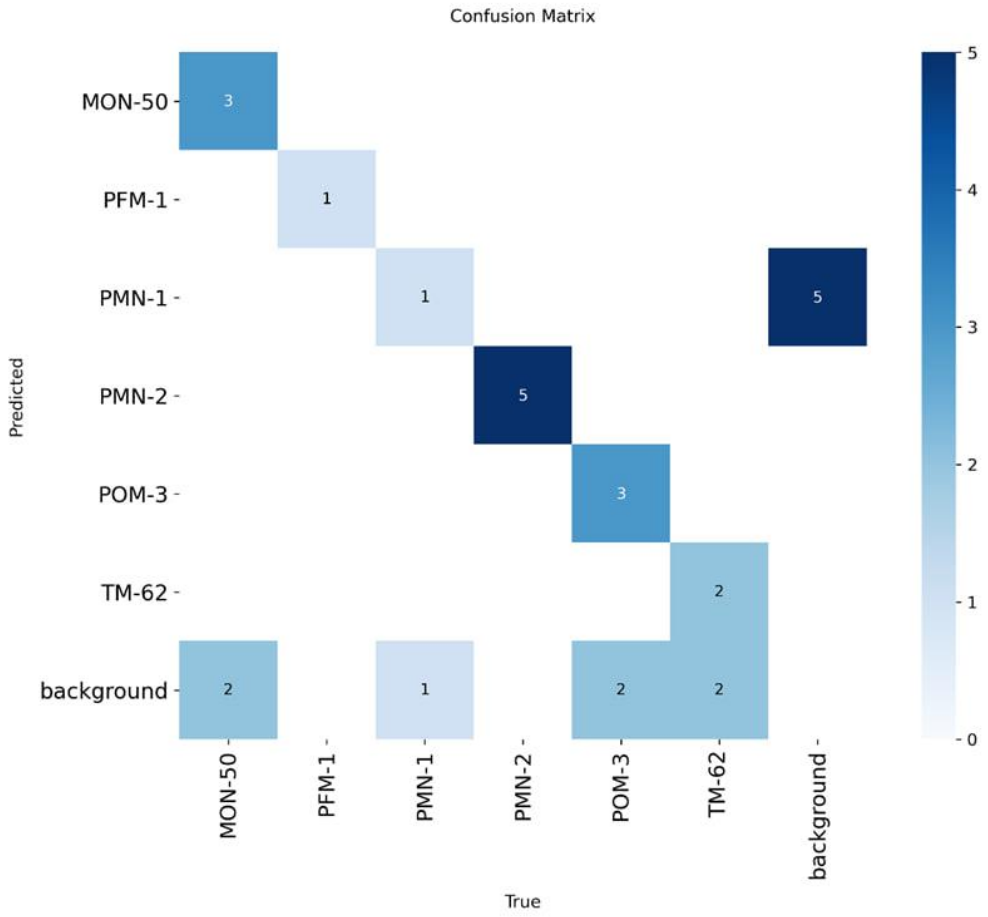


Рис. 4.3 Матриця плутанини (Confusion Matrix) для 6 класів мін

Найбільша плутанина - між PMN-1 та PMN-2 через схожу форму (~110-120 мм), невідрізнювану з висоти понад 10 м; модель також плутає фон з MON-50 та POM-3, що підтверджує необхідність покласових порогів.

Найкращі результати - TM-62 та MON-50 завдяки розміру; найгірший - PFM-1 (62 мм). F1-Confidence криву наведено на рисунку 4.4.

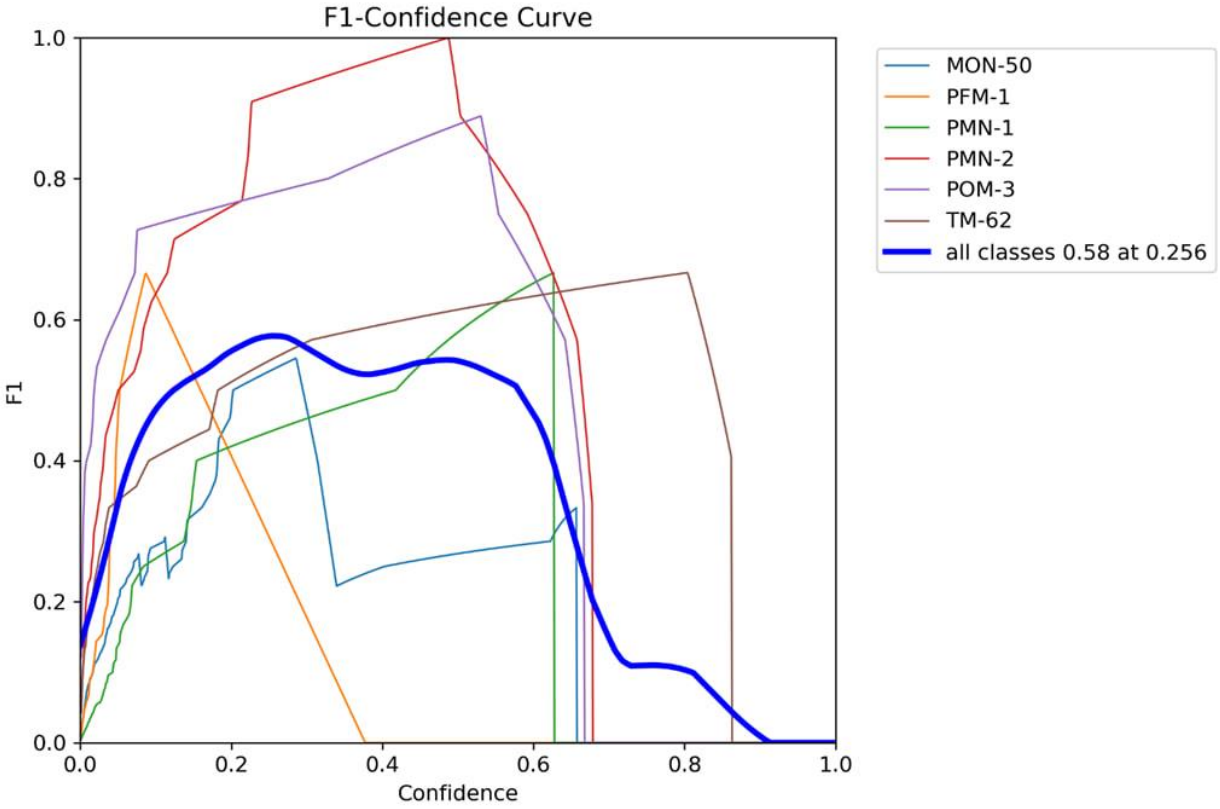


Рис. 4.4 F1-Confidence крива для 6 класів мін

Графік показує залежність F1-score від порогу впевненості. Максимальне значення F1 для всіх класів (0.58) досягається при порозі 0.256. Детальний аналіз результатів по кожному класу наведено у таблиці 4.5.

Таблиця 4.5

Покласний аналіз результатів детекції

Клас	Train зразків	Precision	Recall	Проблеми
MON-50	16	Висока	Висока	Унікальна форма, стабільна детекція
PFM-1	10	Середня	Низька	Малий розмір (60 мм), мало зразків
PMN-1	24	Середня	Висока	Плутанина з PMN-2 (схожі кола)

PMN-2	8	Середня	Висока	Плутанина з PMN-1
POM-3	11	Середня	Середня	Малий розмір (60 мм), мало на поверхні
TM-62	24	Висока	Висока	Великий розмір (320 мм), чітка форма

PMN-1 та PMN-2 мають найбільшу взаємну плутанину через візуальну схожість. PFM-1 показує найнижчі результати через малий фізичний розмір. TM-62 та MON-50 детектуються найкраще завдяки великому розміру та унікальній формі.

4.3 Тестування на реальних аерофотознімках

Тестування проводилось на полігоні з макетами мін на різних типах поверхні. Результати детекції на тестових зображеннях наведено на рисунку 4.5.

