Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

КОМП'ЮТЕРНА СИСТЕМА МОНІТОРИНГУ ТА ОЦІНКИ БЕЗПЕКИ РУХУ ВЕЛОСИПЕДИСТІВ ІЗ ЗАСТОСУВАННЯМ НЕЙРОННИХ МЕРЕЖ

Виконав: студент 2 курсу, групи 2КІ-24м
напряму підготовки (спеціальності)
123 — «Комп'ютерна інженерія»
_____ Циганков О.А.

Керівник роботи: к.т.н., доц. каф. ОТ
_____ Городецька О.С.
«16» 12 2025 р.
Опонент: д.ф., доц. каф. МБІС
_____ Салієва О.В.
«18» 12 2025 р.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.

«19» 12 2025 р.

ВНТУ – 2025

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань 12 — Інформаційні технології
Спеціальність 123 — «Комп'ютерна інженерія»
Освітня програма — «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри обчислювальної техніки
проф., д.т.н. О.Д. Азаров
«25» вересня 2025 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

студенту Циганкову Олексію Андрійовичу

1 Тема роботи «Комп'ютерна система моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж», керівник роботи Городецька Оксана Степанівна, к.т.н., доцент, затверджені наказом вищого навчального закладу від 24 вересня 2025р. № 313

2 Строк подання студентом роботи 4 грудня 2025 року

3 Вихідні дані до роботи: комп'ютерний зір як сфера штучного інтелекту, що базується на методах глибокого навчання та забезпечує виявлення, розпізнавання й класифікацію об'єктів на зображеннях і відео вуличного трафіку; методи розпізнавання, засновані на нейронних мережах; архітектурне рішення нейронної мережі – згорткова нейронна мережа.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): вступ, аналіз предметної області та сучасних методів комп'ютерного зору, вибір архітектури нейронної мережі, вдосконалення методу розпізнавання об'єктів на зображенні, навчання моделі, розробка та реалізація комп'ютерної системи моніторингу та оцінки безпеки руху велосипедистів, програмна реалізація та тестування, економічна частина.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): структура нейронної мережі, блок-схема алгоритму донавчання

нейронної мережі, матриця помилок моделі YOLOv8, архітектура комп'юте
системи моніторингу та оцінки безпеки руху велосипедистів, структурна с
комп'ютерної системи моніторингу та оцінки безпеки руху велосипедистів.

6 Консультанти розділів роботи представлені в таблиці1.

Таблиця 1— Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Городецька О.С., к.т.н., доц. каф. ОТ		
5	Адлер О.О. к.т.н., доц. доцент кафедри ЕПВМ		
Нормоконтроль	Швець С. І. ас. каф. ОТ		

7 Дата видачі завдання 25.09.2025 р.

8 Календарний план виконання МКР наведено в таблиці 2.

Таблиця 2—Календарний план

№	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Прим
1	Постановка задачі	25.09.2025	викон
2	Огляд існуючих рішень	28.09.2025-4.10.2025	викон
3	Розробка структурної схеми	5.10.2025-15.10.2025	викон
4	Підготовка та навчання нейронної мережі	16.10.2025-1.11.2025	викон
5	Моделювання роботи системи	2.11.2025-26.11.2025	викон
6	Виконання магістерської кваліфікаційної роботи	27.11.2025-14.12.2025	викон
7	Перевірка «антиплагіат»	15.12.2025	викон
8	Підготовка і підпис супроводжуючих документів	18.12.2025	викон

Студент _____ Циганков О.А.

Керівник роботи _____ Городецька О. С.

АНОТАЦІЯ

УДК 004.42

Циганков О. А. Комп’ютерна система моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж. Магістерська кваліфікаційна робота зі спеціальності 123 — Комп’ютерна інженерія, Вінниця: ВНТУ, 2025. — 121 с. На укр.мові. Бібліогр.: 25 назв; рис.:34 ; табл.11.

У магістерській кваліфікаційній роботі розроблено комп’ютерну систему моніторингу та оцінки безпеки руху велосипедистів із застосуванням методів комп’ютерного зору та глибинного навчання. Система забезпечує отримання відеопотоку (вебкамера/RTSP/відеофайл), автоматичне виявлення велосипедистів і визначення наявності/відсутності захисного шолома, а також формування статистики порушень за заданими інтервалами часу. Реалізацію виконано мовою Python із використанням бібліотек Ultralytics (YOLOv8), PyTorch та OpenCV; передбачено логування подій із часовими мітками, класами та значеннями впевненості з можливістю подальшого експорту результатів для аналізу. Проведено тестування працездатності на відео різної складності та визначено напрями подальшого розвитку системи.

Ключові слова: комп’ютерна система моніторингу, комп’ютерний зір, безпека руху, нейронна мережа, глибинне навчання.

ABSTRACT

UDC 004.42

Tsygankov O. A. Computer system for monitoring and assessing the safety of cyclists using neural networks. Master's qualification work in specialty 123 - Computer Engineering, Vinnytsia: VNTU, 2025. - 121 p. In Ukrainian. Bibliography: 25 titles; Fig.: 34 ; Table 11.

In the master's qualification work, a computer system for monitoring and assessing the safety of cyclists using computer vision and deep learning methods has been developed. The system provides video stream reception (webcam/RTSP/video file), automatic detection of cyclists and determination of the presence/absence of a protective helmet, as well as the formation of statistics of violations at specified time intervals. The implementation was performed in Python using the Ultralytics (YOLOv8), PyTorch and OpenCV libraries; Event logging with timestamps, classes and confidence values is provided with the possibility of further exporting the results for analysis. Performance testing was carried out on videos of varying complexity and directions for further development of the system were identified.

Keywords: computer monitoring system, computer vision, traffic safety, neural network, deep learning.

ЗМІСТ

ВСТУП

У сучасних містах кількість велосипедистів невпинно зростає, проте рівень їхньої безпеки істотно залежить від дотримання правил дорожнього руху та використання індивідуальних засобів захисту, насамперед шолома. Попри доведену ефективність шоломів у зниженні ризику черепно-мозкових травм, практика демонструє значну частку поїздок без захисту голови, особливо у змішаному транспортному потоці, під час коротких переміщень «поруч із домом» та в умовах обмеженої видимості. Така поведінка підвищує імовірність тяжких наслідків ДТП і ускладнює превентивні заходи, засновані лише на інформаційних кампаніях.

Це зумовлює потребу у впровадженні автоматизованих технічних рішень, які в реальному часі здатні виявляти велосипедистів у відеопотоці та визначати факт наявності/відсутності шолома, формуючи об’єктивні кількісні індикатори ризику. Сучасні методи комп’ютерного зору та глибинного навчання забезпечують необхідний баланс між точністю та швидкодією для роботи на споживчих GPU і дають змогу масштабувати моніторинг з окремих камер до міських підсистем відеоспостереження. Водночас важливо врахувати доменні фактори (ракурси, освітлення, оклюзії, наявність капюшонів/кепок), які впливають на якість розпізнавання.

Актуальність теми зумовлена зростанням кількості велосипедистів у міському трафіку та високою часткою поїздок без шолома, що істотно підвищує ризик тяжких травм. В умовах розвитку дорожньої інфраструктури для велосипедистів з’являється потреба в безпеці. Автоматизований моніторинг із застосуванням глибинного навчання дозволяє оперативно виявляти порушення.

Метою роботи є збільшення точності розпізнавання об’єктів на зображенні під час відеозйомки дорожнього руху із застосуванням нейронних мереж.

Основні задачі:

— провести аналіз предметної області та оглянути сучасні методи детекції та класифікації;

— вдосконалити метод розпізнавання об’єктів на зображенні під час дорожнього руху;

— підготувати та розмітити дані: розширити датасет, уніфікувати анотації у формат YOLO, виконати train/val спліт і аугментації;

— розробити та навчити модель детекції “шолом/без шолома” із підбором гіперпараметрів та контролем якості;

— розробити структурну схему комп’ютерної системи моніторингу відеопотоку в реальному часі;

— здійснити програмну реалізацію модуля детекції;

— виконати перевірку працездатності та тестування системи.

Об’єкт дослідження — процес розпізнавання об’єктів на зображенні під час відеозйомки велосипедного руху.

Предметом дослідження є методи та засоби комп’ютерного зору для автоматичного виявлення велосипедистів і визначення наявності шолома у відеопотоці в реальному часі.

Методи дослідження: аналіз літератури; підготовка та аугментація даних; трансферне навчання і тренування детектора YOLOv8 у PyTorch; метричне оцінювання (Precision/Recall, mAP, PR/F1-криві) на валідаційних піднаборах; порівняльний експериментальний аналіз альтернативних архітектур.

Наукова новизна: вдосконалено метод розпізнавання об’єктів на зображенні під час відеозйомки дорожнього руху шляхом використання одноетапного детектора, який відрізняється від існуючих прямим одноетапним визначенням класів і координат рамок без проміжного етапу генерації регіонів, що дало можливість збільшити точність розпізнавання.

Практична цінність дослідження — розроблена система сприятиме підвищенню рівня безпеки на вулицях для пішоходів, велосипедистів та водіїв автомобілів.

Апробацію результатів магістерської кваліфікаційної роботи — зроблено доповідь на Міжнародній науково-практичній інтернет-конференції Молодь в науці: дослідження, проблеми, перспективи (МН-2026) [22].

Матеріали роботи доповідались та опубліковувались:

Циганков О.А. Комп’ютерна система моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж / О.А. Циганков, С.В. Городецька // Матеріали МНПІК Молодь в науці: дослідження, проблеми, перспективи (МН-2026), 2026 р. Режим доступу: — https://conferences.vntu.edu.ua/index.php/mn/mn2026/paper/viewFile/26536/21886.

1 ОГЛЯД ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ШЛЯХІВ ВИРІШЕННЯ ЗАДАЧІ

Сучасні міста стають дедалі більш велосипедно орієнтованими, однак рівень безпеки велосипедистів істотно залежить від дотримання правил і використання захисних засобів, насамперед шолома. За даними профільних досліджень, носіння шолома знижує ризик тяжкої травми голови, а отже є найпростішим і найефективнішим превентивним заходом. Водночас у реальних умовах відсоток велосипедистів, що їздять без шолома, залишається помітним, особливо у подвір'ях, на малих вулицях і під час коротких поїздок. Це формує практичний запит на автоматизовані системи моніторингу, здатні у реальному часі виявляти велосипедистів і визначати факт наявності чи відсутності шолома, надаючи кількісну оцінку рівня безпеки трафіку.

1.1 Аналіз предметної області та постановка задачі

1.1.1 Сценарії ризиків для велосипедистів

У межах предметної області виділимо типові сценарії ризику:

— рух у змішаному потоці з авто, коли велосипедист може потрапити в «сліпі зони» транспортних засобів;

— перетин перехресть і пішохідних переходів, де помилки пріоритету спричиняють зіткнення;

— поїздки в темний час доби або за складних погодних умов (дощ, туман), коли видимість обмежена;

— маневри на високій швидкості при об’їзді перешкод чи ям.

У кожному з цих сценаріїв відсутність шолома значно підвищує тяжкість наслідків. Визначення просторово-часової частки «без шолома» у відеопотоці з камер міської інфраструктури, навчальних кампусів або корпоративних стоянок дозволяє формувати доказову базу для інтервенцій: інформаційних кампаній, локальних змін організації руху, додаткового освітлення чи попереджувальних табло. Схему сценаріїв ризику зображено на рис. 1.1.

Рисунок 1.1 — Ризики для велосипедиста

1.1.2. Цілі моніторингу та критерії успіху системи

Для цієї задачі потрібно розробити комп'ютерну систему моніторингу та оцінки безпеки руху велосипедистів із застосуванням глибинного навчання, що виконує виявлення велосипедистів/областей голови у кадрі та класифікацію «з шоломом / без шолома», агрегуючи результати в аналітичні метрики. Очікувані вхідні дані — відеопотік або окремі зображення зі звичайних камер спостереження (Full HD або нижче). Обмеження: змінне освітлення, ракурси, часткові оклюзії, різна якість лінз, наявність шапок/капюшонів, що можуть маскувати шолом.

Цілі та критерії успіху формулюємо так:

— реальний час або наближений до нього (≥15 FPS на споживчому GPU);

— стабільне виявлення велосипедистів у типових міських сценах;

— коректна класифікація шолома на різних відстанях.

Якісні критерії: цільові значення точності/повноти та інтегральної метрики mAP (наприклад, mAP@50 ≥ 0,75 на валідації), прийнятна частка хибних

спрацювань на класі «без шолома». Аналітичні критерії: розрахунок ковзної частки велосипедистів без шолома за вибраний інтервал (наприклад, 10 секунд – 60 секунд), формування логів/дашбордів для порівняння локацій і часових періодів. Нефункціональні вимоги: відтворюваність навчання (фіксація гіперпараметрів і підсівів), портативність моделі (експорт у ONNX), прозорий процес підготовки даних і вимоги до апаратного забезпечення. Така формалізація напряму пов’язує алгоритмічні результати з прикладною метою — зниженням травматизму за рахунок моніторингу і своєчасних управлінських рішень.

1.2 Огляд сучасних методів комп’ютерного зору для виявлення та класифікації

Східним пунктом сучасних підходів є глибинні згорткові мережі (CNN) та трансформери. CNN ResNet [14], EfficientNet [6], MobileNet, DenseNet автоматично навчаються багаторівневим ознакам — від контурів до семантичних патернів — завдяки згорткам і підвибірці. Трансформери (ViT, Swin) оперують механізмом уваги: замість локальних фільтрів вони моделюють довгі залежності між патчами зображення, що покращує узагальнюваність, але зазвичай потребує більше даних і обчислень. На практиці обидва сімейства часто поєднують: наприклад, CNN як «стем» для вилучення ознак і трансформерні блоки для глобальної агрегації.

Одноетапні детектори (YOLO, SSD [12], RetinaNet, EfficientDet [13]) напряму прогнозують класи та координати рамок у сітці, забезпечуючи високу швидкодію для застосувань реального часу. Двоетапні підходи (Faster R-CNN) спершу виділяють регіони-кандидати, а далі уточнюють класи/координати, що часто дає вищу точність на складних сценах, але ціною латентності. Окрему гілку становлять трансформерні детектори (DETR і похідні), які відмовляються від «якорів» та нефункціональної NMS, формулюючи пошук об’єктів як задачу зіставлення запитів; вони спрощують пайплайн, проте вимагають ретельного тренування та більше ресурсів. Для задачі розпізнавання шолома у потоковому відео доцільним компромісом є легкі одноетапні моделі, що зберігають прийнятну точність на споживчому GPU та підтримують простий деплой (експорт у ONNX).

Практичний успіх визначається не лише архітектурою, а й повним циклом ML-інженерії: добором і підготовкою даних (якість розмітки, баланс класів), доменними аугментаціями [15] (варіації освітлення, погоди, руховий блюр, ракурси, часткові оклюзії), а також протоколом оцінювання (mAP@50/50–95, PR/F1, матриця помилок). Трансферне навчання з попередньо натренованих ваг (ImageNet/COCO [7]) пришвидшує збіжність і підвищує узагальнюваність за дефіциту даних. Для відеопотоків додатково важливі трекінг об’єктів (щоб рахуати саме велосипедистів, а не кадри) та вибір робочого порогу впевненості, який мінімізує пропуски класу «без шолома» при контрольованій кількості хибних спрацювань. Сукупність цих рішень забезпечує збалансований результат між точністю, швидкістю та стабільністю в реальних міських сценах.

1.2.1. Згорткові нейронні мережі CNN і трансформери: принципи та еволюція

Для виявлення об’єктів сформувалися дві парадигми. Двоетапні детектори (Faster R-CNN) спершу генерують регіони-кандидати, а потім класифікують їх; вони стабільно точні, але повільніші. Одноетапні (SSD, YOLO, RetinaNet, EfficientDet) напряму регресують рамки та класи в сітці, забезпечуючи високу швидкість при дещо нижчій точності на складних сценах. RetinaNet запровадила фокальну втрату для боротьби з дисбалансом «фон/об’єкт», а EfficientDet поєднала легкий бекбон EfficientNet з багатомасштабною «BiFPN»-пірамідою ознак, даючи гарне співвідношення «точність/ресурси». Лінійка YOLO (з v3 до v8) еволюціонувала до якісних «one-stage» детекторів із сильними аугментаціями та зручним деплоєм у реальному часі. Окрема лінія — DETR (Detection Transformer): він формулює детекцію як завдання прямого зіставлення запитів об’єктів і прогнозів, відмовляючись від «якорів» і NMS; це спрощує пайплайн, але потребує ретельного тренування й ресурсів.

Згорткові нейромережі (CNN) базуються на локальних фільтрах, які «ковзають» по зображенню та виділяють ознаки різного рівня: від простих контурів і текстур до складних семантичних патернів. Ключові механізми — згортки,

підвибірка (pooling), нормалізація (batch/group norm), залишкові з’єднання (ResNet), щільні зв’язки (DenseNet), а також легковагові операції, як-от глибинно-роздільні згортки (MobileNet). Удосконалення типу SE/CBAM-додатків уваги до каналів/простору дозволяють мережам зосереджувати ресурси на «інформативних» регіонах, підвищуючи якість при помірних витратах FLOPs і параметрів. Схема процесу розпізнавання зображено на рис. 1.2.