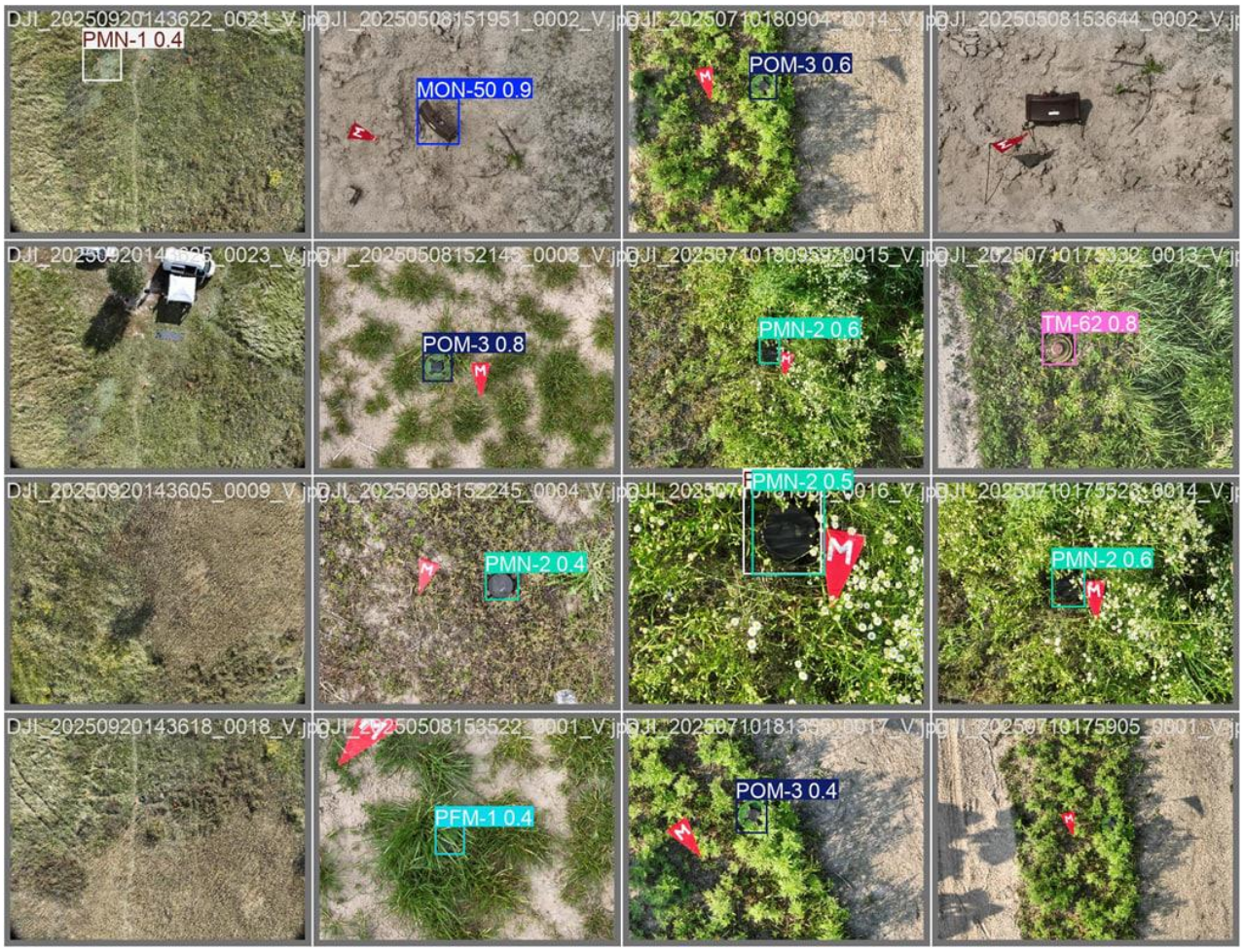


Рис. 4.5 Результати детекції на тестових аерофотознімках

Тестування показало наступні результати по висотах:

— TM-62 та MON-50 впевнено детектуються на висотах до 15 м завдяки великому розміру;
— PMN-1 та PMN-2 стабільно розпізнаються на 5-10 м, вище 10 м детекція погіршується;
— PFM-1 виявляється лише до 8 м через малий розмір міни (62 мм);
— фільтр темних зон коректно пригнічує хибні детекції при втраті відеосигналу;
— grid-pattern detection відкидає артефакти на однорідних текстурах.

На 28 кадрах реального польоту покласові пороги знизили кількість хибних спрацювань з 4-7 до 0-3 на кадр зі збереженням реальних детекцій.

4.4 Тестування продуктивності на мобільних пристроях

Функціональне тестування на трьох пристроях (Redmi K20 Pro, Samsung Note 20 Ultra, DJI Smart Controller) підтвердило стабільну роботу застосунку.

Тестування з DJI Mavic 2 Pro підтвердило коректну роботу телеметрії, відеопотоку, керування gimbal та доступу до SD-карти. Час до першого кадру - 8-15 с. Результати продуктивності наведено у таблиці 4.6.

Таблиця 4.6

Продуктивність ML на мобільному пристрої

Пристрій	SoC	Backend	Інференс (мс)
Samsung Note 20 Ultra	Exynos 990	GPU (OpenCL)	15-20
Redmi K20 Pro	Snapdragon 855	GPU (OpenCL)	20-25
DJI Smart Controller	Qualcomm (custom)	CPU x4	45-55

4.5 Тестування інтеграції з БПЛА

Інтеграцію протестовано на DJI Mavic 2 Pro зі стандартним пультом керування:

— підключення: стабільне через OcuSync 2.0, auto-reconnect працює при короткочасній втраті зв'язку;

— FPV: стабільний відеопотік, затримка ~100 мс;

— телеметрія: GPS координати підтверджено порівнянням з DJI GO 4, похибка менше 2 м;

— підвіс: плавне керування pitch через слайдер;

— AI-детекція: працює, середня кількість хибних спрацювань - до 3 на кадр на складних текстурах;

— карта: позиція дрона на карті відповідає реальним координатам;

— довготривалий тест: 30 хвилин безперервної роботи без збоїв;

— USB reconnect: коректне перепідключення при від'єднанні та під'єднанні кабелю.

Основною проблемою стали хибні спрацювання на однорідних текстурах (трава, ґрунт), вирішені покласовими порогами та grid-pattern suppression (розділ 2.3). Загалом система готова до практичного використання.

4.6 Демонстрація інтерфейсу користувача

На рис. 4.6 показано головне меню з розділами Drone Control, Test Mode, Gallery та Settings. Зверху відображається стан підключення до дрона.

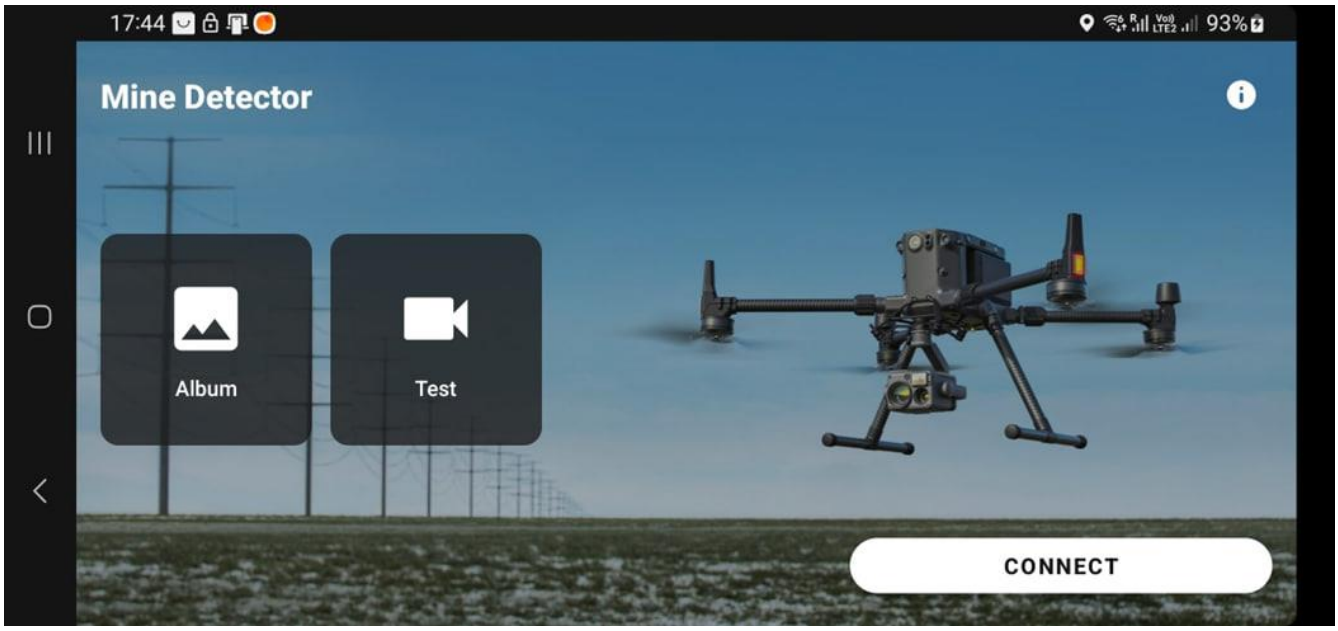


Рис. 4.6 Екран головного меню застосунку MineDetector

На рисунку 4.7 - робочий екран з FPV-відео, bounding boxes, телеметрією, індикатором підвісу, мінікартою та кнопками керування.