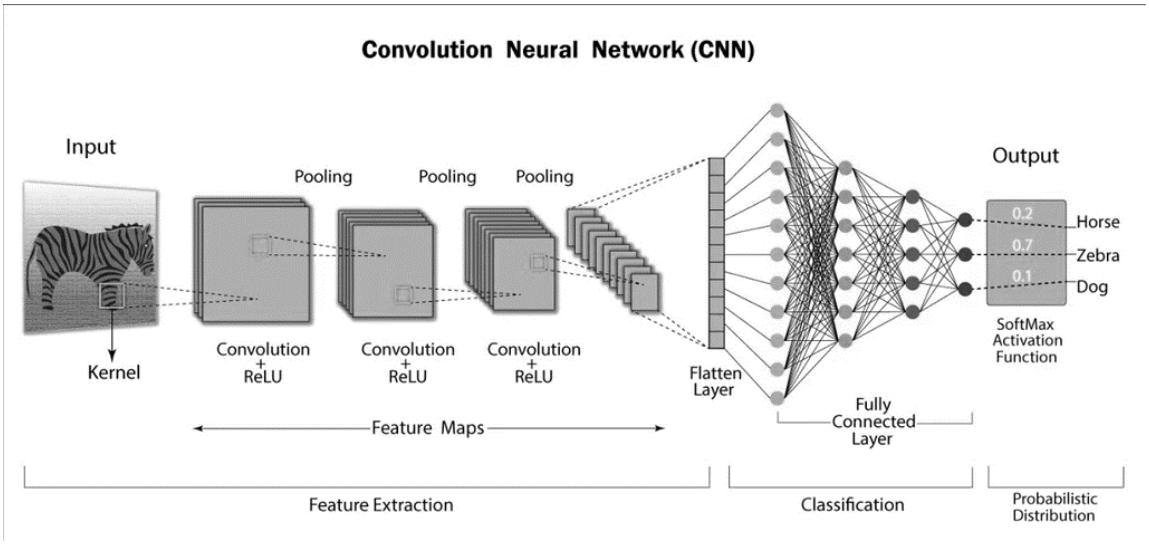


Рисунок 1.2 — Узагальнена схема процесу розпізнавання зображення згортковою нейронною мережею

Трансформери у комп’ютерному зорі (ViT, Swin) переносять механізм самоуваги із мовних моделей на зображення: картинка розбивається на патчі-токени, а самоувага моделює довгі залежності між будь-якими регіонами кадру. Це сприяє кращій узагальнюваності у різних ракурсах і сценах, але потребує більших даних і ретельних режимів навчання. Ієрархічні варіанти (Swin) вводять «ковзні вікна» та багаторівневу піраміду ознак, що наближає їх за структурою до CNN. Паралельно з’явилися гібриди (ConvNeXt), які запозичують дизайн-рішення трансформерів (великі ядра, спрощені блоки), зберігаючи ефективність згорток.

Еволюція і CNN, і трансформерів відбувалася разом із практичними «трик-пакетами»: попереднє навчання (ImageNet/COCO), трансфер, масштабування мереж за єдиними правилами (EfficientNet compound scaling), багатомасштабні піраміди ознак (FPN/BiFPN/PAFPN), сучасні функції втрат (GIoU/CIoU/Focal), а

також сильні аугментації (MixUp, CutMix, Mosaic). Для розгортання у продакшені важливі нормалізація/калібрування порогів, квантизація, дистиляція знань і експорт у ONNX/TensorRT [19]. Саме сукупність архітектурних і інженерних рішень забезпечує баланс точності, швидкодії та стабільності, необхідний для задач моніторингу шоломів у реальному часі.

1.2.2. Порівняння детекторів YOLO, SSD, EfficientDet, DETR і класифікаторів ResNet, EfficientNet, ViT

Для класифікації базовими є ResNet (залишкові зв'язки стабілізують глибоке навчання) і його нащадки. EfficientNet масштабує глибину/ширину/роздільність за єдиним правилом і забезпечує високу якість при малих FLOPs — корисно для вбудованих систем. MobileNetV2/V3 орієнтовані на мобільні пристрої (групові/глибинні згортки). ConvNeXt модернізує CNN у «духу» трансформерів (великі ядра, упрощені блоки), наближаючись до ViT за точністю. ViT безпосередньо працює з патчами як із токенами; на середніх наборах даних зазвичай вимагає предтренування або агресивних аугментацій.

Ключові компоненти тренування: трансферне навчання (ініціалізація ваг із ImageNet/COCO), аугментації (мозаїка, mixup, колірні зсуви, масштабування), баланс класів, коректний спліт і метрики (Precision, Recall, mAP@50 та mAP@50–95). Для відео важливі стабільність кадрів, FPS і латентність інференсу. Обираючи архітектуру, орієнтуються на компроміс: якщо потрібен реальний час на споживчому GPU — практичною є лінійка YOLO або EfficientDet-D0/D1; якщо пріоритет — максимальна точність офлайн, розглядають Faster R-CNN/RetinaNet/DETR або їх сучасні варіанти з ConvNeXt/ViT-бекбонами. Для задач на кшталт «шолом/без шолома» доцільно поєднати легкий детектор з невибагливим класифікатором (EfficientNet-B0/MobileNet), що дає баланс точності, швидкості й простоти розгортання.

Також, при порівнянні детекторів, варто виділити ресурсні профілі та масштабованість. Лінійки YOLO (n/s/m/l/x) і EfficientDet (D0–D7) пропонують «сімейства» моделей із передбачуваним зростанням FLOPs/VRAM разом із

точністю: на споживчому GPU типу GTX 1650 оптимальні yolov8n/s або EfficientDet-D0/D1, які утримують з 15 до 30 FPS у Full HD зі зниженим imgsz. SSD з легким бекбоном (MobileNet) ще швидший, але поступається в AP для дрібних/частково закритих об'єктів (голова під капюшоном). DETR і варіанти з attention-головами краще працюють на складних сценах із накладаннями, проте вимагають довшого навчання та сильнішого апаратного забезпечення; їх доцільно розглядати для офлайн-обробки або коли потрібна єдність пайплайна з трансформерною екосистемою. Графік порівняння точності зображено на рис. 1.3.

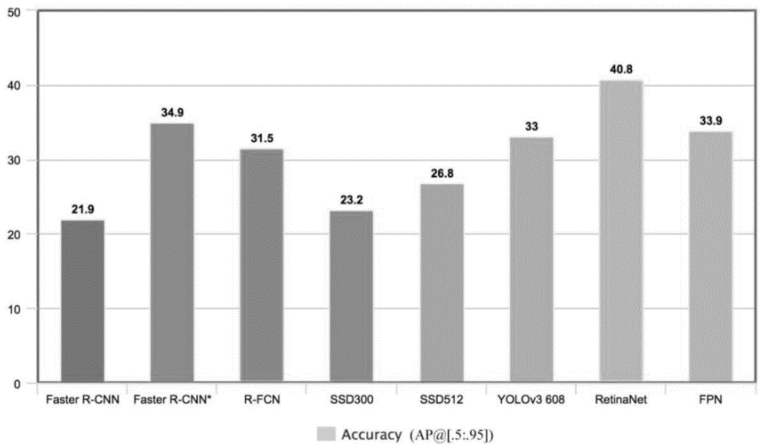


Рисунок 1.3 Порівняння точності класичних детекторів об'єктів [20]

Для класифікаторів ключовий компроміс — «точність ↔ latency ↔ параметри». ResNet-50/101 — надійна «класика» з хорошою узагальнюваністю, але не завжди оптимальна за затримкою. EfficientNet-B0/B1 дає відмінне співвідношення якості до FLOPs і підходить як універсальний бекбон на слабших GPU/CPU. MobileNetV3 — вибір для edge-сценаріїв і вбудованих систем (мінімальні ресурси, прийнятна якість). ViT-Base/16 доречний, якщо є достатньо даних/аугментацій або готові чекпойнти з великих датасетів; зазвичай його беруть, коли потрібна узгодженість зі стеком трансформерів та досліджується перенос знань між задачами. ConvNeXt вирівнює відставання CNN від ViT, але в простих двокласових постановках (шолом/без) надмірна ємність рідко окупається.

З погляду практичної інтеграції, важливі підтримка експорту (ONNX/TensorRT), стабільність інференсу та інструменти. Ultralytics YOLOv8 має

простий CLI/API, а також готові процедури тренування, валідації й експорт у ONNX — це знижує інженерні витрати [1]. EfficientDet у TensorFlow/TF-Lite зручний, якщо ціль — мобільний/вбудований деплой. Для офлайн-аналітики або на сервері з кращим GPU можна випробувати DETR/RetinaNet + ConvNeXt/ViT-бекбон і порівняти AP на «важких» піднаборах (ніч, дощ, щільні сцени). З урахуванням задачі «шолом/без шолома» на міських велодоріжках, типовим «базовим» вибором є YOLOv8n/s як детектор і EfficientNet-B0/MobileNetV3 як бекбон/класіфікатор у разі дворівневого підходу — вони забезпечують потрібний баланс якості, швидкості та простоти розгортання на доступному обладнанні.

Приклад практичної реалізації процесу навчання,конфігурація запуску та основні параметри тренування YOLOv8, наведено в Додатку В [23].

1.3. Підготовка даних та методи оцінювання результатів розпізнавання

1.3.1. Джерела даних, формати розмітки, якість анотацій

Для навчання системи використано відкритий набір «Helmet Detection» Kaggle [4] та додаткові зображення, зібрані з відкритих джерел/власних фото для підсилення різноманітності сцен. Вихідні анотації подані у форматі PASCAL VOC [8](XML) і були конвертовані у формат YOLO (TXT) з нормованими координатами рамок; така уніфікація спрощує тренування детектора. Класи: With Helmet та Without Helmet; на етапі підготовки перевірено відповідність імен файлів, присутність розмірів у XML, а також коректність рамок (xmin<xmax, ymin<ymax). Для уникнення витоку даних у валідацію виконано розділення на train/val у пропорції 80/20 зі збереженням розподілу класів (stratified split), зафіксовано генератор випадкових чисел для відтворюваності. За потреби виділяється test-піднабір для фінальної, «замороженої» оцінки.

Якість даних безпосередньо впливає на узагальнюваність: у підготовчому етапі передбачена візуальна ревізія анотацій, видалення дублікатів і «битих» зображень, а також балансування класів (додавання знімків «без шолома», де це доречно). Для підвищення стійкості моделі застосовано аугментації: випадкові повороти/зсуви/масштабування, зміни яскравості/контрасту, «мозаїка» й випадкові

обрізки — у допустимих межах, щоб не зіпсувати семантику об’єкта. Окремо враховано «важкі» умови (оклюзії, неповні рамки голови, капюшони/кепки, низьке освітлення), які додано до тренувального набору для зменшення зсуву домену. Метадані (розміри, кількість рамок, частка класів) зберігаються у звіті підготовки, що дозволяє контролювати зміни вибірки в часі.

Формати розмітки відрізняються структурою та системою координат: PASCAL VOC зберігає для кожного зображення окремий XML із піксельними координатами рамок (xmin, ymin, xmax, ymax); COCO — це єдиний JSON-каталог із переліком зображень, анотацій та категорій, де рамки подаються як (x, y, width, height) у пікселях і можуть доповнюватися полігональними масками сегментації; YOLO — це окремий TXT на зображення з рядками формату class_id cx cy w h, де координати нормовані в інтервалі [0,1] і подані як центр та розміри рамки. Під час конвертації важливо дотримуватись відповідності між порядком класів у data.yaml та class_id у TXT, правильно інтерпретувати систему координат (центр vs кути, нормовані vs піксельні), а також перевірити відсутність від’ємних або вихідних за межі зображення значень.

Якість анотацій контролюється поєднанням автоматизованих і візуальних перевірок. До автоматизованих належать скрипти перевірки порожніх або дублікатних лейблів, відсів рамок «аномальних» розмірів/пропорцій, детекція виходу за межі кадру, оцінка мінімального покриття об’єкта. Візуальні перевірки включають перегляд підмножини зображень у в’юверах розмітки (random/sample-by-case), контроль консистентності класів (щоб «капюшон» не маркували як «шолом») і щільності рамок (не надто «вільні»/«затісні»). Додатково варто відслідковувати статистики вибірки: баланс класів, розподіл розмірів об’єкта (малі/середні/великі), кількість рамок на кадр, частку «важких» умов (ніч, дощ, оклюзії) — ці метрики прямо впливають на узагальнюваність моделі.

Стратегія поділу даних має мінімізувати витік сцен між train/val. Для відео доцільний трек- або кліп-орієнтований спліт: усі кадри одного треку/сцени залишаються лише в одному піднаборі. Для статичних зображень корисно робити стратифікацію за камерами/локаціями/часом (щоб одна й та сама ділянка

велодоріжки не з’являлась у різних піднаборах), а також усувати майже дублікати (frame subsampling). Рекомендовано версіонувати датасет (маніфест із хешами, логами конвертації), зберігати скрипти перетворення VOC до YOLO та конфігурації data.yaml разом із набором — це забезпечує відтворюваність експериментів і прозорий аудит підготовки даних.

1.3.2. Метрики Precision, Recall, mAP, F1, валідація, баланс класів

Оцінювання проводиться на відкладеній val-вибірці за стандартними метриками детекції: mAP@50 (середня точність при IoU=0.5) і mAP@50–95 (середнє по порогах IoU=0.5…0.95 з кроком 0.05). Додатково аналізуються Precision, Recall, F1-score та AP по кожному класу. Для інтерпретації компромісу чутливість/специфічність будуються PR-криві; оптимальний робочий поріг conf підбирається на максимумі F1 або за вимогами застосування (наприклад, мінімізувати пропуск «без шолома»). Confusion matrix використовується для виявлення домінуючих помилок (плутання класів, помилкові спрацьовування на аксесуари — кепки, навушники). Значення IoU застосовується як критерій збігу прогнозованої рамки з еталоном; для малих об’єктів розглядається зниження порогу IoU або багатомасштабна інференція.

Щоб уникнути перенавчання, використовується early stopping за валідаційною втратою/мAP, логуються криві навчання, зберігаються «найкращі» ваги за mAP. Для стабільності результатів фіксуються seed-и, версії бібліотек і гіперпараметри (epochs, batch, imgsz, аугментації). За наявності часу проводиться k-fold cross-validation на підмножині даних або повторні запуски з різними підсівами для оцінки варіабельності. Для прикладної мети (моніторинг безпеки) вводиться агрегована метрика — ковзна частка детекцій класу “Without Helmet” за інтервал (від 10 секунд до 60 секунд) у відео; вона слугує індикатором ризику локації/періоду та використовується разом із технічними метриками (mAP/PR/F1) для комплексної оцінки системи.

Додатково контролюється баланс класів і коректність порогів. За дисбалансу (With Helmet ≫ Without Helmet) корисно звітувати не лише усереднені

mAP, а й клас-специфічні Precision/Recall/F1, PR-AUC, а також зважені (weighted) метрики. Робочий поріг conf підбирають окремо для класу Without Helmet з пріоритетом чутливості (щоб мінімізувати пропуски), після чого перевіряють вплив на precision. Для підвищення стабільності оцінювання застосовують темпоральну валідацію (кадри/сцени з інших днів або камер), а також перерозмітку вибіркових помилок: повторна перевірка еталонів на фрагментах із найбільшим внеском у хибні спрацювання. Корисно окремо звітувати метрики для малих/середніх об’єктів (голів), оскільки саме на малих розмірах моделі найчастіше втрачають AP.

Для прикладного моніторингу у звітах, крім технічних метрик (mAP/PR/F1), подаються агреговані показники за періодами: кількість і частка фіксацій з шоломом / без шолома за треками (а не по кадрах) для кожної камери, з розбивкою по годинах/добах та локаціях. Такий підхід зменшує дублювання підрахунків одного велосипедиста на послідовних кадрах і дає інтерпретовані для експлуатації числа. Для забезпечення порівнюваності між моделями/версіями фіксуються однакові протоколи валідації, однакові піднабори val/test, однакові процедури підбору порогів і однакові звітні таблиці (включно з confusion matrix і переліком типових помилок) — це дозволяє коректно оцінити прогрес і вплив змін у даних чи гіперпараметрах.

1.4 Аналіз існуючих систем моніторингу використання шоломів та відеоконтролю велосипедистів

На сьогодні запропоновано низку наукових робіт і промислових рішень, що використовують відеоаналітику для контролю засобів індивідуального захисту, зокрема захисних шоломів. Найбільш поширеними є системи, інтегровані з існуючими підсистемами відеоспостереження: відеопотік із IP-камер або відеореєстраторів подається на сервер обробки, де детектор об’єктів виділяє людей та зону голови, після чого класифікатор визначає наявність шолома. Такі системи орієнтовані на роботу в реальному часі, формують сповіщення про порушення та протоколи подій, які далі можуть використовуватися службами безпеки або

адміністрацією об’єкта.

Окрему групу становлять рішення, спрямовані на моніторинг використання касок на промислових майданчиках і будівельних об’єктах. У більш ранніх роботах застосовувалися каскадні класифікатори та правила, що ґрунтуються на кольорі шолома, однак вони погано працювали за змінного освітлення і при часткових оклюзіях. Сучасні системи використовують згорткові нейронні мережі та one-stage-детектори (YOLO, SSD, EfficientDet), що дає змогу виявляти каску як окремий клас об’єктів без попереднього виділення області голови. Такі рішення, як правило, орієнтовані на відносно стабільне положення камери, контрольоване тло й невеликі швидкості руху, а отримані події використовуються переважно для локального контролю дотримання техніки безпеки. Приклад такої системи на базі RTSP і OpenVINO зображено на рис. 1.4

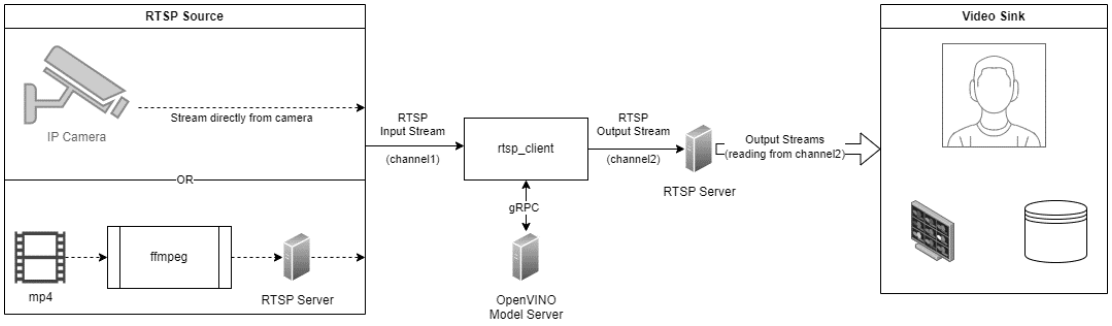


Рисунок 1.4 — Приклад архітектури системи аналізу відеопотоку з використанням RTSP-джерел та серверу моделей OpenVINO

Близькими до задачі цієї роботи є системи контролю шоломів у мототранспорті. Для виявлення мотоциклістів без шолома застосовуються як дворівневі підходи (детекція мотоцикліста з подальшою класифікацією області голови), так і безпосередня детекція класів «rider with helmet» / «rider without helmet» у загальному потоці транспорту. У низці рішень використовуються легковагові варіанти YOLOv3–v7[10,11] для розгортання на вбудованих пристроях біля перехресть, а результати пов’язуються з розпізнаванням номерних знаків для подальшого штрафування. Обмеженням таких систем є орієнтація на інший тип учасників руху (мотоцикли) та інші геометричні умови зйомки, тому безпосереднє

перенесення їхніх моделей на велосипедистів часто призводить до втрати точності.