Рис. 4.7 Екран керування дроном DroneControlActivity з AI-детекцією у реальному часі

На рисунку 4.8 - тестовий режим: детекція на фото з галереї без підключення до дрона з відображенням bounding boxes.

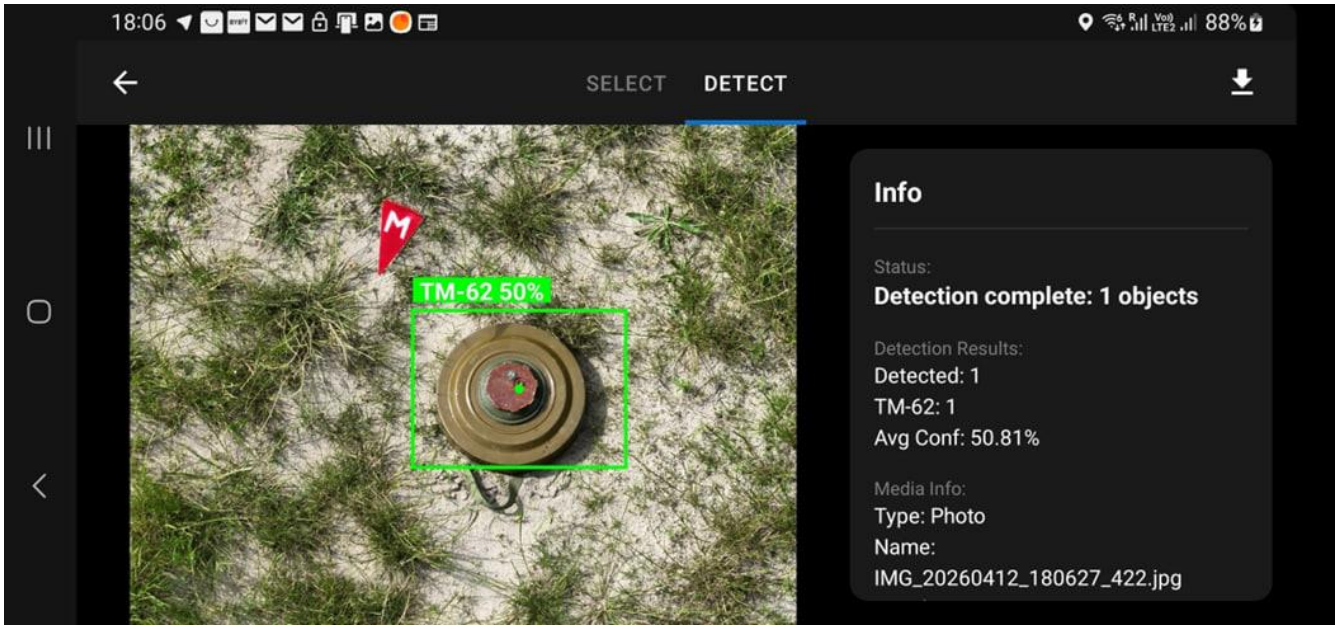


Рис. 4.8 Екран тестового режиму TestVideoActivity з результатами детекції на фото

На рисунку 4.9 зображено галерею з двома вкладками: "All" (медіа з пристрою) та "Drone" (медіа з SD-карти дрона).

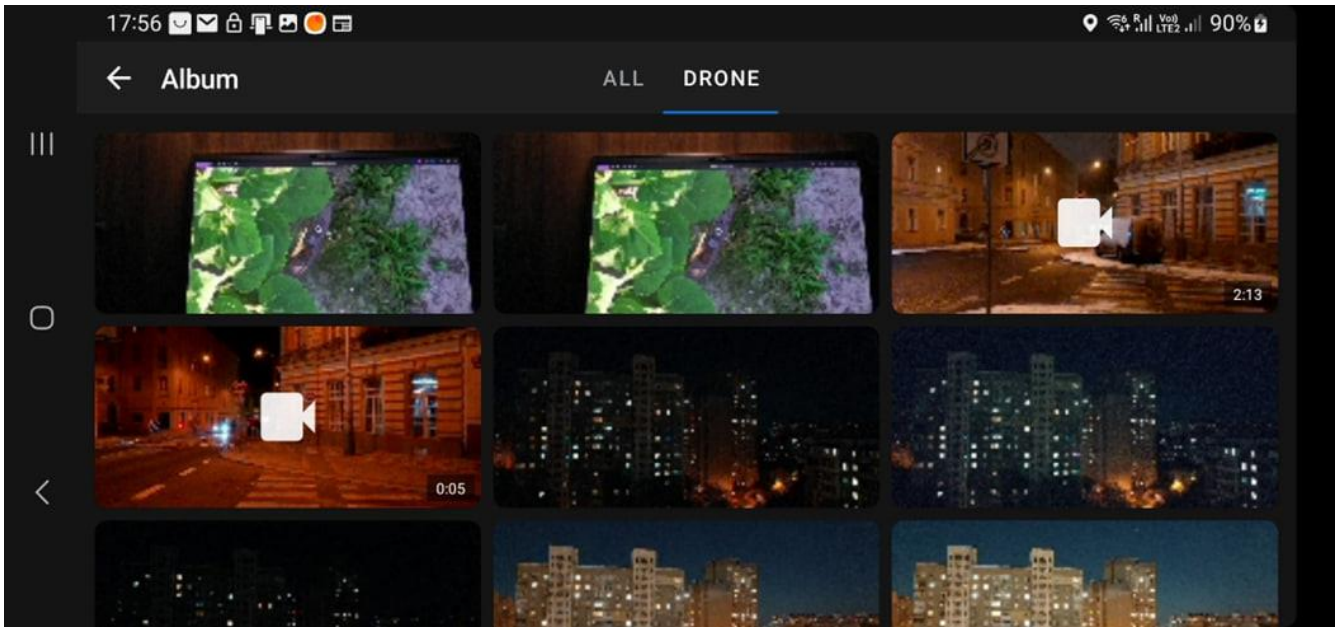


Рис. 4.9 Екран галереї GalleryActivity з вкладками "All" та "Drone"

На рисунку 4.10 зображено панель передпольотної перевірки, що відображає стан компонентів дрона (компас, IMU, батарея, SD-карта, gimbal) та кнопку увімкнення AI-детекції.

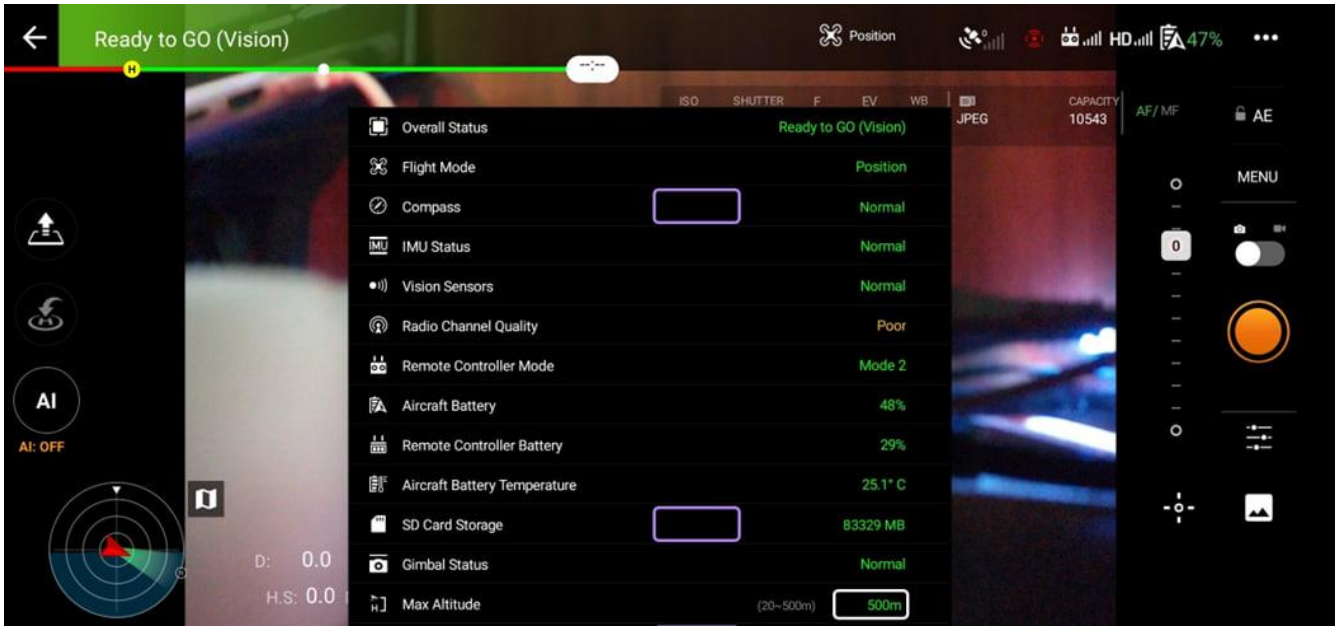


Рис. 4.10 Панель передпольотної перевірки DJI з параметрами дрона

Інтерфейс орієнтований на оператора БПЛА: вся інформація на одному екрані, елементи керування по краях для зручного доступу під час пілотування.

ВИСНОВКИ

У кваліфікаційній роботі розроблено Android-застосунок MineDetector для виявлення вибухонебезпечних предметів на аерофотознімках з БПЛА DJI.

У процесі виконання роботи отримано наступні результати:

1. Проаналізовано проблему мінної загрози та існуючі рішення, що залежать від хмарної інфраструктури або спеціалізованого обладнання. Обґрунтовано вибір YOLOv8n та стеку: Kotlin, TFLite, DJI Mobile SDK v4, Room, Mapbox.
2. Зібрано та анотовано датасет із 129 аерофотознімків з 6 типами мін у CVAT.
3. Навчено модель з трансферним навчанням: mAP@50 = 80.4%, Precision = 80.4%, Recall = 87.5%; конвертовано у TFLite (~6 МБ).
4. Розроблено застосунок з архітектурою MVVM + Repository та конвеєром реального часу: декодування, TFLite інференс, фільтрація, ByteTrack трекінг.
5. Реалізовано офлайн-карту Mapbox, телеметрію та керування підвісом.
6. Тестування на полігоні з макетами мін підтвердило продуктивність 15-20 FPS та повну автономність без інтернету.

Перспективи подальшого розвитку: інтеграція з DJI AI Inside, розширення датасету, перехід на YOLOv8s для кращої детекції дрібних об'єктів.

Робота пройшла апробацію. За її результатами опубліковано наступні тези доповідей:

1. Старцев М.С., Читулян В.О. Аналіз існуючих рішень та проектування мобільної системи виявлення вибухонебезпечних предметів із застосуванням БПЛА. // Матеріали Всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація». 01.05.2026, ДУІКТ, Київ, (подано до друку).

2. Старцев М.С., Читулян В.О. MineDetector: Мобільна система виявлення вибухонебезпечних предметів на основі БПЛА та нейронної мережі YOLOv8. // Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 23.04.2026, ДУІКТ, Київ, (подано до друку).

ПЕРЕЛІК ПОСИЛАНЬ

1. United Nations Mine Action Service (UNMAS). Ukraine: Mine contamination is lethal legacy of Russia's invasion [Електронний ресурс]. URL: https://news.un.org/en/story/2025/04/1161956 (дата звернення: 25.02.2026).

2. Landmine and Cluster Munition Monitor. Ukraine Country Profile [Електронний ресурс]. URL: https://www.the-monitor.org/country-profile/ukraine (дата звернення: 28.02.2026).

3. Державна служба України з надзвичайних ситуацій. Розмінування України [Електронний ресурс]. URL: https://mine.dsns.gov.ua/ (дата звернення: 01.03.2026).

4. Redmon J., Divvala S., Girshick R., Farhadi A. You Only Look Once: Real-Time Object Detection // Proc. IEEE/CVF CVPR. 2016. P. 779-788. URL: https://openaccess.thecvf.com/content_cvpr_2016/html/Redmon_You_Only_Look_CVPR_2016_paper.html (дата звернення: 15.01.2026).

5. Jocher G., Chaurasia A., Qiu J. Ultralytics YOLOv8 [Електронний ресурс]. 2023. URL: https://github.com/ultralytics/ultralytics (дата звернення: 15.01.2026).

6. Google AI Edge. LiteRT (formerly TensorFlow Lite): Deploy machine learning models on mobile and edge devices [Електронний ресурс]. URL: https://ai.google.dev/edge/litert (дата звернення: 10.02.2026).

7. Zhang Y., Sun P., Jiang Y. et al. ByteTrack: Multi-Object Tracking by Associating Every Detection Box // LNCS. Springer, 2022. P. 1-21. URL: https://link.springer.com/chapter/10.1007/978-3-031-20047-2_1 (дата звернення: 15.03.2026).

8. Ren S., He K., Girshick R., Sun J. Faster R-CNN: Towards Real-Time Object Detection // NeurIPS. 2015. P. 91-99. URL: https://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks (дата звернення: 15.03.2026).

9. Lin T.-Y., Maire M., Belongie S. et al. Microsoft COCO: Common Objects in Context [Електронний ресурс] // Lecture Notes in Computer Science. Springer, 2014.

P. 740-755. URL: https://link.springer.com/chapter/10.1007/978-3-319-10602-1_48 (дата звернення: 15.03.2026).

10. Liu W., Anguelov D., Erhan D. et al. SSD: Single Shot MultiBox Detector [Електронний ресурс] // Lecture Notes in Computer Science. Springer, 2016. P. 21-37. URL: https://link.springer.com/chapter/10.1007/978-3-319-46448-0_2 (дата звернення: 15.03.2026).

11. Kotlin Programming Language. Coroutines Guide [Електронний ресурс]. JetBrains. URL: https://kotlinlang.org/docs/coroutines-guide.html (дата звернення: 10.02.2026).

12. DJI Mobile SDK v4 Developer Documentation [Електронний ресурс]. DJI Technology. URL: https://developer.dji.com/mobile-sdk/documentation/ (дата звернення: 20.01.2026).

13. Android Developers. Guide to App Architecture [Електронний ресурс]. Google. URL: https://developer.android.com/topic/architecture (дата звернення: 05.02.2026).

14. Mapbox Android SDK Documentation [Електронний ресурс]. Mapbox Inc. URL: https://docs.mapbox.com/android/ (дата звернення: 15.02.2026).

15. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition [Електронний ресурс] // Proceedings of the IEEE/CVF CVPR. 2016. P. 770-778. URL: https://ieeexplore.ieee.org/document/7780459 (дата звернення: 15.03.2026).

16. Bochkovskiy A., Wang C.-Y., Liao H.-Y.M. YOLOv4: Optimal Speed and Accuracy of Object Detection [Електронний ресурс]. 2020. URL: https://github.com/AlexeyAB/darknet (дата звернення: 15.03.2026).

17. Howard A.G., Zhu M., Chen B. et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications [Електронний ресурс]. Google Research, 2017. URL: https://research.google/blog/mobilenets-open-source-models-for-efficient-on-device-vision/ (дата звернення: 15.03.2026).