Спеціалізованих систем саме для відеоконтролю велосипедистів і моніторингу використання ними шоломів істотно менше. Частина робіт розглядає велосипедистів як один із класів «уразливих учасників дорожнього руху» і зосереджується на виявленні самого факту присутності велосипеда чи велосипедиста у сцені, не деталізуючи стан засобів захисту. Інші рішення пропонують мобільні застосунки або носимі пристрої, які відстежують використання шолома з боку самого велосипедиста, але не дають можливості організувати зовнішній, інфраструктурний моніторинг. Унаслідок цього питання довготривалої статистичної оцінки частки поїздок без шолома на конкретних ділянках веломережі залишаються недостатньо висвітленими.

Порівняльний аналіз показує, що існуючі системи переважно вирішують локальні задачі: контроль техніки безпеки на об’єкті або виявлення поодиноких порушень серед мотоциклістів на дорогах загального користування. Вони рідко поєднують детекцію шолома саме у велосипедистів, трекінг об’єктів у відеопотоці та агреговану аналітику, орієнтовану на оцінку ризиків за часом і локаціями. Запропонована в даній роботі комп’ютерна система заповнює цю нішу: вона базується на сучасному детекторі YOLOv8, адаптованому до домену міських велодоріжок, забезпечує роботу в реальному часі на доступному апаратному забезпеченні та містить модуль аналітики, що обчислює ковзну частку поїздок без шолома, надаючи органам управління об’єктивні кількісні індикатори безпеки руху велосипедистів.

2 ОБГРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ ТА НАВЧАННЯ МОДЕЛІ

2.1. Вимоги до системи та критерії вибору архітектури нейромережі

Необхідні функціональні вимоги, що напряму випливають із прикладної мети моніторингу. Система має забезпечувати автоматичне виявлення велосипедистів і визначення класу з шоломом / без шолома у відеопотоці в режимі наближеному до реального часу. Цільова продуктивність — не менше 15–25 FPS на споживчому GPU (GTX 1650) при стабільній затримці кадру й відсутності «просідань» під час пікових навантажень. Вихід системи — події/треки з часовими мітками та впевненістю прогнозу, які агрегуються у зрозумілу статистику за періодами та локаціями.

Також встановлюються якісні вимоги і цільові метрики. Мінімально прийнятні показники: mAP@50 на валідації не нижче 0,75, збалансовані Precision/Recall для обох класів, контрольоване число хибних спрацювань на «без шолома» [17]. Робочий поріг conf добирається на максимумі F1 або з пріоритетом чутливості до порушень (щоб не пропускати «без шолома»). Додатково аналізуються PR-криві, матриця помилок, стабільність якості на «важких» сценах (ніч, дощ, оклюзії, нестандартні ракурси).

По-третє, фіксуються обмеження середовища та вимоги до розгортання. Рішення має працювати на ноутбуці/сервері з обмеженою VRAM, підтримувати експорт у ONNX для кросплатформенного деплою, а також просту інтеграцію з бібліотеками відео (OpenCV [3]) та трекінгу. Важливі відтворюваність (зафіксовані гіперпараметри, seed-и), можливість донавчання на нових даних, ієрархічна структура логів/результатів (runs, ваги, звіти) [18]. З міркувань експлуатації передбачені прості механізми моніторингу стану (FPS, черги кадрів, відсоток пропусків).

По-четверте, визначаються критерії вибору архітектур. Для детектора пріоритет надається одноетапним моделям із передбачуваним компромісом «точність/латентність» і сімейством масштабів (YOLOv8 n/s/m або EfficientDet

D0/D1), що дозволяє адаптуватися під ресурс. Оцінюється підтримка сильних аугментацій, стабільність навчання, наявність інструментів валідації/експорту, спільнота та зрілість фреймворку. Для можливого дворівневого підходу (детекція + класифікація) критеріями є легкий бекбон (EfficientNet-B0/MobileNetV3), малий розмір моделі та швидкий інференс.

По-п’яте, формулюються вимоги до надійності та робастності. Архітектура має коректно працювати за змінних умов освітлення й погоди, бути стійкою до оклюзій і дрібних об’єктів, що забезпечується добором даних і доменними аугментаціями (яскравість/контраст, блюр руху, геометричні варіації). Для підрахунку унікальних велосипедистів застосовується трекінг і/або логіка підсумування за треками, щоб уникати дублювання детекцій по кадрах. У сукупності ці вимоги та критерії задають рамки технічного вибору, в яких оптимальним стартовим рішенням є легковаговий одноетапний детектор із підтримкою реального часу та прозорим шляхом до промислового розгортання.

2.2 Вдосконалення методу розпізнавання об’єктів на зображенні під час дорожнього руху

Перше вдосконалення стосується ядра детекції: обрано одноетапну архітектуру YOLOv8 з anchor-free регресією рамок і «decoupled head», що зменшує накладні витрати на узгодження якорів і пришвидшує збіжність. Для поліпшення виявлення малих об’єктів (голова/шолом) використано піраміду ознак із посиленими нижніми рівнями, збільшений розмір зображення на тренуванні, а також покадрове масштабування та випадкові кропи з обмеженням мінімальної видимої частки об’єкта. Баланс втрат налаштовано за рахунок ваг для bbox-компоненти (CIoU) та класової складової (focal/BCE), що підвищує чутливість до класу Without Helmet за дисбалансу вибірки.

Друге вдосконалення — доменні аугментації, орієнтовані на умови дорожнього руху: варіації освітлення (ніч/контрове світло/тіні), погодні ефекти (легкий дощ/туман через шум та розмиття), motion-blur, перспективні зсуви та часткові оклюзії (капюшони/кепки). Параметри аугментацій підібрано так, щоб не

руйнувати семантику об’єкта, але розширювати розподіл тренувальних прикладів. Додатково застосовано керовану стратифікацію даних за камерами/локаціями, щоб уникнути витоку сцен між train/val і підсилити узагальнюваність на нових ділянках велоінфраструктури.

Третє вдосконалення — постобробка та темпоральна узгодженість. Поріг упевненості калібрується на валідації з урахуванням пріоритету мінімізації пропусків класу Without Helmet; NMS налаштовано з адаптивним iou_thres і клас-агностичним режимом у «щільних» сценах. Для відео застосовано трекер (наприклад, BYTETrack/DeepSORT), який агрегує покадрові детекції в унікальні треки; підсумковий клас треку визначається за правилом більшості або за максимумом сумарної впевненості. Це знижує дублювання підрахунків одного велосипедиста на сусідніх кадрах і стабілізує рішення в умовах короткочасних оклюзій.

Четверте вдосконалення — експлуатаційна оптимізація. Для прискорення інференсу використано FP16 (збереження точності завдяки масштабуванню ознак), експорт у ONNX та, за потреби, прискорення ONNX Runtime/TensorRT. Передбачено профілювання FPS/латентності й адаптацію робочих параметрів (зменшення розміру зображення, обрізання області інтересу ROI, частота вибірки кадрів) під обмеження обчислювальних ресурсів. У підсумку вдосконалений метод поєднує точність детекції на складних міських сценах із реальним часом обробки, забезпечуючи надійну статистику «з шоломом / без шолома» для подальшої аналітики.

П’яте вдосконалення — керована підготовка даних і активне донавчання. Після початкового запуску модель застосовується на нових потоках, а кадри з низькою впевненістю або конфліктними прогнозами автоматично потрапляють у «буфер перевірки». Оператор швидко верифікує лише складні приклади (hard samples), після чого запускається короткий цикл донавчання з підвищеною вагою цих випадків. Така схема скорочує витрати розмітки, підвищує чутливість до рідкісних ситуацій (незвичні каски, дощ/сутінки, нетипові ракурси) і стабілізує показники на нових локаціях без повного перетренування.

Шосте вдосконалення — експлуатаційний моніторинг та калібрування. Під час роботи система збирає технічні метрики (FPS, латентність, використання VRAM/CPU) і якісні індикатори (частка відфільтрованих детекцій, розподіл впевненостей, частота NMS-конфліктів) з тривогами за порогами. Періодично виконується калібрування порогів (conf/NMS IoU) за валідним піднабором із останніх даних, а також перевірка стабільності на «контрольних» сценах (ніч, дощ, щільний потік). За потреби активується спрощений режим (менший imgsz, ROI-обрізання), що гарантує цільовий FPS на слабшому обладнанні, з мінімальним впливом на точність у виробничих сценаріях.

2.3 Підготовка даних, доповнення датасету і виконання аугментації

Перший варіант передбачає інвентаризацію джерел, уніфікацію форматів і базову перевірку якості. Вихідні анотації PASCAL VOC (XML) конвертуються у формат YOLO (TXT з нормованими координатами cx, cy, w, h) з жорсткою перевіркою відповідності імен файлів, наявності розмірів кадру, коректності меж рамок і відсутності «битих» зображень [16]. Далі формують маніфест датасету (список зображень із хешами, шляхами, кількістю рамок і класами) — це полегшує аудит і відтворюваність. Розділення виконується як stratified split 80/20 (train/val) із фіксованим seed і, за можливості, зі стратифікацією за камерами/локаціями/датами зйомки, щоб уникнути «витоку» сцен у валідацію. Для фінальної оцінки (за наявності часу) виділяють test-піднабір, який не використовується під час підбору гіперпараметрів

Другий варіант — аналіз структури вибірки та балансування класів. Обчислюють розподіли: кількість прикладів у класах (With Helmet / Without Helmet), спектр розмірів об’єкта (малі/середні/великі рамки), гістограми яскравості/контрасту, частку «важких» умов (ніч, туман, дощ, зустрічне світло), кількість рамок на кадр. Якщо спостерігається дисбаланс (типово With Helmet ≫ Without Helmet), застосовують комбіновані стратегії: таргетовані аугментації «рідкісного» класу, oversampling «важких» сцен, добір додаткових прикладів або ж підвищення ваги класу у функції втрат. На цьому ж етапі коректують номенклатуру

класів (єдині назви/ID) і політику маркування прикордонних випадків (капюшон, кепка, навушники), щоб зменшити шум еталонів.

Третій варіант — геометричні та фотометричні аугментації, які допомагають моделі узагальнювати поза межами тренувальних прикладів. До геометричних належать випадкові кропи (із мінімальною видимою часткою об’єкта), масштабування/повороти, горизонтальні віддзеркалення, перспективні зсуви; до фотометричних — зміни яскравості/контрасту/балансу білого, колірний дрифт, гама-корекція, легкий Gaussian/ISO-«шум». Додатково застосовують MixUp/CutMix/Mosaic (з помірними параметрами), які ефективно збагачують контексти і допомагають при дисбалансі класів, але потребують акуратного налаштування, щоб не зруйнувати семантику малого об’єкта (голова/шолом). Для задачі «шолом/без шолома» рекомендовано обмежувати силу кропів і поворотів, зберігаючи читабельність деталей у верхній частині тулуба.

Четвертий варіант — доменні аугментації, що імітують умови міського відеомоніторингу. Це легкий motion blur (рух камери/об’єкта), «мокра» оптика (м’яке розмиття + шум), часткові оклюзії (штучні прямокутні маски в ділянці голови), «нічні» сцени (затемнення + підвищення контрасту), жорсткі тіні/контрове світло, дощ/туман (простими шарами шуму/розмиття без агресивного спотворення текстур). Доменно орієнтовані трансформації підбирають експериментально: їхня інтенсивність має підвищувати стійкість моделі, не призводячи до деградації на «нормальних» умовах. Практика показує, що помірне поєднання доменних і базових аугментацій покращує Recall на класі Without Helmet, зберігаючи Precision у прийнятних межах.

П’ятий варіант — контроль якості даних і безперервне вдосконалення. У конвеєр включають автоматичні тести: перевірка «порожніх» label-файлів [5], меж рамок, відсутності негативних значень і виходу за кордони кадру, а також детекція майже дублікатів (perceptual hash) для валідних піднаборів. Після першого циклу навчання запускають active learning: кадри з низькою впевненістю або з конфліктними прогнозами потрапляють до «черги перевірки», де оператор швидко верифікує еталон; далі модель коротко донавчають із підвищеною вагою цих

«важких» прикладів. Усі версії датасету (маніфест, скрипти конвертації VOC→YOLO, data.yaml, статистики) зберігаються під контролем версій, що гарантує відтворюваність, прозорий аудит та коректне порівняння результатів між експериментами.

Шостий варіант — це політика балансування і вагових коефіцієнтів у тренуванні. За помітного перекосу на користь With Helmet доцільно комбінувати class weighting у функції втрат (підвищення внеску Without Helmet), oversampling рідкісних сцен (ніч, дощ, далекі плани) та цілеспрямоване формування мінібатчів з обмеженням максимальної частки «легких» прикладів. Додатково корисними є label smoothing для зменшення перенавчання на шумних еталонах, а також «hard-negative mining» — цілеспрямований підбір негативів, де шолом відсутній, але є схожі об’єкти (капюшони, кепки, навушники). Така стратегія знижує чутливість моделі до випадкових артефактів і підвищує роздільну здатність між класами.

Сьомий варіант — керування сценаріями аугментацій у часі (augmentation schedule). На ранніх епохах доцільно використовувати більш інтенсивні фотометричні й геометричні перетворення (Mosaic/CutMix, сильні колірні зсуви), щоб розширити простір ознак і прискорити узагальнення. На пізніх епохах інтенсивність аугментацій поступово зменшують, віддаючи пріоритет «реалістичним» трансформаціям і доменним ефектам (легкий motion blur, м’які тіні, помірне затемнення), аби модель «доточила» кордони класів на умовах, близьких до продакшну. Разом із цим узгоджують learning rate schedule (наприклад, cosine decay) і EMA ваг, щоб стабілізувати збіжність на фінальному етапі.

Восьмий варіант це розширення даних синтетикою та напівавтоматичною розміткою. За дефіциту кадрів «без шолома» можна застосувати обережний copy–paste об’єктів (голів з різними аксесуарами) між схожими сценами, підтримуючи консистентність масштабу та освітлення, або ж генерувати напівсинтетичні сцени з доданими ефектами дощу/туману. Після первинного навчання запускають pseudo-labeling: модель прогнозує розмітку на нерозмічених потоках, а оператор верифікує лише невпевнені/конфліктні випадки — це суттєво скорочує витрати ручної розмітки й прискорює ітерації. Усі синтетичні та напівавтоматичні дані чітко

маркуються у маніфесті, а їхній вплив контролюється окремими валідаційними піднабором, щоб уникнути зсуву оцінки якості.

2.4 Навчання моделі

2.4.1 Налаштування та навчання моделі: гіперпараметри, регуляризація, early-stoping

Першим кроком виконано підготовку середовища та перевірку апаратного прискорення: створено ізольоване середовище conda, встановлено PyTorch із CUDA, пакет Ultralytics та OpenCV. Перевірено відповідність версій (torch/torchvision/cudnn), доступність GPU (GTX 1650) і базову продуктивність інференсу на «сухому» прогоні. Структуру проєкту організовано так, щоб відтворювано зберігати артефакти: data/, datasets/helmet/, runs/helmet/…, скрипти конвертації й конфігурації data.yaml. Така організація дозволяє швидко повторювати експерименти й порівнювати результати.

На рисунку 2.2 наведено алгоритм підготовки даних та процесу навчання моделі YOLOv8, застосованої в роботі для виявлення велосипедистів із шоломом і без шолома. На першому етапі формуються початкові набори зображень із відкритого датасету Helmet Detection та власних фотографій, після чого виконується розмітка об’єктів і перевірка якості анотацій. Далі розмітка конвертується у формат YOLO, що містить лише номер класу та нормовані координати прямокутників. Отриманий датасет розбивається на тренувальну, валідаційну та тестову підмножини, до тренувальної частини застосовуються геометричні, фотометричні та доменні аугментації з метою підвищення узагальнювальної здатності моделі. На основі підготовлених даних виконується навчання моделі YOLOv8 із заданими гіперпараметрами, після чого її якість оцінюється за метриками mAP, precision, recall, а також за допомогою матриці неточностей і PR-кривих. За результатами оцінювання обирається найкращий варіант моделі, який зберігається у вигляді файлу best.pt та використовується під час роботи комп’ютерної системи моніторингу.

Другим кроком налаштовано тренувальний конфіг: вибрано модель yolov8n

як базову (компроміс швидкодії/точності), розмір зображення imgsz=640, початковий batch=8 (з пониженням до 4 при дефіциті VRAM), epochs=50–100 залежно від експерименту, workers=0 для стабільності на Windows.

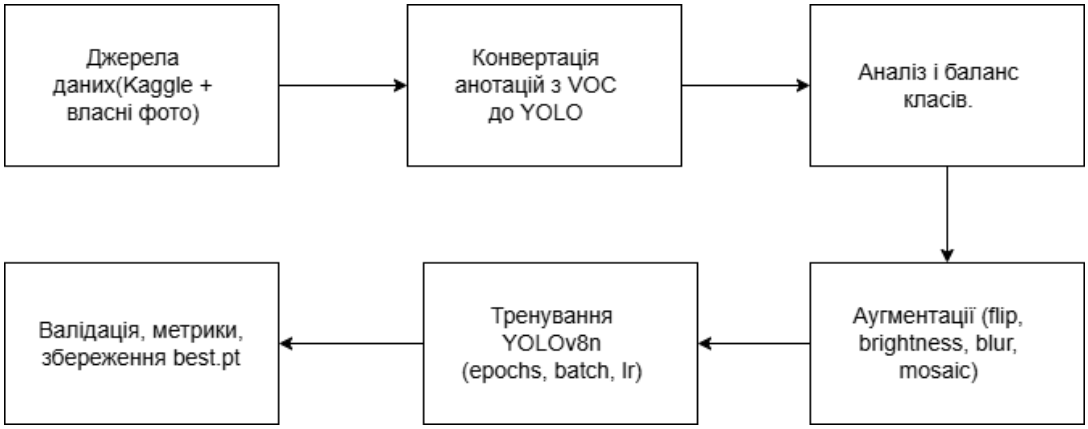


Рисунок 2.1 — Алгоритм підготовки даних та процесу навчання моделі

У файлі data.yaml зафіксовано два класи — With Helmet, Without Helmet. Для контроля випадковості встановлено фіксовані seed та однакові правила розбиття train/val (80/20). Вміст файла args.yaml у папці запуску зберігає фактичні параметри кожного експерименту.