18. Wang C.-Y., Bochkovskiy A., Liao H.-Y.M. YOLOv7: Trainable Bag-of-Freebies // Proc. IEEE/CVF CVPR. 2023. P. 7464-7475. URL [Електронний ресурс]: https://openaccess.thecvf.com/content/CVPR2023/html/Wang_YOLOv7_Trainable_Ba

g-of-Freebies_Sets_New_State-of-the-Art_for_Real-Time_Object_Detectors_CVPR_2023_paper.html (дата звернення: 15.03.2026).

19. Padilla R., Netto S.L., da Silva E.A.B. A Survey on Performance Metrics for Object-Detection Algorithms [Електронний ресурс] // Proceedings of IWSSIP. IEEE, 2020. P. 237-242. URL: https://ieeexplore.ieee.org/document/9145130 (дата звернення: 15.03.2026).

20. Sandler M., Howard A., Zhu M. et al. MobileNetV2: Inverted Residuals and Linear Bottlenecks // Proc. IEEE/CVF CVPR. 2018. P. 4510-4520. URL: https://ieeexplore.ieee.org/document/8578572 (дата звернення: 15.03.2026).

21. CVAT.ai. Computer Vision Annotation Tool: Documentation [Електронний ресурс]. URL: https://docs.cvat.ai/ (дата звернення: 09.12.2025).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ

Android-додаток для керування дроном і аналізу відеопотоку DJI з детекцією мін на базі YOLO

Виконав студент 4 курсу
групи ТЦР-42
Старцев Микола Сергійович
Керівник роботи
ст. викладач кафедри ТЦР Читулян Вадим Олегович

Київ - 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

• **Мета роботи** – автоматизоване виявлення вибухонебезпечних предметів на відеопотоці з БПЛА DJI за допомогою розробленого Android-застосунку на основі нейронної мережі YOLOv8.

• **Об'єкт дослідження** – процес автоматизованого виявлення вибухонебезпечних предметів на відеопотоці з БПЛА.

• **Предмет дослідження** – програмне забезпечення для детекції мін у реальному часі на відеопотоці з БПЛА з використанням нейронної мережі YOLOv8 на мобільному пристрої.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз предметної області та існуючих рішень для дистанційного виявлення мін.
2. Обгрунтувати вибір архітектури нейронної мережі (YOLOv8n) та технологій розробки мобільного застосунку.
3. Зібрати та анотувати датасет, навчити модель YOLOv8n та конвертувати у формат TFLite.
4. Розробити Android-застосунок MineDetector з повним конвеєром обробки відеопотоку у реальному часі: декодування, ML-детекція, трекінг, візуалізація, карта Mapbox.
5. Провести тестування системи на полігоні з макетами мін та оцінити продуктивність на мобільних пристроях.

АНАЛІЗ АНАЛОГІВ

Функціональність / Показник	**SpotlightAI ONSITE**	**Dropla BLUE EYES**	**Blackthorn.ai**	**MineDetector**
Робота у реальному часі	+	-	+	+
Мобільний застосунок	-	-	-	+
Офлайн-робота без інтернету	+	+	-	+
Підтримка споживчих дронів DJI	-	-	-	+
Відкритий вихідний код	-	-	-	+
Відображення bbox у реальному часі	+	-	+	+
Підтримка термокамери	+	+	-	-

ВИМОГИ ДО ANDROID-ЗАСТОСУНКУ

Функціональні вимоги:

- детекція 6 типів мін (MON-50, PFM-1, PMN-1, PMN-2, POM-3, TM-62);
- підключення до дронів DJI з підтримкою Mobile SDK v4 через USB;
- відображення відео з камери дрона;
- відображення bounding boxes з класом та впевненістю;
- телеметрія: GPS, висота, швидкість, батарея;
- керування підвісом камери (pitch);
- тестовий режим детекції на фото без дрона;
- перегляд галереї дрона та пристрою;
- відображення позиції дрона на карті Mapbox.

Нефункціональні вимоги:

- повна автономність (без інтернету);
- всі ML-обчислення локально на пристрої;
- швидкість ML-обробки залежно від можливостей пристрою;
- підтримка Android 7.0+ (API 24);
- автоматичний перехід з GPU на CPU при відсутності підтримки;
- захищеність від крашів при втраті сигналу дрона.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Kotlin
Мова розробки Android

Android SDK
API 24+

TensorFlow Lite
ML-обробка на пристрої

YOLOv8n
Нейромережа детекції

DJI Mobile SDK v4
Інтеграція з дроном

Mapbox Android SDK
Офлайн-карти

Android Room
База даних SQLite

Python
Мова навчання моделі

CVAT
Анотація датасету

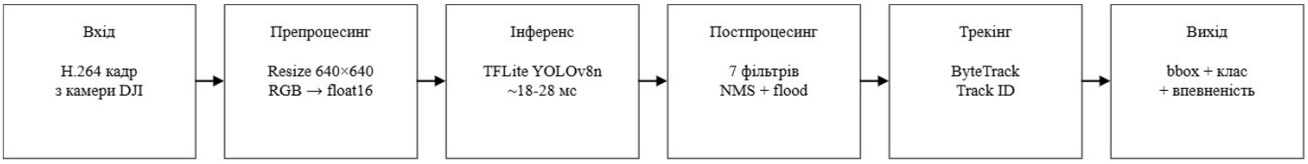
Android Studio
IDE розробки

Gradle
Система збірки

onnx2tf
PyTorch to TFLite конвертер

6

ML-КОНВЕЄР ДЕТЕКЦІЇ



7

РЕЗУЛЬТАТИ НАВЧАННЯ МОДЕЛІ

Метрики на валідаційному наборі:

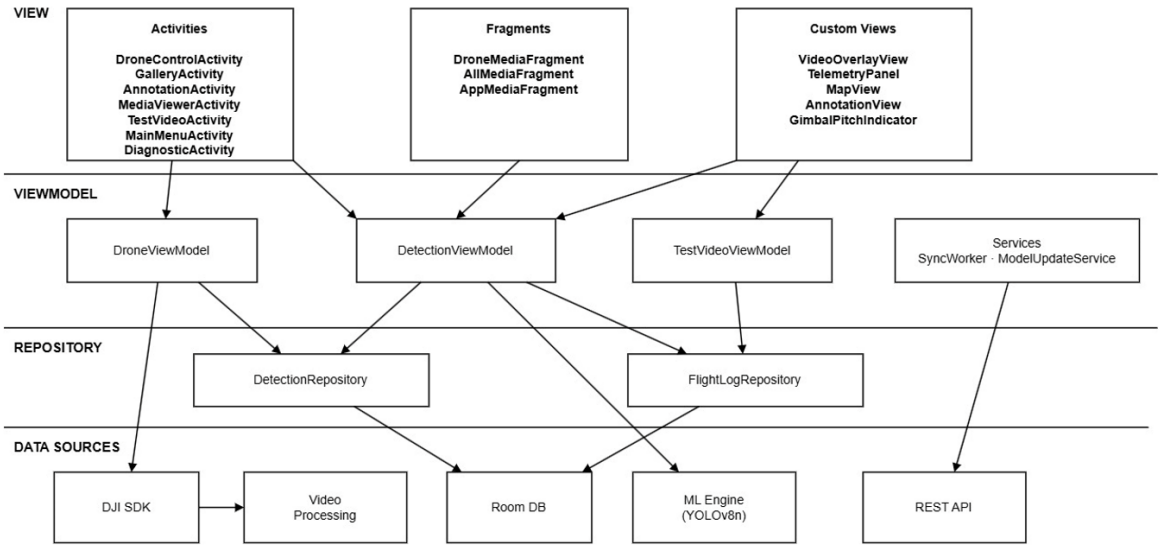
Метрика	Значення
mAP@50	80.4%
mAP@50-95	53.8%
Precision	80.4%
Recall	87.5%
F1 max	0.58 (при conf=0.256)

Датасет:

- 129 зображень (107 train + 22 val)
- 125 анотацій по 6 класах
- Анотація у CVAT
- Дрони: Mavic 3 Pro, Mavic 3 Enterprise, Mavic 3T, Matrice 30T
- Висота зйомки: 5-15 м
- 200 епох тренування
- Transfer learning з COCO
- Аугментація: mosaic, HSV, flip, обертання ±40°

8

АРХІТЕКТУРА ЗАСТОСУНКУ



ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ (ОПЕРАТОР БПЛА)

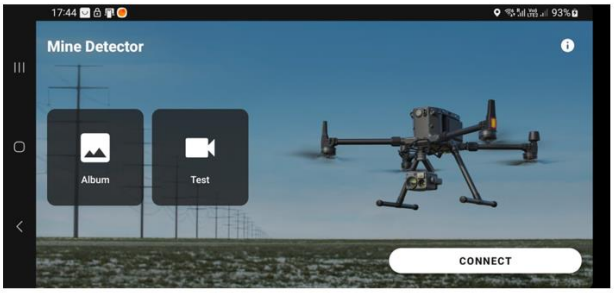
ТЕСТУВАННЯ ПРОДУКТИВНОСТІ

Пристрій	Chip	Backend	Інференс	FPS
Redmi K20 Pro	Snapdragon 855	GPU (OpenCL)	15-20 мс	15-20
Samsung Note 20 Ultra	Exynos 990	GPU (OpenCL)	20-25 мс	15-20
DJI Smart Controller	Qualcomm (custom)	CPU x4	45-55 мс	10-15

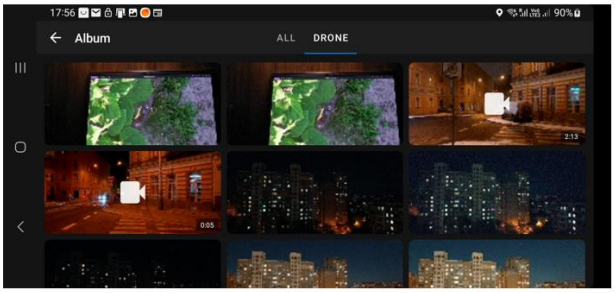
Тестування проведено на полігоні з макетами мін:

- Стабільне підключення до DJI Mavic 2 Pro через OcuSync 2.0;
- Хибні спрацювання: 0-3 на кадр після фільтрації;
- Робота протягом 30 хвилин без збоїв;
- Коректне відображення bounding boxes у реальному часі.

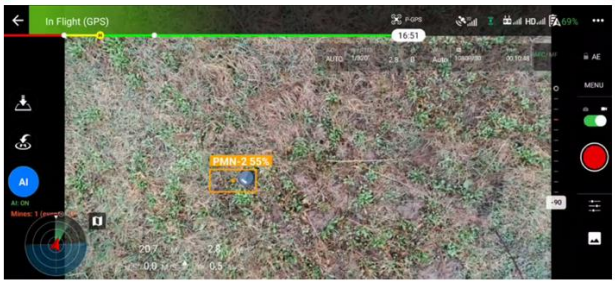
ЕКРАННІ ФОРМИ



Головне меню

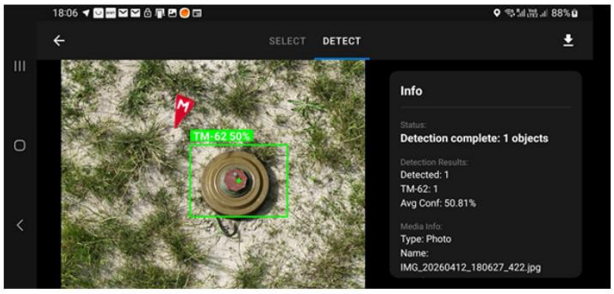


Галерея медіа

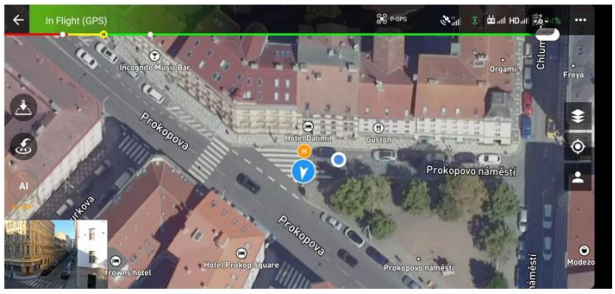


Екран керування дроном з AI-детекцією

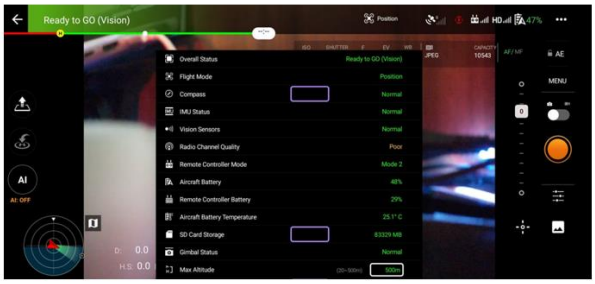
ЕКРАННІ ФОРМИ



Тестовий режим детекції на фото



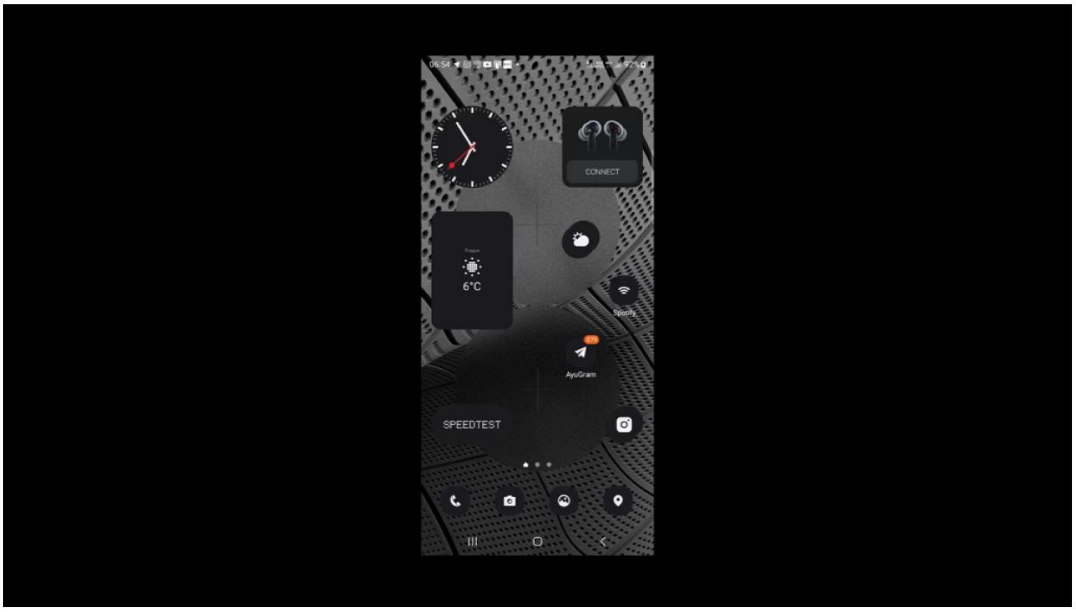
Карта Mapbox з позицією дрона



Панель передпольотної перевірки DJI

13

ДЕМОНСТРАЦІЯ РОБОТИ ЗАСТОСУНКУ



14

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Старцев М.С., Читулян В.О. Аналіз існуючих рішень та проектування мобільної системи виявлення вибухонебезпечних предметів із застосуванням БПЛА. // Матеріали Всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація». 01.05.2026, ДУІКТ, Київ, (подано до друку).

2. Старцев М.С., Читулян В.О. MineDetector: Мобільна система виявлення вибухонебезпечних предметів на основі БПЛА та нейронної мережі YOLOv8. // Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 23.04.2026, ДУІКТ, Київ, (подано до друку).

15

ВИСНОВКИ

1. Проведено аналіз проблеми мінної загрози в Україні та порівняльний огляд існуючих рішень для дистанційного виявлення мін.

2. Обґрунтовано вибір архітектури YOLOv8n та технологій мобільної розробки (Kotlin, TFLite, DJI SDK).

3. Зібрано та анотовано датасет: 129 аерофотознімків, 6 класів мін.

4. Навчено модель YOLOv8n з mAP@50 = 80.4%, Precision = 80.4%, Recall = 87.5%.

5. Розроблено Android-застосунок MineDetector з повним конвеєром обробки відеопотоку у реальному часі (15-20 FPS).

6. Реалізовано систему фільтрації хибних спрацювань.

7. Проведено тестування на полігоні з макетами мін, застосунок готовий до практичного використання.