Необхідним елементом стала стратегія аугментацій. Увімкнено стандартні перетворення Ultralytics (масштабування, фліпи, колірні зсуви), а також помірний Mosaic/CutMix на ранніх епохах для розширення контекстів. Додано доменні ефекти: легкий motion blur, варіанти затемнення/контрасту, обережні перспективні зсуви й часткові оклюзії (імітація капюшона/рук на рівні голови). На пізніх епохах інтенсивність аугментацій знижувалась, аби модель донавчалась на «реалістичних» варіаціях. Для дисбалансу класів використовувались class-weight у втраті та таргетоване накладання шарів oversampling сцен «без шолома».

Четвертим кроком також слід приділити увагу час функції втрат і оптимізації. Для локалізації рамок застосовано CIoU-loss, для класифікації й об’єктності — BCE з фокальною модифікацією за потреби (приглушення «легких» негативів). Оптимізатор — SGD/AdamW із cosine decay для швидкої початкової

збіжності та плавного «доточування». Використано EMA ваг, що зменшує шум оновлень і підвищує стабільність валідаційних метрик.

П’ятим кроком стала валідація та добір робочих порогів. Після кожної сесії аналізувались results.png, PR_curve.png, BoxPR/BoxR/BoxF1_curve.png, confusion_matrix.png у каталозі запуску; за їхніми кривими підбирався поріг conf (зазвичай 0.30–0.45) із пріоритетом Recall для класу Without Helmet при контрольованому падінні Precision. Для перевірки узагальнюваності додатково тестували інференс на власних зображеннях/відео та на «важких» сценах (ніч/дощ/далекі плани), після чого за потреби повторювали навчання з оновленими аугментаціями або збільшеною кількістю епох.

Шостим кроком виконано документування й підготовку матеріалів до звіту. Всі ключові артефакти збережено у runs/helmet/yolov8n_kaggle/: results.png, confusion_matrix.png, labels.jpg, приклади train_batch*.jpg, val_batch*_labels.jpg, val_batch*_pred.jpg, а також results.csv із покадровими метриками; ваги моделі — у weights/best.pt. Для ілюстрацій у роботі використано ці файли з папки експерименту: зовнішній вигляд графіків навчання, приклади батчів і матриця помилок наведені нижче, де посилання у тексті вказують на відповідні зображення з каталогу runs.

У процесі навчання дані обробляються не по одному зображенню, а пакетами (батчами). Батч — це група з N зображень, яку модель пропускає через себе за один крок навчання (одне оновлення ваг). Відповідно, батч міток/розмітки — це набір анотацій для цих N зображень (класи та координати обмежувальних рамок), які використовуються для обчислення функції втрат. Для зручності контролю якості Ultralytics автоматично формує колажі батчів: на них видно вихідні зображення та накладену розмітку/прогнози.

Під час навчання детектора оптимізація параметрів нейронної мережі виконується шляхом мінімізації функції втрат, яка узагальнено враховує точність локалізації обмежувальних рамок, коректність класифікації та помилки, пов’язані з хибними спрацюваннями на фоні. Для моделей сімейства YOLO функція втрат може бути подана як сума складових: (1) похибка координат центра рамки та її

розмірів, (2) похибка “впевненості” у наявності об’єкта (objectness) та (3) похибка класифікації за класами, при цьому використовуються вагові коефіцієнти для балансування внеску локалізації та фону [24]. Узагальнений вигляд такої функції наведено формулою 2.1.

$$
\begin{aligned}
L \;=\; & \lambda_{coord} \sum_{i=1}^{S^2} \sum_{j=1}^{B} \mathbb{1}_{ij}^{obj} \left((x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right) \\
& + \lambda_{coord} \sum_{i=1}^{S^2} \sum_{j=1}^{B} \mathbb{1}_{ij}^{obj} \left(\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right) \\
& + \sum_{i=1}^{S^2} \sum_{j=1}^{B} \mathbb{1}_{ij}^{obj} \left(C_i - \hat{C}_i \right)^2 + \lambda_{noobj} \sum_{i=1}^{S^2} \sum_{j=1}^{B} \mathbb{1}_{ij}^{noobj} \left(C_i - \hat{C}_i \right)^2 \\
& + \sum_{i=1}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in classes} \left(p_{i(c)} - \hat{p}_{i(c)} \right)^2
\end{aligned}
$$

де I_{ij}^{obj} —індикатор того, що в комірці і є об’єкт і j-та рамка відповідає за нього;

I_{ij}^{noobj} — індикатор, що об’єкта немає (для штрафу за хибні спрацювання);

x, y, w, h — параметри рамки (центр/розміри),

C — “objectness/впевненість”,

p(C)— ймовірність класу;

λ_{coord} і λ_{noobj} — ваги для балансування внеску локалізації та фону.

На рисунку 2.3 наведено динаміку основних функцій втрат для навчальної та валідаційної вибірок (box_loss, cls_loss, dfl_loss), а також зміну метрик якості детекції: precision, recall, mAP@0.5 та mAP@0.5:0.95. Суцільна лінія відображає значення по епохах, пунктир — згладжений тренд. Загальне зниження втрат і

стабілізація метрик наприкінці навчання свідчать про збіжність процесу та покращення якості розпізнавання.

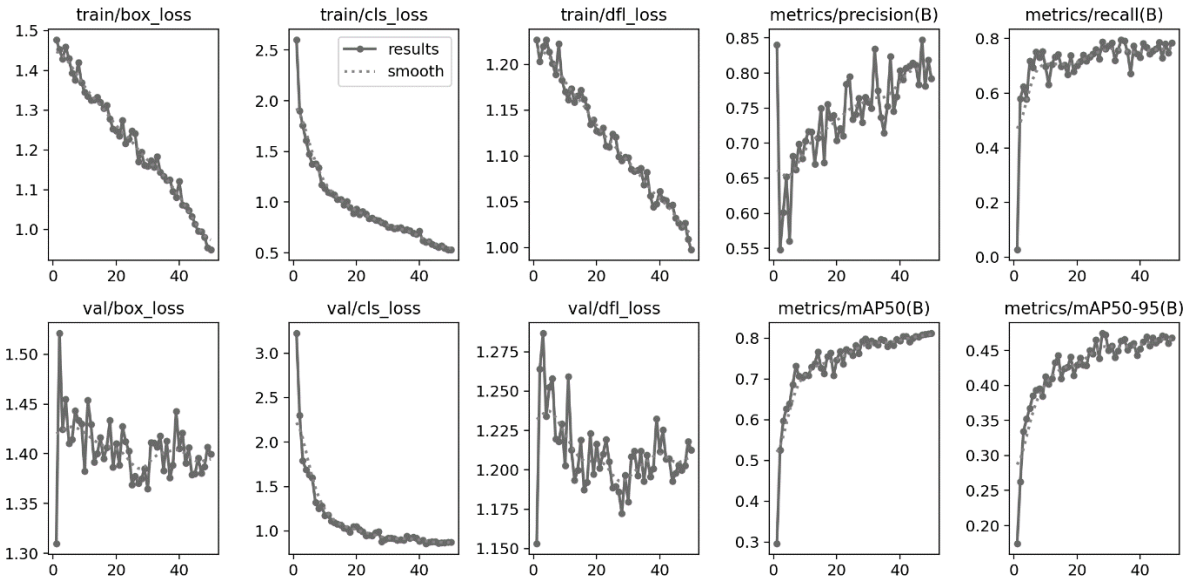


Рисунок 2.2 — Графіки навчання моделі

Для оцінювання якості детекції важливо розрізняти точність локалізації об’єкта та впевненість моделі у правильності розпізнавання класу. Узгодженість між передбаченою обмежувальною рамкою та еталонною розміткою кількісно описується метрикою Intersection over Union (IoU) — відношенням площі перетину рамок до площі їх об’єднання. Чим більшим є значення IoU, тим точніше рамка накриває об’єкт; на практиці часто застосовують порогове значення (наприклад, IoU ≥ 0,5) для відбору коректних спрацювань і подальшого аналізу результатів. Узагальнений вигляд такої функції наведено формулою 2.2.

$$IoU = \frac{Area(B_{pred} \cap B_{gt})}{Area(B_{pred} \cup B_{gt})},$$

де $S_{\text{перетину}}$ — площа перетину передбаченої та еталонної рамок,

$S_{\text{об’єднання}}$ — площа об’єднання цих рамок.

Крім того, у моделях сімейства YOLO оцінка “надійності” передбачення для

певного класу може бути подана як добуток умовної ймовірності класу за наявності об’єкта, ймовірності наявності об’єкта та міри перекриття IoU. Такий підхід дозволяє одночасно враховувати і правильність класифікації, і точність локалізації під час ранжування та фільтрації детекцій. Узагальнений вигляд такої функції наведено формулою 2.3.

$$Score = P(Class \mid Object) \cdot P(Object) \cdot IoU,$$

де $P(Class \mid Object)$ — умовна ймовірність належності до класу i і за умови, що об’єкт присутній;

P(Object) — ймовірність наявності об’єкта в комірці/позиції;

IoU — міра перекриття рамки з еталонною розміткою.

На рисунку 2.4 показано колаж із кількох кадрів, що одночасно подаються на вхід моделі під час навчання. Поверх кожного об’єкта нанесено обмежувальні рамки, а поруч — ідентифікатор класу (наприклад, 0/1), що відповідає конкретній категорії в датасеті. Таке представлення дає змогу швидко перевірити коректність розмітки (чи рамки потрапляють на шолом/голову) та наявність складних випадків (часткові перекриття, мала відстань, різні ракурси). Сірі поля в окремих фрагментах — це заповнення (padding) під час формування мозаїки/вирівнювання розмірів кадрів.

Рисунок 2.3 — Приклад набору навчальної вибірки з накладеною розміткою

На рисунку 2.5 наведено приклади з валідаційного набору, де поверх об’єктів відображено рамки та текстові назви класів With Helmet або Without Helmet. Така візуалізація використовується для контролю якості розмітки та перевірки, що класи інтерпретуються правильно і відповідають прийнятому словнику категорій. Батчі з валідації дозволяють оцінити, наскільки набір прикладів є різноманітним (освітлення, ракурси, масштаб, фони) та чи немає систематичних помилок у розмітці.

Рисунок 2.4 — Приклад набору валідаційної вибірки з назвами класів

Матриця помилок показує розподіл правильних і хибних класифікацій між двома класами, а також випадки віднесення об’єктів до background (пропуски/фонові спрацювання). Значення на головній діагоналі відповідають коректним розпізнаванням, позадіагональні елементи — помилкам переплутування класів або помилкам фону. Така візуалізація дозволяє швидко оцінити, які саме типи помилок трапляються найчастіше та між якими класами виникає плутанина.

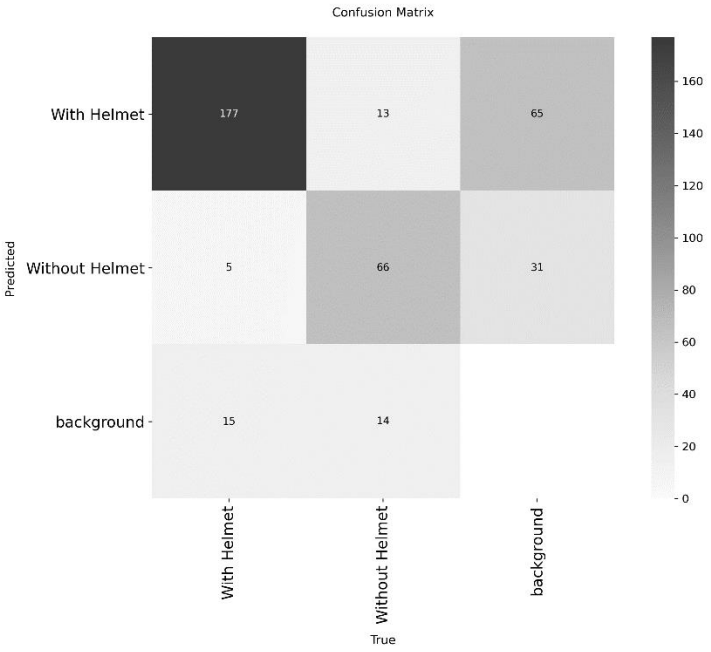


Рисунок 2.5 — Матриця помилок

Наступним кроком проведено тонке налаштування продуктивності під цільовий FPS. Експериментально підібрано розмір малюнку imgsz (640→512 для «важких» сцен), вимкнено зайві перетворення на останніх епохах, увімкнено інференс у FP16 там, де це не впливало на стабільність. Для «щільних» кадрів з багатьма об’єктами протестовано class-agnostic NMS та адаптивний поріг IoU, що знижує кількість дубльованих рамок. Результати профілювання фіксувались у таблиці з полями FPS/latency/VRAM, що дозволило оперативно знаходити вузькі місця.

Приділимо увагу виводу та інтеграції: натреновану модель експортовано в ONNX для прискореного інференсу та сумісності з різними рантаймами. Перевірено тотожність виходів PyTorch ↔ ONNX на вибірці з 200 кадрів (середня різниця score < 1e-3), що зафіксовано у results.csv.

2.4.2 Донавчання моделі

Після первинного навчання моделі YOLOv8 на основному наборі даних (Helmet Detection з Kaggle) було виявлено характерну проблему: у сценах із кімнатним або слабким освітленням модель часто помилково відносила обличчя

без шолома до класу «With Helmet». Особливо це проявлялося під час тестування в режимі реального часу через вебкамеру ноутбука, коли елементи одягу (капюшон, комір, тінь від волосся) інтерпретувалися як шолом. Для зменшення кількості таких хибних спрацьовувань було прийнято рішення виконати додатковий етап донавчання (fine-tuning) моделі на розширеній вибірці зображень велосипедистів у шоломах та без шоломів.

Для другого етапу навчання як додатковий набір даних було обрано відкритий датасет Bike Helmet Detection з платформи Roboflow[21], що надається відразу у форматі YOLOv8. Ця вибірка містить зображення велосипедистів у різних умовах зйомки: денні та вечірні сцени, різні ракурси камер, наявність фонових об’єктів та часткових перекриттів, що суттєво відрізняється від більш «лабораторних» кадрів з основного набору. Така різноманітність дозволяє моделі краще узагальнювати ознаки шолома і голови, а також адаптуватися до реалістичних умов експлуатації системи. Приклад картинок з цього датасету зображено на рис. 2.7)

Рисунок 2.6 — Приклади зображень із додаткового набору

Після розпакування архіву структура датасету Roboflow містила стандартні для YOLOv8 підкаталоги train, valid, test з відповідними папками images та labels, а також файл data.yaml, де були визначені шляхи до підмножин і список класів. Для

коректної інтеграції в локальне середовище шляхи у файлі data.yaml було відредаговано таким чином, щоб вони відповідали реальному розташуванню датасету на диску (train: train/images, val: valid/images, test: test/images). Класи залишилися незмінними й повністю сумісними з попередньою моделлю: 0 – With Helmet, 1 – Without Helmet, що дозволило уникнути змін у коді інференсу. Структура теки зображена на рис.2.8.

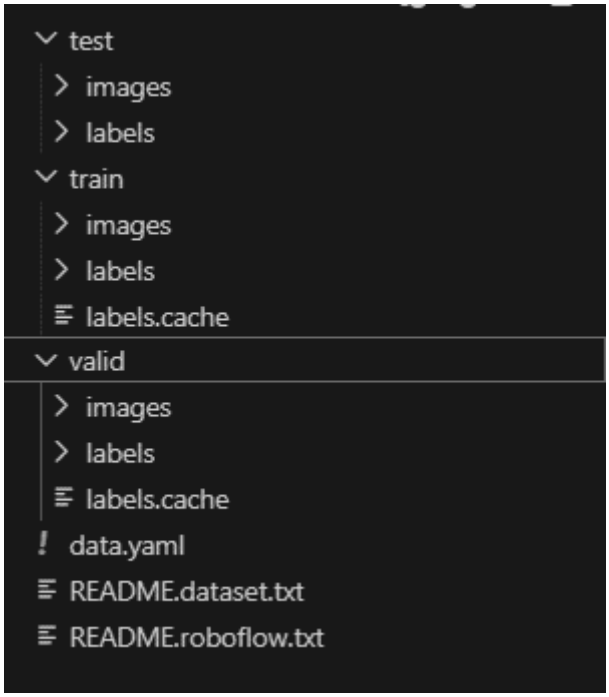


Рисунок 2.7 — Структура теки для донавчання

Донавчання виконувалося у форматі transfer learning: як початкові ваги використовувався файл best.pt, отриманий після першого етапу навчання на базовому датасеті. Команда навчання Ultralytics YOLOv8 запускалася з параметрами на кшталт model=best.pt, data=data.yaml, epochs≈50, imgsz=640, batch=16, збереженням результатів у окремому проєкті (project=... rf_retrain). Таким чином, на другому етапі мережа не «вчилася з нуля», а донастроювала вже наявні ваги під нові сцени та статистику даних, що значно скорочує час навчання й дає кращу збіжність.

У процесі донавчання використовувались стандартні аугментації, реалізовані фреймворком YOLOv8: випадкові повороти та масштаби кадрів, горизонтальні

віддзеркалення, зміни яскравості й контрасту. Це особливо важливо для задачі детекції шоломів, оскільки система має бути стійкою до різних куточків зйомки, часткового закриття шолома іншими об’єктами, а також до зміни освітлення (наприклад, перехід з вулиці у приміщення). Передбачалося, що додаткові приклади із Roboflow-датасету в поєднанні з аугментаціями дозволять моделі краще розрізняти реальні шоломи та «псевдо-шоломи» (капюшони, зачіски, фрагменти фону). Результат навчання показано на рис 2.9 .

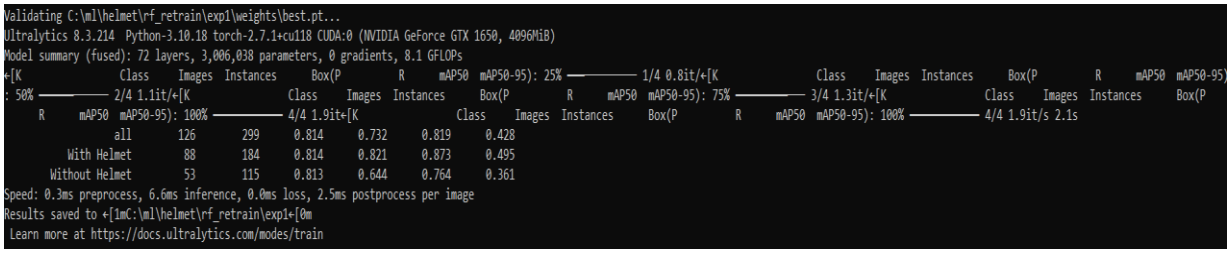


Рисунок 2.8 — Результати навчання в робочому середовищі

Логи навчання та графіки, згенеровані YOLOv8 (файл results.png), показали стабільне зменшення функції втрат і покращення метрик для обох класів. Особливу увагу приділяли показникам точності (precision) та повноти (recall) саме для класу «Without Helmet», оскільки він є критичним з точки зору виявлення порушень. Порівняння кривих до та після донавчання демонструє, що на другому етапі модель виходить на кращий баланс між двома класами, зменшуючи кількість хибних спрацьовувань «Helmet» на зображеннях без шолома. Результати донавчання зображено на рис. 2.10.

Для більш наочної оцінки якості моделі після донавчання було побудовано матриці помилок (confusion matrices) на валідаційному піднаборі. Порівняння двох матриць – для базової та донавченої моделі – показало, що кількість хибних позитивних спрацьовувань класу «With Helmet» для прикладів без шолома суттєво зменшилася, тоді як правильних спрацьовувань «Without Helmet» стало більше. Це означає, що система краще розпізнає відсутність засобу захисту, що й було основною метою додаткового етапу навчання. Матрицю помилок зображено у Додатку Г.

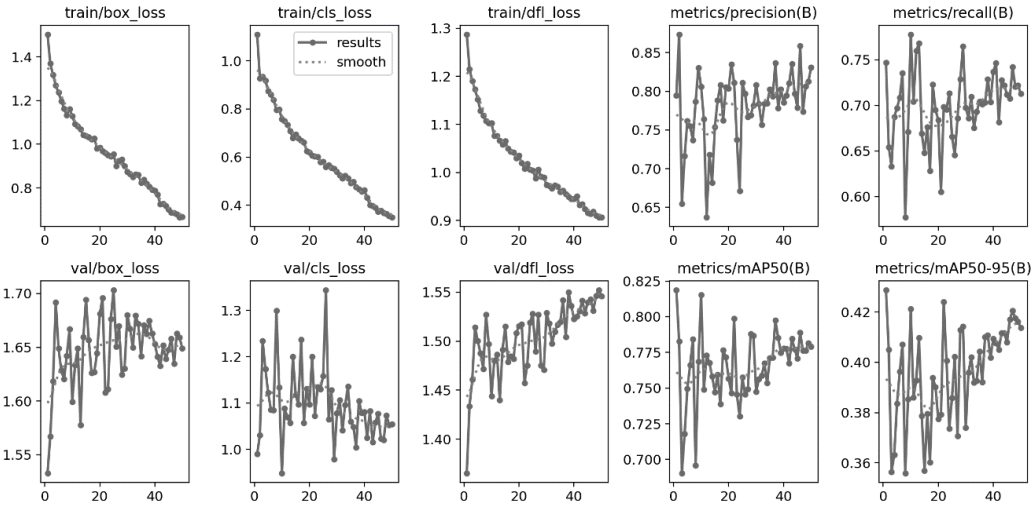


Рисунок 2.9 — Графіки результатів донавчання

Окремо було проведено якісну оцінку роботи системи в режимі реального часу з вебкамери ноутбука. Для цього використовувалися ті самі сцені, на яких базова модель систематично помилялася: кадри з кімнатним освітленням, фоном з яскравими вертикальними об’єктами, користувач без шолома, але в одязі з коміром або капюшоном. Після донавчання кількість кадрів, на яких система хибно позначала такі сцени як «Helmet», істотно зменшилась, а в значній частині випадків модель коректно видавала клас «No helmet» з упевненістю близько 0.6–0.7 навіть у поганих умовах освітлення. Приклад наведено на рис. 2.11.

Рисунок 2.10 — Приклад точності в поганих умовах камери та освітлення

Таким чином, виконаний етап донавчання на додатковому датасеті дозволив адаптувати детектор шоломів до реальних умов експлуатації в рамках розробленої комп’ютерної системи моніторингу велосипедистів. У розділі 4 ці результати будуть деталізовані у вигляді кількісних метрик (точність, повнота, mAP) та додаткових прикладів роботи програми, однак уже на рівні опису другого розділу можна зробити висновок, що саме включення різноманітніших даних та повторне навчання моделі є ключовим фактором підвищення надійності системи в задачі розпізнавання шолома. Алгоритм навчання зображено у Додатку В.

Він включає такі етапи: спочатку формується набір вхідних даних на основі датасету Kaggle Helmet Detection, який слугує базою для початкового навчання моделі. Далі виконується анотування та перевірка розмітки: для кожного зображення задаються об’єкти у вигляді обмежувальних рамок (BBox) і відповідних класів, після чого проводиться контроль якості розмітки (повнота, точність позиціювання рамок, відсутність помилок у класах). Після підготовки даних здійснюється перше навчання моделі, за результатами якого оцінюється коректність налаштувань і базова якість детекції на валідаційних прикладах.

Наступним кроком виконується повторне навчання з урахуванням додаткових даних — зокрема із датасету Roboflow Bike Helmet Detection, що дозволяє підвищити різноманітність прикладів і покращити узагальнювальну здатність моделі в реальних умовах (різні ракурси, освітлення, типи шоломів і сцени). Після отримання стабільних результатів модель та супровідні артефакти експерименту передаються на етап інтеграції в модуль моніторингу, де реалізовується використання навчених ваг у прикладній частині системи: обробка відеопотоку/зображень, формування результатів детекції та подальше відображення або збереження даних.

2.5 Оцінювання якості і вибір робочого порога

Оцінювання здійснюється на відкладеній val-вибірці за стандартним протоколом детекції: обчислюються mAP@50 і mAP@50–95, клас-специфічні Precision/Recall/F1, а також будуються PR-криві для кожного класу. Додатково

аналізуються confusion matrix (звичайна і нормована) та розподіли IoU і score; окремо звітуються метрики для малих/середніх об’єктів. Для контролю стабільності проводиться повторна валідація на піднаборах «складних сцен» (ніч, дощ, далекі плани, оклюзії), що дозволяє відстежити, де саме падає якість і чи не виникає зсув домену.

Вибір робочого порога conf виконується як задача пошуку «робочої точки» на PR-кривій з урахуванням вартості помилок: для класу Without Helmet пріоритет — Recall (мінімізувати пропуски), після чого перевіряється прийнятний рівень Precision (обмеження хибних спрацювань). Базово точку фіксуємо на максимумі F1 або за заданим мінімальним Recall (наприклад, ≥0.80) і далі коригуємо за експлуатаційними вимогами. Додатково калібрується IoU-поріг NMS (class-agnostic для щільних сцен) та перевіряється, як зміни порогів впливають на mAP і кількість детекцій на кадр.

На цьому рівні повторно обчислюються Precision/Recall/F1, а також перевіряється стабільність рішень при короткочасних оклюзіях і сплесках шуму. Підібрані пороги (conf/NMS, правила агрегації треків) фіксуються в конфігурації релізу; поруч зберігаються контрольні графіки і матриці помилок, що дозволяє відтворити вибір робочої точки та коректно порівнювати наступні версії моделі. Зовнішній вигляд кривої точності та повноти зображено на рис.2.12

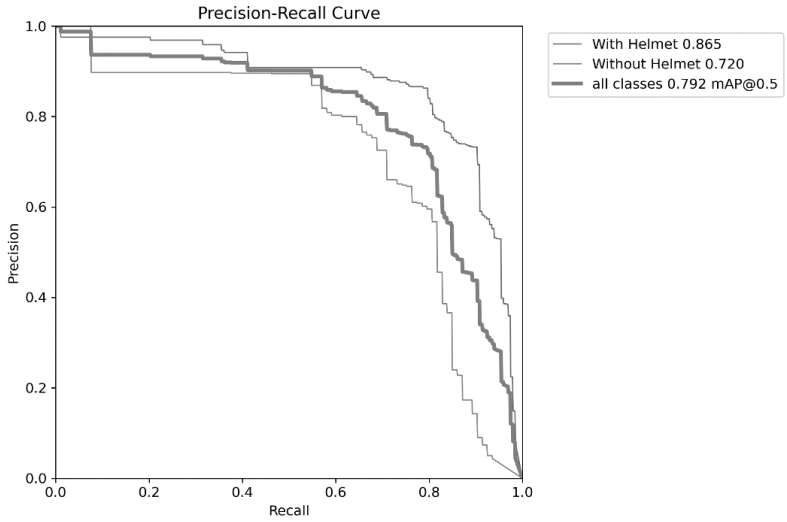


Рисунок 2.11 — Крива точності та повноти

На графіку подано криві Precision–Recall для обох класів моделі та усередненний результат по всіх класах. Світло-блакитна крива відповідає класу «With Helmet» і має площу під кривою (AP) близько 0,865, що свідчить про високу точність при широкому діапазоні значень повноти: до значень recall ~0,6–0,7 precision утримується на рівні вище 0,9, а помітне падіння починається лише в області дуже високої повноти, коли модель намагається «захопити» майже всі позитивні приклади.

Помаранчева крива відповідає класу «Without Helmet» з AP ≈ 0,720. Вона розташована нижче, ніж для класу «With Helmet», тобто модель гірше відокремлює відсутність шолома: при зростанні повноти вже від значень recall ~0,4–0,5 точність помітно падає. Це означає, що для виявлення більшості порушень (велосипедистів без шолома) доводиться миритися з більшим числом хибних спрацювань. Товста темно-синя крива відображає усереднений результат по всіх класах (mAP@0.5 ≈ 0,792) і показує загальну якість детектора; її положення між двома кривими підтверджує, що система в цілому краще розпізнає наявність шолома, ніж його відсутність, але забезпечує прийнятний компроміс між точністю і повнотою для обох типів об’єктів.

3 РОЗРОБКА ТА РЕАЛІЗАЦІЯ СИСТЕМИ МОНІТОРИНГУ ТА ОЦІНКИ БЕЗПЕКИ РУХУ ВЕЛОСИПЕДИСТІВ

У цьому розділі викладено практичну реалізацію комп’ютерної системи моніторингу та оцінки безпеки руху велосипедистів: від архітектури та потоків даних до алгоритмів інференсу і підрахунку показників. Система складається з модулів захоплення відео (RTSP/HTTP), попередньої обробки та інференсу (YOLOv8 на GPU), трекінгу для об’єднання покадрових детекцій у унікальні проходи, а також сервісів логування й аналітики, що агрегують кількість фіксацій «з шоломом/без шолома» по камерах і періодах. Окрему увагу приділено вимогам реального часу, стійкості до умов зйомки (ніч/дощ/оклюзії) та відтворюваності: конфігурації, версії та ваги моделі зберігаються разом з артефактами експериментів. Наведено рішення щодо структури API/схем даних та робочих порогів, а також засоби експлуатаційного контролю (FPS, латентність, частка хибних спрацювань), які забезпечують стабільну роботу системи на доступному апаратному забезпеченні.

3.1 Розробка архітектури та структурної схеми комп’ютерної системи моніторингу та оцінки безпеки руху велосипедистів

На контекстній діаграмі система моніторингу та оцінки безпеки руху велосипедистів представлена як один узагальнений процес — модуль детекції та розпізнавання, навколо якого показані основні зовнішні сутності та обміни даними. Вхідним джерелом інформації виступає камера або відеофайли, з яких надходить відеопотік/кадри для подальшої обробки. До системи також підключено оператора, який взаємодіє з рішенням через інтерфейс і отримує результати у вигляді візуалізації та статистики. Окремо виділено базу даних як сховище для накопичення результатів роботи модуля та подій моніторингу, а також блок експорту даних для вивантаження сформованих результатів у зручному для подальшого використання вигляді. Розглянемо основні кроки розробки системи за допомогою контекстної діаграми, зображеної на рис 3.1.

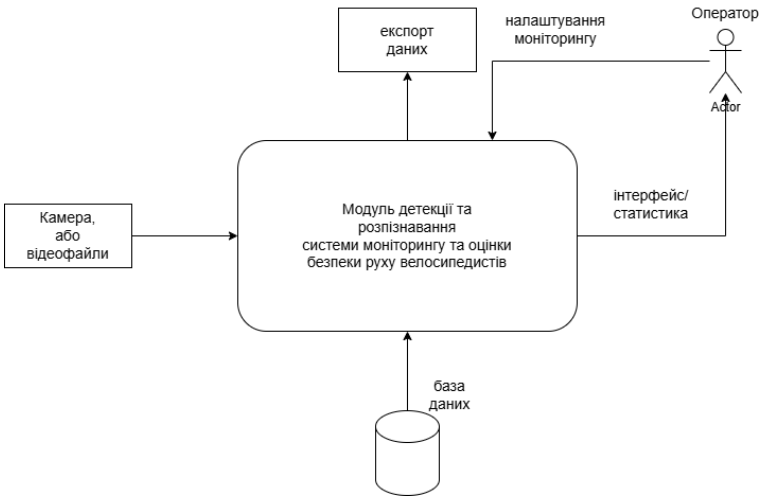


Рисунок 3.1 — Контекстна діаграма комп’ютерної системи моніторингу

Логіка взаємодії, показана на діаграмі, узгоджує ключові етапи побудови системи. Спочатку забезпечується приймання відеоданих (підключення камери або завантаження файлів) і передача їх до модуля обробки, де виконуються основні операції аналізу кадрів, детекції об’єктів і формування підсумкових результатів. Далі результати зберігаються у базі даних, що дає змогу вести історію спостережень, накопичувати статистику та підтримувати подальший аналіз. Через інтерфейс оператор отримує доступ до поточних даних і зведених показників, а також виконує налаштування моніторингу (параметри роботи, правила фіксації подій, вибір джерела тощо). Завершальним елементом взаємодії є експорт даних, який забезпечує формування звітних матеріалів або передавання накопичених даних до інших засобів обробки та документування. Наступним кроком буде розробка архітектури.

Архітектура відображає систему як єдиний програмно-апаратний комплекс та визначає її межі через перелік зовнішніх сутностей і основних інформаційних потоків. У центрі схеми розташовано основний функціональний блок — комп’ютерна система моніторингу та оцінки безпеки руху велосипедистів, яка приймає вхідні мультимедійні дані, виконує їх аналіз і формує результати для користувача та накопичення. Вхідним джерелом виступає камера або відеофайли, що забезпечують надходження відеопотоку/кадрів до системи. На схемі цей потік подано як основний канал введення: він задає первинні дані, з якими працює

рішення, та визначає вимоги до пропускної здатності, стабільності й безперервності обробки. На виході система формує результат аналізу, який може бути представлений як інформація про виявлених велосипедистів, наявність/відсутність шолома, а також супутні показники (підрахунок випадків, часові мітки, узагальнені метрики за інтервали).

Окремим важливим елементом архітектури є взаємодія з оператором, який виконує роль користувача та контролює процес моніторингу. Зі сторони оператора до системи надходить потік “налаштування моніторингу”: вибір джерела відео, параметри обробки (наприклад, пороги впевненості, частота аналізу, правила фіксації подій), а також режими роботи (запуск/зупинка, вибір сценарію спостереження). У зворотному напрямку система повертає оператору потік “інтерфейс/статистика”, який містить візуалізований результат (поточний стан, підсвічування об’єктів, лічильники) та узагальнені дані для прийняття рішень. Така двостороння взаємодія визначає вимоги до зручності інтерфейсу, оперативності оновлення інформації та прозорості представлення результатів.

Для забезпечення накопичення та подальшої обробки результатів в архітектурі виділено базу даних як зовнішній (відносно основного процесу обробки відео) компонент зберігання. До БД система записує зафіксовані події та службову інформацію: часові мітки, ідентифікатори джерела/камери, результати класифікації “у шоломі/без шолома”, лічильники та агреговані показники. Наявність такого сховища робить можливим формування статистики не лише в реальному часі, а й за історичний період, що важливо для аналітики та створення звітів. Паралельно архітектура передбачає блок “експорт даних”, який отримує інформацію з бази даних або підготовлених аналітичних вибірок і формує вихідні матеріали у форматі, придатному для подальшого використання (наприклад, таблиці, підсумкові показники, звітні набори даних). У сукупності ці елементи показують завершений цикл взаємодії: система приймає відеодані, керується налаштуваннями оператора, забезпечує відображення результатів, накопичує дані у БД та надає механізм експорту для зовнішнього використання. Зовнішній вигляд архітектури зображено у Додатку Е.

Структурна схема у вигляді UML-діаграми компонентів відображає систему як набір взаємопов’язаних програмних модулів, кожен з яких виконує окрему функцію та обмінюється даними через визначені інтерфейси. Зовнішніми сутностями виступають джерела відеоданих (камера або відеофайл) та оператор, який здійснює керування і контролює результати. Вхідний відеопотік надходить до компонента захоплення кадрів, де виконується декодування та формування послідовності кадрів для подальшої обробки. Далі кадри передаються до модуля попередньої обробки, який приводить їх до формату, необхідного для нейромережевої моделі. Після підготовки даних модуль інференсу YOLOv8 виконує детекцію та формує первинні результати у вигляді набору об’єктів з координатами обмежувальних рамок, класом і значенням упевненості. Отримані детекції надходять до компонента відстеження, який поєднує спрацювання між кадрами, присвоює ідентифікатори та формує стабільні траєкторії, що використовуються модулем логіки для визначення подій порушень і підрахунку показників.

Підсистема зберігання й представлення результатів у структурній схемі виділена окремими компонентами, що забезпечують запис подій, доступ до даних, аналітику та взаємодію з користувачем. Події, сформовані модулем логіки, надходять до компонента логування, який виконує їх збереження у базі даних через окремий шар доступу до сховища, що зменшує залежність прикладної логіки від конкретної реалізації БД. На основі накопичених даних модуль аналітики формує зведені метрики, статистику та візуалізації, які відображаються оператору через інтерфейс користувача. Окремо передбачено компонент експорту, що забезпечує вивантаження результатів у зовнішні файли або звітні набори даних. Керування режимами роботи системи реалізується через модуль налаштувань: оператор задає параметри моніторингу в інтерфейсі, після чого вони застосовуються до ключових компонентів обробки (захоплення, детекції та формування подій). У сукупності структурна UML-схема фіксує повний цикл: надходження відеоданих → детекція та відстеження → формування подій → збереження й аналітика → відображення та експорт результатів. Зовнішній вигляд структурної схеми зображено у Додатку Ж.