16

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

Лістинг Б.1 - Ініціалізація TFLiteYoloDetector з GPU

```
init {
    try {
        labels = context.assets.open(LABELS_FILE).bufferedReader().readLines()
        val modelBuffer = loadModelFile(context)

        val (interp, backend) = createInterpreter(modelBuffer)
        interpreter = interp

        val inp = interpreter.getInputTensor(0)
        val out = interpreter.getOutputTensor(0)
        configureOutputLayout(out.shape())
        Log.d(TAG, "✅ Model loaded [$backend] | classes=${labels.size} " +
            "in=${inp.shape().contentToString()} dtype=${inp.dataType()} " +
            "out=${out.shape().contentToString()} dtype=${out.dataType()} | " +
            "layout=${if (outputChannelsFirst) "C×N[1,$outputChannels,$outputAnchors]" else
"N×C[1,$outputAnchors,$outputChannels]"} | " +
            "threshold=${confidenceThreshold}")
    } catch (e: Exception) {
        Log.e(TAG, "❌ Model init failed", e)
        throw RuntimeException("TFLiteYoloDetector init failed: ${e.message}", e)
    }
}

private fun createInterpreter(modelBuffer: java.nio.MappedByteBuffer): Pair<Interpreter, String> {
    try {
        val delegate = GpuDelegate()
        val opts = Interpreter.Options().apply { addDelegate(delegate) }
        val interp = Interpreter(modelBuffer, opts)
        gpuDelegate = delegate
        return Pair(interp, "GPU")
    } catch (t: Throwable) {
        gpuDelegate = null
        Log.w(TAG, "⚠️ GPU delegate unavailable, falling back to CPU: " +
            "${t.javaClass.simpleName} - ${t.message}")
    }
    val opts = Interpreter.Options().apply { setNumThreads(4) }
```

```
    return Pair(Interpreter(modelBuffer, opts), "CPU×4 (GPU unavailable)")
}
```

Лістинг Б.2 - Постпроцесинг: фільтри та NMS

```
val c0 = rawAt(output, 0, i) * coordScale
    val c1 = rawAt(output, 1, i) * coordScale
    val c2 = rawAt(output, 2, i) * coordScale
    val c3 = rawAt(output, 3, i) * coordScale

    val rect = decodeXywhRect(c0, c1, c2, c3, imgW, imgH)
    if (rect == null) { filtBounds++; continue }

    val rectW = rect.width()
    val rectH = rect.height()
    if (rectW < 4f || rectH < 4f) { filtSize++; continue }

    val wNorm    = rectW / imgW
    val hNorm    = rectH / imgH
    val areaNorm = wNorm * hNorm
    if (areaNorm < MIN_BOX_AREA_NORM || areaNorm > MAX_BOX_AREA_NORM) { filtArea++; continue }

    val aspect = rectW / rectH
    if (aspect > MAX_ASPECT_RATIO || aspect < 1f / MAX_ASPECT_RATIO) { filtAspect++; continue }

    val cx = rect.centerX()
    val cy = rect.centerY()
    val edgeMarginX = imgW * EDGE_MARGIN_FRAC
    val edgeMarginY = imgH * EDGE_MARGIN_FRAC
    if (cx < edgeMarginX || cy < edgeMarginY ||
        cx > imgW - edgeMarginX || cy > imgH - edgeMarginY) { filtBounds++; continue }

    if (rectW / imgW > 0.55f || rectH / imgH > 0.60f) { filtFrac++; continue }

    val bx0 = (rect.left  * resizeScaleX).toInt().coerceIn(0, INPUT_SIZE - 1)
    val by0 = (rect.top   * resizeScaleY).toInt().coerceIn(0, INPUT_SIZE - 1)
    val bx1 = (rect.right * resizeScaleX).toInt().coerceIn(0, INPUT_SIZE - 1)
    val by1 = (rect.bottom * resizeScaleY).toInt().coerceIn(0, INPUT_SIZE - 1)
    var brightSum = 0L; var brightCount = 0
    var sy = by0; while (sy <= by1) {
        var sx = bx0; while (sx <= bx1) {
            val px = pixels[sy * INPUT_SIZE + sx]
            brightSum += ((px shr 16) and 0xFF) + ((px shr 8) and 0xFF) + (px and 0xFF)
```

```
            brightCount++
            sx += 4
        }
        sy += 4
    }
    val meanBright = if (brightCount > 0) brightSum / brightCount / 3f else 0f
    if (meanBright < MIN_BOX_BRIGHTNESS) { filtDark++; continue }

    candidates.add(
        Detection(
            classId     = maxClass,
            label       = labels.getOrNull(maxClass) ?: "Unknown",
            confidence  = maxConf,
            boundingBox = rect
        )
    )
}
```

Лістинг Б.3 - SimpleByteTracker: метод update()

```
fun update(detections: List<Detection>): List<Detection> {
    tracks.forEach {
        it.age++
        it.timeSinceUpdate++
    }

    val highDetections = detections.filter { it.confidence >= trackHighThresh }
    val lowDetections = detections.filter {
        it.confidence >= trackLowThresh && it.confidence < trackHighThresh
    }

    val activeTracks = tracks.filter { it.isActive }
    val (matchedTracks, unmatchedTracks, unmatchedHighDets) =
        matchDetectionsToTracks(highDetections, activeTracks)

    matchedTracks.forEach { (track, detection) ->
        track.detection = detection.copy()
        track.hits++
        track.timeSinceUpdate = 0
    }

    val lostTracks = tracks.filter {
        !it.isActive && it.timeSinceUpdate < trackBuffer
```

```
    }
    val remainingDetections = unmatchedHighDets + lowDetections
    val (matchedLost, unmatchedLostTracks, unmatchedRemainDets) =
        matchDetectionsToTracks(remainingDetections, lostTracks)

    matchedLost.forEach { (track, detection) ->
        track.detection = detection.copy()
        track.hits++
        track.timeSinceUpdate = 0
    }

    val matchedInSecondPass = matchedLost.map { it.second }.toSet()
    unmatchedHighDets
        .filter { it !in matchedInSecondPass && it.confidence >= trackHighThresh }
        .forEach { detection ->
            tracks.add(Track(nextId++, detection.copy()))
        }

    tracks.removeAll { it.timeSinceUpdate >= trackBuffer }

    return tracks
        .filter { it.isActive && it.hits >= 1 }
        .map { track ->
            track.detection.copy(trackId = track.id)
        }
}
```

Лістинг Б.4 – DJIVideoStreamManager

```
class DJIVideoStreamManager(
    private val textureView: TextureView
) {
    companion object { private const val TAG = "DJIVideoStream" }

    private var codecManager: DJICodecManager? = null
    @Volatile
    private var frameCallback: ((Bitmap) -> Unit)? = null
    private val isStarted = AtomicBoolean(false)

    private var videoDataListener: VideoFeeder.VideoDataListener? = null

    @Volatile
    private var reusableBitmap: Bitmap? = null
```

```
init {
    textureView.surfaceTextureListener = object : TextureView.SurfaceTextureListener {
        override fun onSurfaceTextureAvailable(surface: SurfaceTexture, width: Int, height: Int) {
            Log.d(TAG, "Surface available: ${width}x${height}")
            codecManager = DJICodecManager(textureView.context, surface, width, height)
        }
        override fun onSurfaceTextureSizeChanged(surface: SurfaceTexture, width: Int, height: Int) {
            codecManager?.onSurfaceSizeChanged(width, height, 0)
        }
        override fun onSurfaceTextureDestroyed(surface: SurfaceTexture): Boolean {
            stop()
            codecManager?.cleanSurface()
            codecManager = null
            return true
        }
        override fun onSurfaceTextureUpdated(surface: SurfaceTexture) {
            val cb = frameCallback ?: return
            val bitmap = reusableBitmap?.let {  textureView.getBitmap(it) }
                ?: textureView.getBitmap()
            if (bitmap != null) {
                reusableBitmap = bitmap
                val copy = bitmap.copy(bitmap.config, false)
                if (copy != null) {
                    cb(copy)
                }
            }
        }
    }
}

fun start() {
    if (!isStarted.compareAndSet(false, true)) return

    val feeder = VideoFeeder.getInstance()
    if (feeder == null) {
        Log.w(TAG, "VideoFeeder is null (no product?)")
        isStarted.set(false)
        return
    }

    val primaryFeed = feeder.primaryVideoFeed
    if (primaryFeed == null) {
        Log.w(TAG, "primaryVideoFeed is null - video stream unavailable for this drone")
```

```
            isStarted.set(false)
            return
        }

        val listener = VideoFeeder.VideoDataListener { videoBuffer, size ->
            codecManager?.sendDataToDecoder(videoBuffer, size)
        }
        videoDataListener = listener
        primaryFeed.addVideoDataListener(listener)

        Log.d(TAG, "Video stream started")
    }
```