3.2 Вимоги до апаратної та програмної частини системи

Функціонування комп’ютерної системи моніторингу та оцінки безпеки руху велосипедистів потребує узгоджених вимог до апаратної та програмної складових. Апаратна частина забезпечує отримання відеоданих у придатній якості та їх стабільну передачу на обчислювальний вузол, де виконується нейромережевий аналіз. Програмна частина відповідає за обробку відеопотоку, виконання інференсу моделі, формування подій, зберігання результатів і відображення інформації оператору. Вимоги сформульовані таким чином, щоб система могла працювати як у режимі реального часу (за наявності відповідних ресурсів), так і в режимі обробки записів з камер.

Основним джерелом вхідних даних є камера відеоспостереження, що встановлюється у зоні контролю (велодоріжка, перехрестя, під’їзди до пішохідних переходів тощо). Для забезпечення коректної детекції бажано використовувати камеру з роздільною здатністю не нижче Full HD (1920×1080) та частотою кадрів 25–30 fps, що дозволяє стабільно фіксувати об’єкти в русі без значного змазування. Важливою є підтримка режиму WDR (компенсація різкої зміни освітлення), а також достатня світлочутливість або наявність нічного режиму, оскільки умови спостереження можуть змінюватися протягом доби. Рекомендовано використовувати камери з мережевим інтерфейсом Ethernet та підтримкою передачі потоку по RTSP/ONVIF, що забезпечує стандартизоване підключення до програмного модуля захоплення відео. Оптика (фокусна відстань) підбирається залежно від відстані до зони контролю: кадр має містити велосипедиста в масштабі, достатньому для розпізнавання шолома.

Для підвищення якості збору даних та стабільності роботи система може доповнюватися простими датчиками або тригерами, які уточнюють умови реєстрації подій. Наприклад, датчик освітленості дозволяє автоматично перемикати режими зйомки (денний/нічний) або коригувати параметри попередньої обробки кадрів. Датчики руху (PIR) або інші тригери можуть застосовуватися для запуску запису/аналізу лише при появі активності в зоні

контролю, що зменшує навантаження на сервер і обсяг збережених даних. Такі датчики не є обов’язковими для роботи нейромережевого модуля, однак можуть бути корисними в сценаріях, де важлива енергоефективність або обмежені канали передачі.

Обчислювальний вузол (сервер/робоча станція) виконує декодування відеопотоку, підготовку кадрів та інференс нейромережевої моделі. Архітектурно сервер містить: центральний процесор (CPU), оперативну пам’ять (RAM), накопичувач для зберігання даних та, за можливості, графічний прискорювач (GPU) для пришвидшення роботи моделі. Мінімальна конфігурація для тестового режиму або обробки відеофайлів може включати багатоядерний CPU та достатній обсяг RAM для буферизації кадрів; однак для роботи в реальному часі або при підключенні кількох камер бажано використовувати сервер із GPU (NVIDIA CUDA), оскільки інференс YOLOv8 істотно виграє від апаратного прискорення. Накопичувач рекомендується типу SSD, оскільки зберігання результатів, журналів і проміжних артефактів потребує швидкого доступу та стабільної пропускної здатності. Мережевий інтерфейс сервера має забезпечувати приймання потоків від камер (Ethernet 1 Gbps або вище у разі кількох камер), а також доступ оператора до інтерфейсу системи.

Програмна частина реалізується у вигляді застосунку, що працює під керуванням сучасної ОС (Windows або Linux). Для виконання модулів комп’ютерного зору використовується Python-середовище, яке забезпечує гнучку інтеграцію бібліотек машинного навчання та обробки зображень. Для роботи з відеопотоками застосовується OpenCV, а для запуску та використання моделі — фреймворк PyTorch і бібліотека Ultralytics YOLOv8. Такий стек дозволяє виконувати як навчання/доонавчання, так і інференс на готових вагах.

ПЗ повинно містити модуль захоплення потоку (підключення RTSP/файлу, декодування, виділення кадрів), модуль попередньої обробки (масштабування, перетворення), модуль інференсу YOLOv8, а також модулі постобробки (агрегація результатів, формування подій, підрахунок показників). Для накопичення результатів передбачається використання бази даних (локальної або серверної), що

зберігає події та метадані (час, джерело, клас, значення впевненості, статистика). Інтерфейс користувача має забезпечувати перегляд поточного стану, статистики та, за потреби, ініціювання експорту даних. У програмних вимогах також доцільно передбачити механізм логування та конфігурації (файли налаштувань, параметри запуску), щоб забезпечити відтворюваність експериментів і керованість системи.

Система повинна обробляти відеодані стабільно та без критичних затримок; допустимі значення затримки залежать від режиму роботи (онлайн/офлайн), проте програмні модулі мають забезпечувати ефективне використання ресурсів і можливість масштабування (перехід на продуктивніший сервер або використання GPU). Також важливо забезпечити сумісність із типовими форматами відео (MP4/AVI) та мережевими протоколами камер (RTSP/ONVIF), щоб підключення джерел даних не вимагало суттєвих змін у програмній частині.

3.3 Обґрунтування вибору програмних засобів для реалізації роботи системи

Вибір Python зумовлений його зрілістю як платформи для наукових обчислень і комп’ютерного зору: мова має низький поріг входу, велику екосистему бібліотек (NumPy, Pandas, Matplotlib), інтеграцію з апаратним прискоренням через CUDA та багатий інструментарій для експериментів (Jupyter/Colab). На практиці це дає короткий цикл «ідея → прототип → експеримент», що критично для дослідження архітектур і гіперпараметрів у задачі виявлення шоломів. Крім того, Python спрощує побудову утиліт підготовки даних (конвертація VOC→YOLO, перевірка анотацій), сценаріїв автоматизації тренувань і написання сервісних компонентів (логування, моніторинг, валідація), забезпечуючи відтворюваність дослідження.

Як основний фреймворк глибинного навчання обрано PyTorch [2], оскільки він пропонує динамічну обчислювальну графіку, прозору налагоджуваність і безшовну роботу з GPU (CUDA/cuDNN), що спрощує профілювання та оптимізацію моделі під заданий FPS. PyTorch підтримує зрілий стек деплою — TorchScript, ONNX і подальший прискорений інференс через ONNX

Runtime/TensorRT, що дозволяє переносити модель між середовищами (ноутбук, сервер, edge) без переписування коду. Крім того, велика спільнота й кількість перевірених практиками рішень (пакети аугментацій, трекінгу, оптимізаторів) знижують інженерні ризики та пришвидшують імплементацію виробничих функцій.

Разом з PyTorch використано Ultralytics YOLO як готовий високорівневий інструмент для детекції: він надає уніфікований CLI/API для тренування/валідації/експорту, підтримує сучасні one-stage архітектури (yolov8n/s/m) і «з коробки» містить сильні аугментації, метрики (mAP/PR/F1), графіки навчання, конф'южн-матрицю. Це мінімізує «клейовий» код і помилки інтеграції, дозволяючи зосередитися на якості даних і підборі порогів. Завдяки ультралайт-варіантам моделей досягається компроміс між швидкодією на споживчому GPU (GTX 1650) та достатньою точністю; стандартні сценарії експорту (ONNX) спрощують подальше розгортання сервісу.

Компонент OpenCV обрано як дефакто-стандарт для роботи з відеопотоками (RTSP/HTTP), попередньої обробки кадрів (масштабування, нормалізація, фільтри) і візуалізації результатів (оверлеї рамок/підписів) у реальному часі. OpenCV дає контроль над буферами, каденсом кадрів, багатопоточністю, що важливо для стабільності FPS і низької латентності. У поєднанні з трекерами (BYTETrack/DeepSORT) на Python забезпечується агрегація детекцій у треки та підрахунок унікальних «проходів», а отже — достовірна експлуатаційна статистика «з шоломом/без шолома».

Обраний стек також підтримує добрі практики інженерії: фіксацію середовища (conda env/requirements), керування експериментами (версіонування датасетів, seed-и, збереження ваг і артефактів у runs/…), швидке створення допоміжних інтерфейсів (наприклад, Streamlit/Gradio для демо-панелей) і CI-процедур (перевірки форматів даних, smoke-тести інференсу). Серед ризиків — чутливість до якості відеоджерел і дисбалансу класів; вони пом'якшуються доменними аугментаціями, трек-орієнтованою агрегацією, регулярною перекалібровкою порогів conf/NMS, а також можливістю масштабувати модель від

yolov8n до yolov8s/m. У сумі Python + PyTorch + Ultralytics + OpenCV надають збалансований набір засобів для швидкого R&D, відтворюваного навчання і стабільного реального часу, що безпосередньо відповідає вимогам системи моніторингу безпеки велосипедистів.

3.4 Реалізація модуля детекції та інтерфейсу YOLOv8, постобробка результатів

Модуль детекції реалізовано як окремий сервіс на Python із простим інтерфейсом: методи load_model(path), infer(image|frame, conf, iou), draw(results) і save(outputs_dir). Завантаження ваг (best.pt) відбувається один раз під час ініціалізації, далі виклики інференсу працюють без додаткових накладних витрат. Архітектуру модуля та його точки інтеграції з відеопотоком показано на рис. .

Взаємодія з Ultralytics YOLO здійснюється через офіційний API: from ultralytics import YOLO; model = YOLO(weights). Параметри інференсу (conf, iou, imgsz) передаються явно, що дозволяє швидко калібрувати робочу точку для різних камер. Скрін налаштувань інференсу та приклад виклику API наведено на рис. 3 (взяти з фрагментів коду у проєкті).

Вхідні дані підтримують три джерела: шлях до зображення, шлях до відео, або дескриптор кадру з OpenCV (numpy.ndarray). На етапі препроцесингу виконуємо масштабування до фіксованого imgsz і нормалізацію. Приклад обробки кадру “до/після” препроцесингу показано на рис. .

Результат інференсу — структура з масивами рамок xyxy, класів cls, впевненостей conf і імен класів names. Для зручності створено перетворювач у власний формат подій: Detection(xyxy, cls_name, score, camera_id, ts). Схему структури даних (класів і полів) зображено на рис. .

Постобробка включає фільтрацію за порогом conf, застосування NMS (class-agnostic для щільних сцен) і, за потреби, об’єднання дуже близьких рамок. Після фільтрації виконується сортування за score і відсікання «хвоста» дрібних детекцій, що не впливають на рішення. Вплив порогів на кількість детекцій наведено на рис. 3.2.

Рисунок 3.2 — Вплив порогів на кількість детекцій

Для інтерфейсу виводу використано утиліту візуалізації: рамка кольором за класом (синій — With Helmet, сизий — Without Helmet), підпис label = f"{name} {conf:.2f}". Малювання виконується через cv2.rectangle та cv2.putText. Знімки екрана з оверлеями для кількох сцен показано на рис 3.3 .

Рисунок 3.3 — Знімки екрана з оверлеями

У режимі відео модуль повертає також службову статистику: час препроцесингу, інференсу й постобробки на кадр (мс), оцінку FPS і використання VRAM. Ці значення логуються кожні N кадрів у SQLite для подальшого аналізу продуктивності. зовнішній вигляд бд розміщено на рис.3.4.

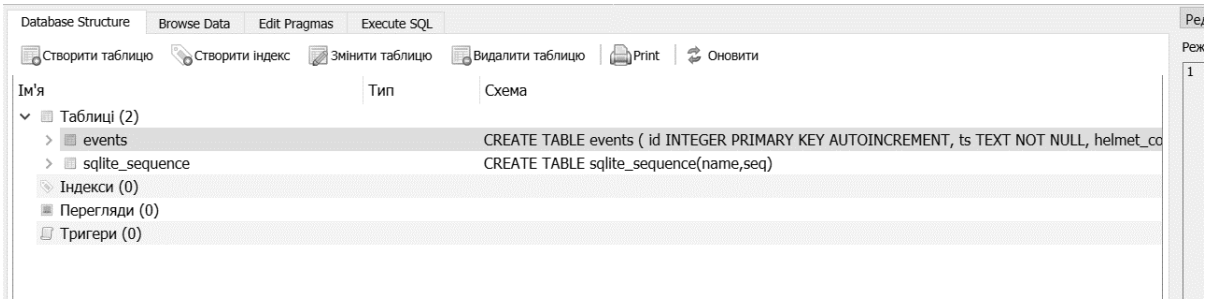


Рисунок 3.4 — Структура бази даних результатів

3.5 Програмна реалізація модуля моніторингу відеопотоку в реальному часі: підрахунок порушень і сповіщення

Модуль моніторингу відеопотоку реалізовано як окремий Python-скрипт, який працює в режимі реального часу з використанням вебкамери ноутбука. Захоплення кадрів здійснюється за допомогою бібліотеки OpenCV (клас VideoCapture), після чого кожен кадр з інтервалом приблизно 0,5 секунди передається на вхід детектору YOLOv8 із завантаженими вагами моделі шолома (best.pt). Такий підхід дає змогу поєднати відносно стабільний FPS від камери з контрольованим навантаженням на процесор/відеокарту, оскільки інференс запускається не для кожного кадру, а з фіксованим часовим кроком. Зовнішній вигляд модуля моніторингу зображено на рис. 3.5.

Рисунок 3.5 — Записи в базі даних з результатів

На етапі обробки кадру модель YOLOv8 повертає набір обмежувальних прямокутників (bounding boxes) із класами With Helmet та Without Helmet та значеннями впевненості (confidence). Ці прямокутники накладаються на зображення: для класу з наявністю шолома використовуються зелені рамки й підпис «Helmet», для відсутності шолома – червоні рамки й підпис «No helmet». Таким чином оператор отримує візуальне підтвердження того, як саме система інтерпретує кожного велосипедиста в кадрі. Додатково у верхній частині екрана виводиться службовий текст із поточними налаштуваннями інтервалу обробки. Приклад прямокутника зображено на рис. 3.6.

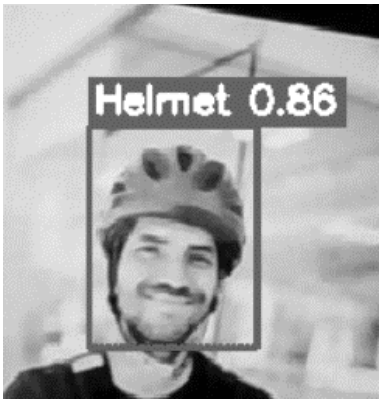


Рисунок 3.5 — Обмежувальний прямокутник

Ключовою функцією модуля моніторингу є автоматичний підрахунок кількості спрацювань із шоломом та без шолома. Після кожного запуску інференсу (раз на ~0,5 с) скрипт проходить по всіх знайдених об’єктах і підраховує, скільки з них належать до класу With Helmet і скільки – до Without Helmet. Результати підрахунку відображаються у вигляді двох типів лічильників: «Frame: Helmet = …, No helmet = …» (кількість об’єктів у поточному кадрі) та «Total: Helmet = …, No helmet = …» (накопичувальна статистика з моменту запуску програми). Такий інтерфейс дозволяє оператору в режимі реального часу бачити як миттєву ситуацію, так і загальну кількість порушень за зміну. Інтерфейс зображено на рис. 3.6.

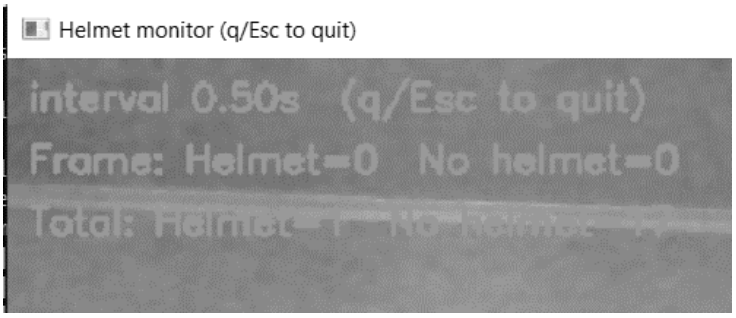
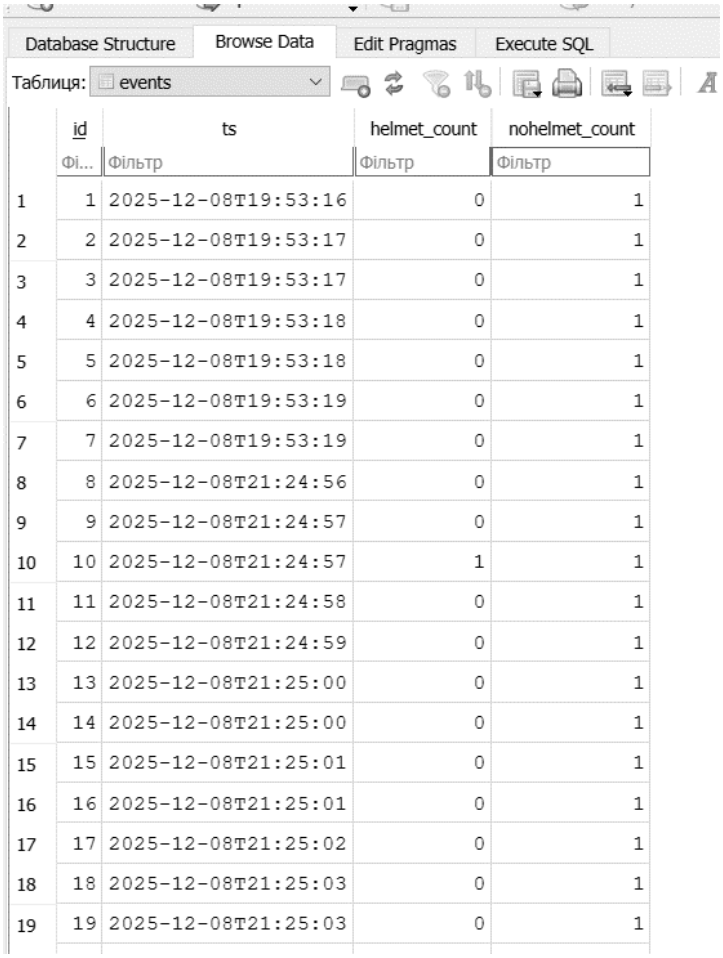


Рисунок 3.6 — Інтерфейс підрахуноку спрацювань

Для збереження статистики роботи системи й подальшого аналізу модуль моніторингу інтегровано з простою базою даних на основі SQLite. База зберігається у файлі helmet_events.db у робочому каталозі проєкту й створюється автоматично при першому запуску програми. У таблиці events для кожного інтервалу, коли в кадрі зафіксовано хоча б одного велосипедиста, записується дата й час події (ts), кількість детекцій з шоломом (helmet_count) та кількість детекцій без шолома (nohelmet_count). У такий спосіб формується подієвий журнал, який відображає «потік спрацювань» системи з прив’язкою до часу. Вигляд записів показано на рис. 3.7.