Лістинг Б.5 - DJITelemetryManager: GPS callback

```
private fun setupFlightControllerCallback(fc: FlightController) {
    boundFlightController = fc
    fc.setStateCallback { state: FlightControllerState ->
        val lat = state.aircraftLocation?.latitude ?: Double.NaN
        val lon = state.aircraftLocation?.longitude ?: Double.NaN
        val altitude = state.aircraftLocation?.altitude?.toFloat() ?: 0f

        val now = System.currentTimeMillis()
        if (now - lastGpsLogTime > 5000L) {
            lastGpsLogTime = now
            Log.d(TAG, "🌍 GPS callback: lat=$lat lon=$lon alt=$altitude" +
                " sat=${state.satelliteCount} loc=${state.aircraftLocation}")
        }

        isFlying = altitude > 0.5f

        if (!isHomeSet && !isFlying && !lat.isNaN() && !lon.isNaN() && lat != 0.0 && lon != 0.0) {
            homeLatitude = lat
            homeLongitude = lon
            isHomeSet = true
            lastReportedDistance = 0f
            Log.d(TAG, "🏠 Home point set: $homeLatitude, $homeLongitude")
        }

        var distanceFromHome = if (isHomeSet && !lat.isNaN() && !lon.isNaN()) {
            calculateDistance(homeLatitude, homeLongitude, lat, lon)
        } else 0f
```

```
        if (!isFlying && isHomeSet) {
            val distanceChange = kotlin.math.abs(distanceFromHome - lastReportedDistance)
            if (distanceChange < GPS_DRIFT_THRESHOLD) {
                distanceFromHome = lastReportedDistance
            } else {
                lastReportedDistance = distanceFromHome
            }
        } else if (isFlying) {
            lastReportedDistance = distanceFromHome
        }

        val horizontalSpeed = sqrt(
            state.velocityX * state.velocityX + state.velocityY * state.velocityY
        )

        _telemetryFlow.update { current ->
            current.copy(
                latitude = if (lat.isNaN()) 0.0 else lat,
                longitude = if (lon.isNaN()) 0.0 else lon,
                altitude = altitude,
                satelliteCount = state.satelliteCount,
                speed = horizontalSpeed,
                verticalSpeed = -state.velocityZ,
                heading = state.aircraftHeadDirection.toFloat(),
                flightMode = state.flightMode.name,
                distanceFromHome = distanceFromHome,
                timestamp = now
            )
        }
    }
    Log.d(TAG, "✅ FlightController state callback registered on $fc")
}
```

Лістинг Б.6 - DetectionDao та FlightLogDao

```
package com.minedetector.data.local.dao

import androidx.room.*
import com.minedetector.data.local.entities.DetectionEntity

@Dao
interface DetectionDao {
```

```
    @Query("SELECT * FROM detections ORDER BY timestamp DESC")
    suspend fun getAllDetections(): List<DetectionEntity>

    @Query("SELECT * FROM detections WHERE id = :id")
    suspend fun getDetectionById(id: Long): DetectionEntity?

    @Query("SELECT * FROM detections WHERE flightId = :flightId")
    suspend fun getDetectionsByFlightId(flightId: String): List<DetectionEntity>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertDetection(detection: DetectionEntity): Long

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertDetections(detections: List<DetectionEntity>)

    @Delete
    suspend fun deleteDetection(detection: DetectionEntity)

    @Query("DELETE FROM detections")
    suspend fun deleteAllDetections()

    @Query("SELECT COUNT(*) FROM detections")
    suspend fun getDetectionCount(): Int
}


package com.minedetector.data.local.dao

import androidx.room.*
import com.minedetector.data.local.entities.FlightLogEntity

@Dao
interface FlightLogDao {

    @Query("SELECT * FROM flight_logs ORDER BY startTime DESC")
    suspend fun getAllFlightLogs(): List<FlightLogEntity>

    @Query("SELECT * FROM flight_logs WHERE id = :id")
    suspend fun getFlightLogById(id: Long): FlightLogEntity?

    @Query("SELECT * FROM flight_logs WHERE flightId = :flightId")
    suspend fun getFlightLogByFlightId(flightId: String): FlightLogEntity?
```

```
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertFlightLog(flightLog: FlightLogEntity): Long

    @Update
    suspend fun updateFlightLog(flightLog: FlightLogEntity)

    @Delete
    suspend fun deleteFlightLog(flightLog: FlightLogEntity)

    @Query("DELETE FROM flight_logs")
    suspend fun deleteAllFlightLogs()
}
```

Лістинг Б.7 - VideoOverlayView: onDraw()

```
override fun onDraw(canvas: Canvas) {
    super.onDraw(canvas)

    canvas.save()
    canvas.clipRect(0f, 0f, width.toFloat(), height.toFloat())

    for (detection in detections) {
        drawDetection(canvas, detection)
    }

    if (showSignalLost) {
        drawSignalLostOverlay(canvas)
    }

    canvas.restore()
}

private fun drawDetection(canvas: Canvas, detection: Detection) {
    val color = colorMap[detection.classId] ?: Color.RED
    boxPaint.color = color
    textBackgroundPaint.color = color

    val viewRect = toViewRect(detection.boundingBox)

    if (viewRect.right < 0f || viewRect.bottom < 0f ||
        viewRect.left > width.toFloat() || viewRect.top > height.toFloat()) {
        return
    }
```

```
    canvas.drawRect(viewRect, boxPaint)
    drawLabel(canvas, detection, viewRect, color)
    drawCenterPoint(canvas, viewRect, color)
}

private fun toViewRect(imageRect: RectF): RectF {
    val pm = photoViewMatrix
    if (pm != null && scale > 0f) {
        val mv = FloatArray(9)
        pm.getValues(mv)
        val matrixScale = mv[Matrix.MSCALE_X]
        if (abs(matrixScale - scale) > 0.02f) {
            val r = RectF(imageRect)
            pm.mapRect(r)
            return r
        }
    }
    return scaleRect(imageRect)
}
```

Лістинг Б.8 – MineDetectorApplication

```
class MineDetectorApplication : Application() {

    companion object {
        private const val TAG = "MineDetectorApp"

        const val TEST_MODE = false

        @Volatile
        lateinit var instance: MineDetectorApplication
            private set
    }

    override fun attachBaseContext(base: Context) {
        super.attachBaseContext(base)
        instance = this
        MultiDex.install(this)

        if (!TEST_MODE) {
            try {
                Class.forName("com.secneo.sdk.Helper")
```

```
                .getMethod("install", Application::class.java)
                .invoke(null, this)
            Log.d(TAG, "DJI Helper installed successfully")
        } catch (e: ClassNotFoundException) {
            Log.w(TAG, "DJI Helper class not found - running without DJI SDK")
        } catch (e: Exception) {
            Log.e(TAG, "Failed to install DJI Helper: ${e.message}")
        }
    } else {
        Log.d(TAG, "Application context attached (DJI Helper disabled for testing)")
    }
}

override fun onCreate() {
    super.onCreate()
    Log.d(TAG, "Application created")
    installCrashLogger()

    try {
        if (BuildConfig.MAPBOX_ACCESS_TOKEN.isNotEmpty()) {
            com.mapbox.common.MapboxOptions.accessToken = BuildConfig.MAPBOX_ACCESS_TOKEN
            Log.d(TAG, "Mapbox initialized with access token")
        } else {
            Log.w(TAG, "Mapbox token not configured in local.properties")
        }
    } catch (e: Exception) {
        Log.e(TAG, "Failed to initialize Mapbox: ${e.message}")
    }
}
```