Database Structure | Browse Data | Edit Pragmas | Execute SQL

Таблиця: events

	id	ts	helmet_count	nohelmet_count
	Фі...	Фільтр	Фільтр	Фільтр
1	1	2025-12-08T19:53:16	0	1
2	2	2025-12-08T19:53:17	0	1
3	3	2025-12-08T19:53:17	0	1
4	4	2025-12-08T19:53:18	0	1
5	5	2025-12-08T19:53:18	0	1
6	6	2025-12-08T19:53:19	0	1
7	7	2025-12-08T19:53:19	0	1
8	8	2025-12-08T21:24:56	0	1
9	9	2025-12-08T21:24:57	0	1
10	10	2025-12-08T21:24:57	1	1
11	11	2025-12-08T21:24:58	0	1
12	12	2025-12-08T21:24:59	0	1
13	13	2025-12-08T21:25:00	0	1
14	14	2025-12-08T21:25:00	0	1
15	15	2025-12-08T21:25:01	0	1
16	16	2025-12-08T21:25:01	0	1
17	17	2025-12-08T21:25:02	0	1
18	18	2025-12-08T21:25:03	0	1
19	19	2025-12-08T21:25:03	0	1

Рисунок 3.7 — Вигляд записів у базі даних

Функцію «сповіщення» в рамках даного прототипу реалізовано у вигляді візуальних індикаторів на екрані оператора. Зокрема, у статусному рядку підсумковий стан кадру (наприклад, переважають об’єкти без шолома) може відображатися іншим кольором, а загальна кількість порушень (клас Without Helmet) зростає у реальному часі. Такий підхід є достатнім для лабораторного прототипу і дозволяє наочно демонструвати роботу системи на захисті магістерської роботи. У перспективі цей механізм може бути доповнений звуковими сигналами, push-сповіщеннями або інтеграцією з зовнішніми інформаційними системами (наприклад, диспетчерською панеллю міського центру безпеки).

Загалом модуль моніторингу відеопотоку поєднує декілька важливих функцій: захоплення й попередню обробку відео, виклик детектора шоломів, підрахунок кількості спрацювань у реальному часі та логування результатів до бази

даних. Завдяки цьому він виступає центральною ланкою між нейромережевою моделлю та інтерфейсом оператора, забезпечуючи не лише візуальне відображення роботи системи, а й збирання інформації, необхідної для подальшої аналітики й оцінки ефективності контролю безпеки руху велосипедистів.

3.6 Розробка інтерфейс користувача

Для накопичення історичних даних про роботу системи використано вбудовану в Python СУБД SQLite, яка зберігає всю інформацію у вигляді одного файлу helmet_events.db в робочому каталозі. При першому запуску програми створюється таблиця events, що містить ідентифікатор запису, штамп часу (ts) та два числові поля: кількість об’єктів з шоломом (helmet_count) і кількість об’єктів без шолома (nohelmet_count) у відповідному інтервалі. Таким чином формується подієвий журнал роботи системи, де кожен запис відповідає одному «кроці» аналізу відеопотоку.

Використання SQLite як сховища має кілька важливих переваг для прототипу. По-перше, база даних не потребує окремого серверного ПЗ – у ролі СУБД виступає стандартний модуль sqlite3 мови Python, що включений до складу інтерпретатора. По-друге, структура таблиці подій є мінімалістичною, але розширюваною: у разі розгортання системи в реальних умовах до неї можна додати поля з ідентифікатором камери, географічною локацією, рівнем ризику, типом дороги тощо. По-третє, формат SQLite добре підтримується сторонніми інструментами (наприклад, DB Browser for SQLite), що спрощує налагодження та ручну перевірку коректності записів.

Окремим компонентом програмної реалізації є модуль аналітики, який виконує агрегацію записів з таблиці events та формує для оператора або адміністратора системи узагальнені статистичні звіти. В аналітичному скрипті передбачено щонайменше два режими роботи: аналіз за поточну добу і аналіз за весь час. У першому режимі програма відбирає всі записи з поточною датою, групує їх по годинах та обчислює сумарну кількість спрацювань з шоломом і без шолома для кожної години. У другому режимі проводиться групування за датою,

що дозволяє будувати довгострокову статистику порушень по днях. Приклад графіків за сьогодні та весь час зображено на рис 3.8 та 3.9 відповідно.

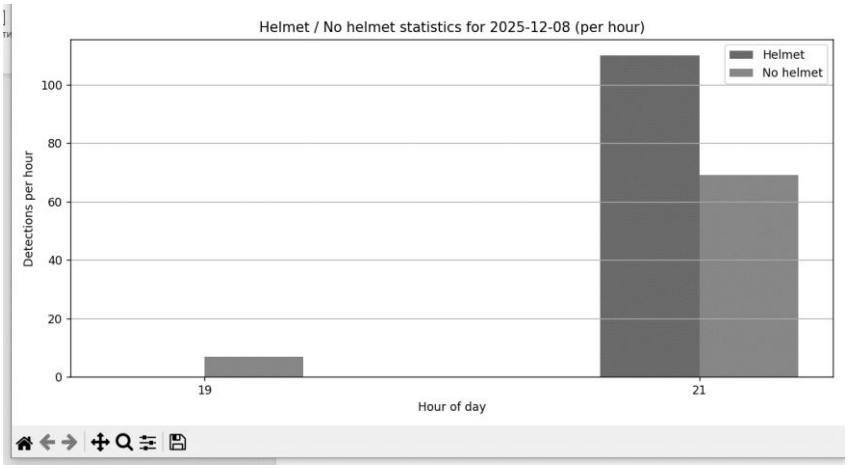


Рисунок 3.8 — Графік за сьогоднішній день

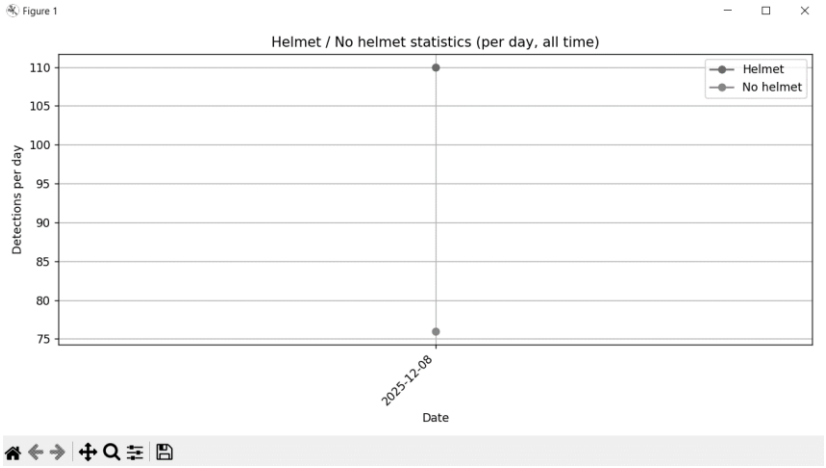


Рисунок 3.9 — Графік за весь час

Інтерфейс користувача орієнтований на оперативність. Головний екран складається з плиток камер (живе прев’ю або останній кадр), лічильників з шоломом / без шолома за вибраний період, індикаторів технічного стану (FPS, VRAM/CPU, черги кадрів) і стрічки сповіщень. Фільтри дозволяють обрати камери, період, пороги conf/NMS, а також режим відображення: «проїзди» (за треками) чи «детекції» (покадрово для діагностики). Клік по камері відкриває деталейну сторінку з графіками, списком треків і кнопкою переходу до відеофрагмента.

Передбачено ролі: оператор (моніторинг/сповіщення) і аналітик (експорт/порівняння релізів/налаштування порогів).

Далі подано загальну схему обробки відеопотоку в розробленій комп'ютерній системі моніторингу та оцінки безпеки руху велосипедистів. Відеодані надходять із джерел різного типу – вуличних IP-камер, вебкамери або попередньо записаних відеофайлів – і через модуль захоплення відео на основі бібліотеки OpenCV перетворюються на послідовність кадрів. Кожен кадр передається до модуля детекції шоломів, де попередньо навчена нейронна мережа YOLOv8 виконує пошук велосипедистів і класифікацію їх стану за ознакою наявності чи відсутності шолома. Результати детекції потрапляють до модуля трекінгу та обліку порушень, у якому формуються треки об'єктів, обчислюються кількість і тривалість перебування велосипедистів без шолома в контрольній зоні, а також генеруються події моніторингу. На основі цих даних система одночасно будує відеопотік із візуальною розміткою (рамки, класи, лічильники) для онлайн-перегляду оператором та записує події і агреговані показники до бази даних, що використовується для подальшого аналізу та відображення статистики у веб-дашборді.

Інтерфейс користувача розробленого програмного модуля реалізовано у вигляді одновіконного застосунку, що забезпечує одночасний контроль відеопотоку та перегляд поточних результатів роботи алгоритму. У лівій частині вікна відображається живе відео з камери з накладеними результатами детекції: для кожного знайденого об'єкта виводяться обмежувальні рамки та підписи класу ("Helmet" / "No helmet") із значенням довіри. Такий підхід дозволяє оператору в реальному часі оцінювати коректність розпізнавання та швидко виявляти помилкові спрацювання.

Права панель інтерфейсу містить елементи керування та блоки оперативної аналітики. Через кнопки Start/Stop оператор запускає або зупиняє спостереження, Reset використовується для обнулення лічильників сесії, а Snapshot — для збереження поточного кадру як доказової інформації або прикладу для подальшого аналізу. Додатково передбачено налаштування інтервалу обробки кадрів, що дає

змогу збалансувати навантаження на обчислювальні ресурси та швидкодію системи.

Окремим компонентом інтерфейсу є модуль статистики та журналювання подій. У вікні відображаються лічильники для поточного кадру та сумарні показники за сесію (кількість виявлень у шоломі та без шолома), що дозволяє швидко оцінити частоту порушень. Для забезпечення відтворюваності результатів та подальшого аналізу реалізовано журнал подій, який зберігає час фіксації та кількість спрацювань у базі даних і відображається у вигляді таблиці останніх записів. За потреби дані журналу можуть експортуватися у CSV для формування звітів і проведення розширеної статистичної обробки. Зовнішній вигляд інтерфейсу зображено на рис. 3.10.

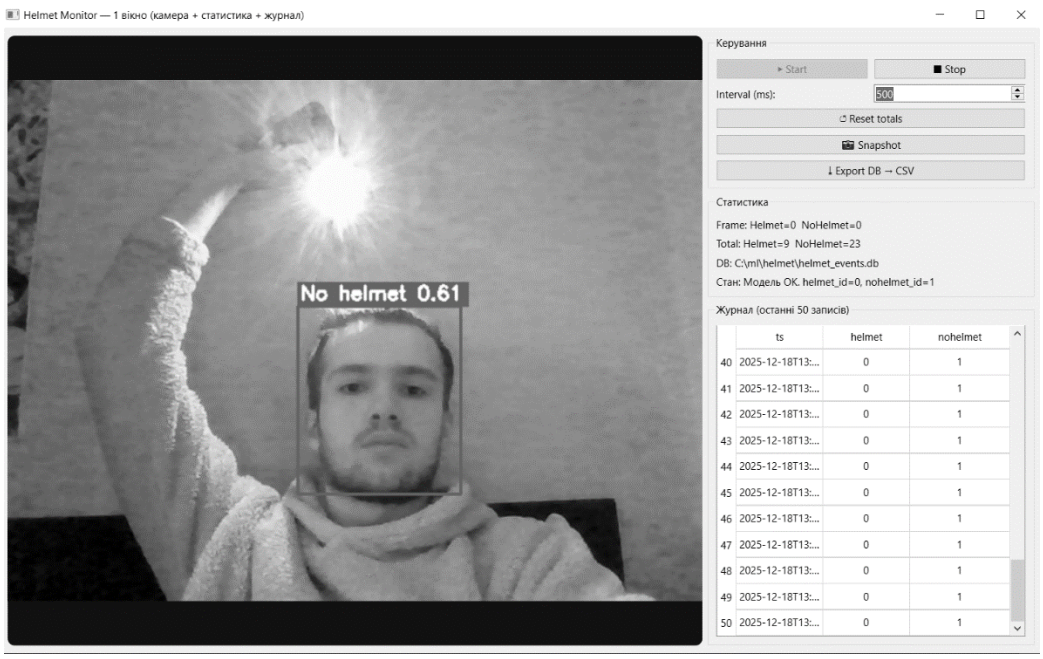


Рисунок 3.10 — Інтерфейс користувача

Усі зміни конфігурацій та ручні дії з треками пишуться в audit-лог. Для стабільності роботи інтерфейс візуально позначає деградацію (падіння FPS, зростання латентності, збій потоку RTSP) і автоматично створює задачі на технічну перевірку. Такий комплекс логування, аналітики та UI забезпечує прозору експлуатацію системи, швидкий пошук причин відхилень і коректне прийняття управлінських рішень на основі даних.

4 ПЕРЕВІРКА ПРАЦЕЗДАТНОСТІ ТА ТЕСТУВАННЯ

У цьому розділі наведено методику й результати перевірки працездатності та тестування системи: описано тестове середовище (апаратна конфігурація, версії бібліотек), склад і підготовку контрольних піднаборів (val/test, «складні сцени»: ніч, дощ, далекі плани, оклюзії), протокол вимірювань (mAP@50/50–95, Precision/Recall/F1, PR-криві, confusion matrix) та критерії приймання. Окремо оцінено роботу в реальному часі: FPS, латентність кадру, стабільність при зміні освітлення/трафіку, а також коректність підрахунку «проїздів» за треками. Для валідації обраних порогів і налаштувань проведено порівняння з базовими конфігураціями (yolov8n vs yolov8s, різні conf/NMS/IoU) і простежено вплив аугментацій та донавчання на якість. Усі експерименти відтворювані: зафіксовано seed-и, версії, конфігурації, а ключові артефакти (графіки, таблиці, приклади детекцій) включено до звіту для подальшого аналізу й обґрунтування висновків.

4.1 Тестування моделі роботи системи

Тестування розробленої системи проводилося у два взаємопов’язані етапи: по-перше, оцінювалася якість самої моделі YOLOv8 на відкладених вибірках зображень, а по-друге, перевірялася робота інтегрованого програмного модуля детекції шоломів у режимі реального часу. Такий підхід дозволяє окремо проаналізувати здатність нейромережі правильно класифікувати зображення «з шоломом» і «без шолома» та перевірити, як ця модель поводиться в реальних умовах експлуатації при роботі з потоком кадрів від вебкамери та подальшою обробкою результатів.

На першому етапі тестування використовувалися валідаційні та тестові підвибірки з основного набору Helmet Detection (Kaggle) та додаткового набору Bike Helmet Detection (Roboflow), що не брали участі у навчанні моделі. Для цих зображень YOLOv8 генерувала детекції шолома, після чого за стандартними інструментами Ultralytics розраховувалися метричні показники моделі: точність (precision), повнота (recall), середня точність за різних порогів IoU (mAP) для обох

класів. Окрему увагу приділяли саме класу «без шолома», оскільки він є критичним для виявлення порушень правил безпеки руху велосипедистів.

Для глибшого розуміння помилок моделі була побудована матриця помилок (confusion matrix) на відкладеній вибірці. Аналіз цієї матриці дозволив оцінити, як часто модель плутає класи «With Helmet» та «Without Helmet», які типові ситуації призводять до хибних позитивних (false positive) і хибних негативних (false negative) спрацювань. Зокрема, перевірялися випадки, коли шолом частково закритий рюкзаком, капюшоном або іншими об’єктами, а також сцени з нестандартними ракурсами зйомки. Матриця помилок зображена на рис 4.1.

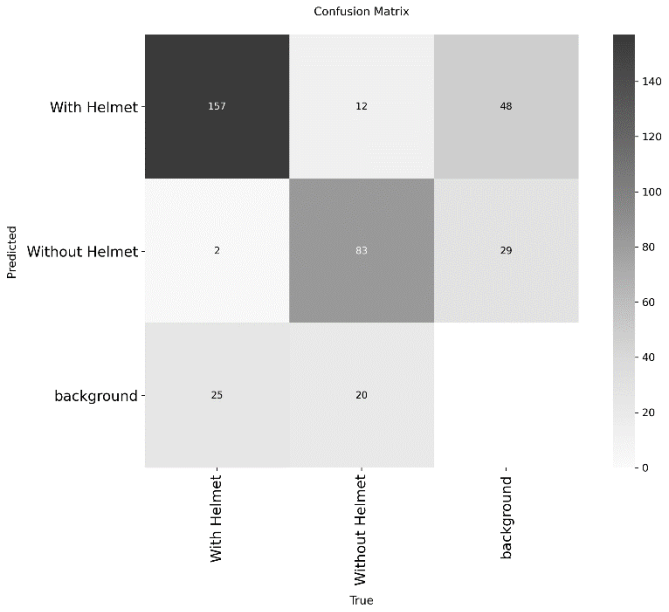


Рисунок 4.1 — Матриця помилок для моделі

На матриці помилок видно, що модель загалом добре розрізняє наявність і відсутність шолома, але має кілька характерних типів помилок. Для класу «With Helmet» більшість об’єктів розпізнаються правильно (157 випадків), однак частина прикладів із шоломом помилково відноситься до класу «background» (25) або навіть до «Without Helmet» (2). Аналогічно для класу «Without Helmet» переважають правильні передбачення (83), але є помилки як у бік «With Helmet» (12), так і у бік фону (20).

Найбільше хибних спрацювань спостерігається між класами «об’єкт» та

«background»: модель іноді не детектує велосипедиста, а приклади йдуть у фон, а іноді, навпаки, приймає елементи фону за шолом (48 випадків) або за обличчя без шолома (29). Це свідчить, що подальше покращення якості можливо за рахунок розширення вибірки саме «складними» фоновими сценами та прикладами з частковими перекриттями, а також за рахунок додаткової аугментації, спрямованої на відокремлення велосипедиста від навколишнього середовища.

Наступним кроком було якісне тестування детекцій на окремих прикладах із тестових наборів. Для цього згенеровані за допомогою YOLOv8 зображення з накладеними рамками (bounding boxes) та підписами класів аналізувалися вручну: перевірялося, чи коректно модель відрізняє шолом від волосся, шапки або капюшона, як вона поводиться на групових сценах з кількома велосипедистами в кадрі, а також у випадках зі складним фоном. На основі цих прикладів формувався набір ілюстрацій, що демонструють як правильні спрацьовування, так і характерні помилки моделі.

На рисунку 4.2 наведено приклади роботи модуля детекції шоломів у реальному часі. Ліві фрагменти демонструють коректні спрацювання: у верхньому лівому випадку система правильно класифікує користувача без засобів захисту як клас «No helmet» (червона рамка), а в нижньому лівому – впевнено визначає наявність шолома на голові велосипедиста та відносить його до класу «Helmet» (зелена рамка). Одночасно оновлюються лічильники Frame і Total, що показує узгоджену роботу детектора з інтерфейсом оператора.

Праві фрагменти ілюструють типові помилки моделі. У верхньому правому прикладі рамка з написом «No helmet» накладається не на голову велосипедиста, а на фрагмент руки на передньому плані, тобто відбувається хибне спрацювання на елемент фону. У нижньому правому випадку система хибно відносить до класу «Helmet» обличчя на зображенні зі смартфона, де шолом відсутній, а в людини просто кольорове волосся. Такі приклади наочно показують, що, попри загальну працездатність детектора, він залишається чутливим до нетипових ракурсів, відображень з екрана та об’єктів, які частково нагадують шолом, що обґрунтовує необхідність подальшого розширення навчальної вибірки та донавчання моделі.

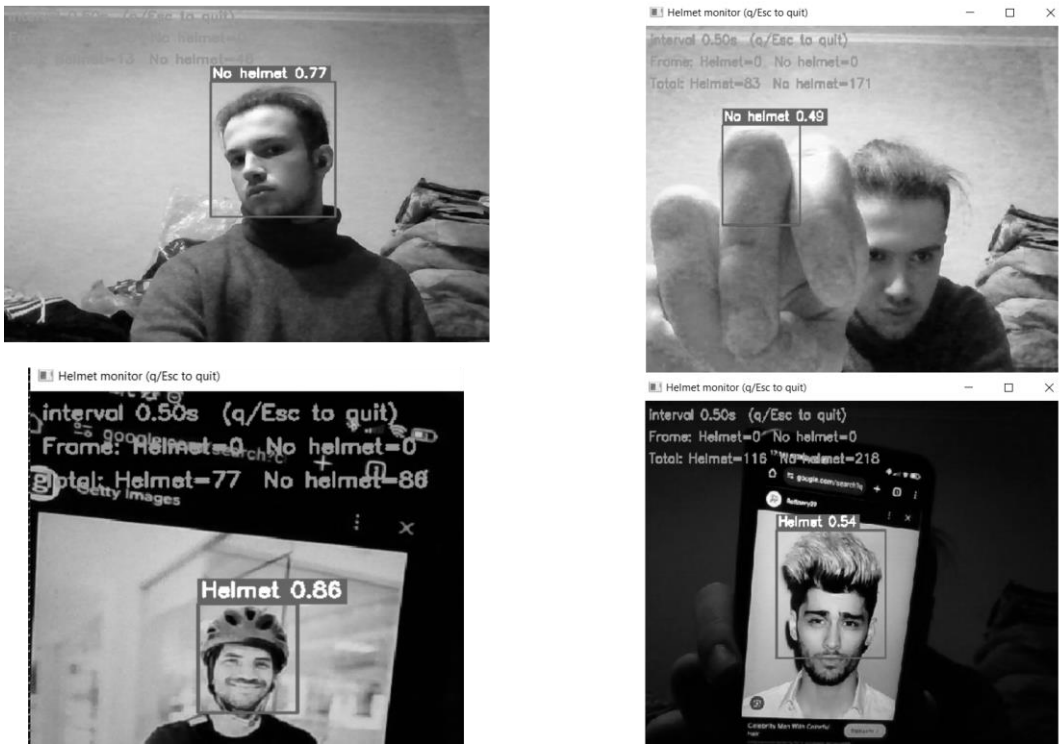


Рисунок 4.2 — Перевірка роботи модуля детекції

Другий етап тестування був спрямований на перевірку працездатності програмного модуля моніторингу відеопотоку в реальному часі. У цьому режимі вхідним джерелом слугувала вебкамера ноутбука, а результат детекції відображався безпосередньо у вікні програми: для кожного виявленого об’єкта накладалася зелена або червона рамка з підписом «Helmet» або «No helmet», відповідно до класу, який повернула модель. Паралельно перевірялася коректність роботи лічильників: кількість об’єктів у поточному кадрі (Frame: Helmet / No helmet) та накопичувальна статистика з початку сеансу (Total: Helmet / No helmet).

Окремий набір сценаріїв тестування було присвячено перевірці роботи системи в різних умовах освітлення та оточення. Зокрема, проводилися експерименти при денному та вечірньому світлі, у приміщенні з штучним освітленням, на тлі однорідних і складних фонів, з використанням змінного одягу (з капюшоном, без капюшона, у яскравих і темних куртках). Для кожного сценарію аналізувалося, як часто система правильно класифікує наявність шолома, чи виникають стабільні хибні спрацьовування на певних типах одягу або фону, та наскільки стабільною є робота лічильників у динаміці.

Паралельно з візуальним контролем роботи детектора у режимі реального часу проводилася перевірка коректності запису подій у базу даних SQLite. Для цього після серії тестових запусків зчитувався вміст таблиці events і звірявся з очікуваними результатами: кількістю спрацювань з шоломом і без шолома за відповідні часові проміжки. Окрім ручної перевірки через допоміжні утиліти на зразок DB Browser for SQLite, дані з БД додатково аналізувалися через аналітичний скрипт, який будує графіки розподілу детекцій за годинами доби та за днями. Це дозволило підтвердити, що модуль моніторингу коректно формує та передає інформацію до «центрального сервера», а модуль аналітики правильно агрегує й візуалізує накопичені дані. На рис. 4.3 зображено вигляд статистики за весь час

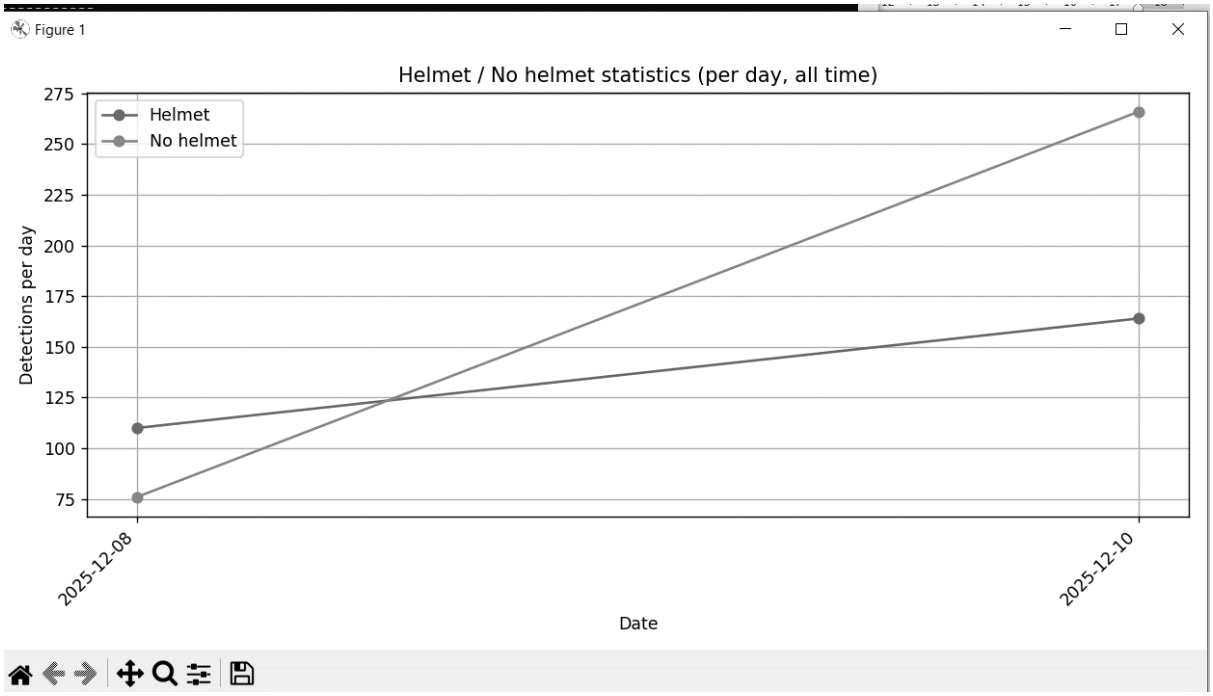


Рисунок 4.3 — Графік статистики за весь час

Узагальнюючи результати тестування, можна зробити висновок, що розроблена модель YOLOv8 і програмний модуль моніторингу забезпечують працездатність системи в режимі реального часу: система здатна виявляти велосипедистів з шоломом і без шолома на відеопотоці, відображати результати оператору та накопичувати статистику подій у базі даних для подальшого аналізу. При цьому у процесі тестування були виявлені типові ситуації, у яких модель

допускає помилки, що дозволяє сформувати вимоги до подальшого покращення якості (зокрема, за рахунок розширення навчальної вибірки новими прикладами з реальних умов експлуатації). Саме ці аспекти розглядаються детальніше у наступних підрозділах, присвячених оцінці ефективності та можливим напрямкам удосконалення системи.

4.2 Аналіз результатів тестування, оцінка ефективності та можливості для покращення системи

Результати, отримані під час тестування моделі YOLOv8 на валідаційних та тестових вибірках, а також під час роботи прототипу в режимі реального часу, підтверджують працездатність розробленої комп'ютерної системи моніторингу велосипедистів. На рівні нейромережевої моделі було показано, що детектор у більшості випадків коректно відрізняє класи «With Helmet» та «Without Helmet», що підтверджується значною кількістю правильних передбачень у діагональних комірках матриці помилок. Водночас наявні хибні спрацювання вказують на конкретні ситуації, у яких система потребує подальшого вдосконалення.

Аналіз матриці помилок показує, що найбільш проблемними є два типи помилок: по-перше, коли об'єкти класів «With Helmet» або «Without Helmet» потрапляють до класу «background», тобто система не розпізнає велосипедиста взагалі; по-друге, коли фрагменти фону (частини одягу, елементи оточення) помилково класифікуються як наявність шолома. Такі помилки є типовими для задачі детекції на складних реальних сценах і свідчать про те, що межа між об'єктом та фоном у деяких випадках є недостатньо чіткою. Саме тому важливу роль відіграє використання додаткових датасетів і аугментацій, що покривають велику кількість «нетипових» прикладів.

Порівняння результатів базової моделі та моделі після донавчання на додатковому наборі Bike Helmet Detection демонструє покращення балансу між класами. У донавченої моделі зменшується кількість випадків, коли приклади без шолома помилково потрапляють до класу «With Helmet», натомість збільшується число правильних спрацювань класу «Without Helmet». Це особливо важливо для

практичного використання системи, оскільки саме помилки типу «порушення не виявлено» (велосипедист без шолома позначений як такий, що шолом має) є критичними з точки зору безпеки. Разом із тим частина прикладів і надалі класифікується некоректно, що вказує на резерви для подальшого розширення навчальної вибірки та налаштування моделі.

Оцінка роботи системи в режимі реального часу показала, що запропонований підхід придатний для використання в умовах потокового відео. При фіксованому інтервалі інференсу модуль моніторингу стабільно обробляє кадри з вебкамери, накладає результати детекції на зображення та веде підрахунок кількості спрацювань із шоломом та без шолома. Під час експериментів із різними умовами освітлення, типами одягу та фоном було виявлено, що система впевнено працює на «чистих» сценах (добре освітлений користувач, контрастний фон), але в складніших сценаріях може давати як пропуски об’єктів, так і хибні позитивні спрацювання. Ці спостереження узгоджуються з аналізом матриці помилок і підтверджують, що загальні тенденції поведінки моделі зберігаються і в реальному відеопотоці.

Додаткову інформацію про ефективність системи надає модуль аналітики центрального сервера, який агрегує дані з бази helmet_events.db та будує графіки розподілу детекцій за годинами доби та за днями. Аналіз добових гістограм дозволяє виявити часові інтервали, у які спостерігається найбільша кількість детекцій без шолома, а лінійні графіки по днях дають змогу оцінити динаміку порушень у більш тривалій перспективі. Навіть у рамках лабораторного тестування такі графіки наочно демонструють, що система коректно накопичує статистику й може використовуватися для побудови звітів про стан дотримання вимог безпеки. У реальній експлуатації ці інструменти можуть стати основою для управлінських рішень (наприклад, визначення ділянок чи годин доби, де потрібне посилення контролю).

Загалом результати тестування свідчать, що розроблена комп’ютерна система досягає поставлених у роботі цілей: вона здатна автоматично виявляти велосипедистів з шоломом та без шолома, відображати результати оператору, а

також накопичувати та аналізувати статистику порушень. Водночас виявлені обмеження – чутливість до умов освітлення, складних фонів та нетипових ракурсів — визначають напрями подальшого розвитку. До таких напрямів належать розширення навчальної вибірки за рахунок даних з реальних міських камер, удосконалення архітектури моделі або комбінування кількох детекторів та впровадження додаткових евристик постобробки. У наступних підрозділах ці перспективи розглядаються детальніше з точки зору потенційних покращень точності та надійності системи.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Проведення комерційного та технологічного аудиту розробки комп’ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж.

Актуальність проведення комерційного та технологічного аудиту комп’ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж зумовлена зростанням ролі міської мобільності та необхідністю підвищення рівня безпеки вуличного руху. У багатьох містах збільшується кількість велосипедистів, що, з одного боку, сприяє екологічності та зменшенню транспортного навантаження, а з іншого — підвищує потребу у точному, оперативному та автоматизованому контролі ризиків. Використання нейронних мереж дозволяє суттєво покращити якість виявлення небезпечних ситуацій, аналізу траєкторій руху та прогнозування потенційних інцидентів.

Проведення технологічного аудиту такого рішення є необхідним для оцінки точності алгоритмів, стабільності роботи системи, рівня інтеграції з інфраструктурою «розумного міста» та відповідності сучасним вимогам кібербезпеки. Аудит дозволяє виявити технічні вузькі місця, оцінити ефективність навчальних вибірок, протестувати роботу системи в реальних умовах та визначити можливості масштабування. Це важливо для забезпечення високої надійності, оскільки навіть невелика похибка у системі безпеки дорожнього руху може мати серйозні наслідки.

Комерційний аудит, у свою чергу, дозволяє оцінити конкурентоспроможність технології, її ринковий потенціал, економічну доцільність впровадження та можливі моделі монетизації. Він допомагає зрозуміти, наскільки система відповідає потребам органів місцевого самоврядування, велосипедної інфраструктури та приватних операторів транспорту. Такий комплексний підхід забезпечує обґрунтованість прийняття рішень щодо подальшого розвитку проєкту, інвестицій у технологію та її масштабного впровадження для підвищення безпеки всіх учасників дорожнього

руху.

Для проведення комерційного та технологічного аудиту залучаємо 3-х незалежних експертів, якими є провідні викладачі випускової або спорідненої кафедри.

Оцінювання науково-технічного рівня комп'ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж та її комерційного потенціалу здійснюємо із застосуванням п'ятибальної системи оцінювання за 12-ма критеріями, а результати зводимо до таблиці 1.

Таблиця 5.1 — Результати оцінювання науково-технічного рівня і комерційного потенціалу комп'ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж

Критерії	Експерти		
	Експерт 1	Експерт 2	Експерт 3
	Бали, виставлені експертами		
Технічна здійсненність концепції	3	3	3
Ринкові переваги (наявність аналогів)	2	2	2
Ринкові переваги (ціна продукту)	3	3	3
Ринкові переваги (технічні властивості)	3	3	3
Ринкові переваги (експлуатаційні витрати)	3	2	3
Ринкові перспективи (розмір ринку)	2	3	2
Ринкові перспективи (конкуренція)	2	2	2
Практична здійсненність (наявність фахівців)	3	3	3
Практична здійсненність (наявність фінансів)	2	2	2
Практична здійсненність (необхідність нових матеріалів)	3	3	3
Практична здійсненність (термін реалізації)	3	3	2
Практична здійсненність (розробка документів)	3	3	3
Сума балів	32	32	31
Середньоарифметична сума балів, СБ	32		

За результатами розрахунків, наведених в таблиці 1 робимо висновок про те,

що науково-технічний рівень та комерційний потенціал комп'ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж — вищий середнього.

5.2 Розрахунок витрат на здійснення розробки комп'ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж

Належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці, також будь-які види грошових і матеріальних доплат, які належать до елемента «Витрати на оплату праці».

Витрати на основну заробітну плату дослідників ($З_о$) розраховують відповідно до посадових окладів працівників, за формулою:

$$З_о = \sum_{i=1}^{к} \frac{М_{ni} \cdot t_i}{Т_р},$$

де к — кількість посад дослідників, залучених до процесу дослідження;

$М_{ni}$ — місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.;

$Т_р$ — число робочих днів в місяці; приблизно $Т_р$ = (21…23) дні, приймаємо 22 дні;

t_i — число робочих днів роботи розробника (дослідника).

Зроблені розрахунки зводимо до таблиці 2.

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт розраховують за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i,$$

де C_i — погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i — час роботи робітника на виконання певної роботи, год.

Таблиця 5.2 — Витрати на заробітну плату дослідників

Посада	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник	28 000	1273	10	12727
Розробник	22 000	1000	60	60000
Консультанти	25 000	1136	8	9091
Всього:	81818			

Погодинну тарифну ставку робітника відповідного розряду С і можна визначити за формулою:

$$С_i = \frac{М_м \cdot К_i \cdot К_с}{Т_р \cdot t_{зм}},$$

де $М_м$ — розмір прожиткового мінімуму працездатної особи або мінімальної місячної заробітної плати (залежно від діючого законодавства), у 2025 році $М_м$=8000 грн;

$К_i$ — коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду;

$К_с$ — мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати, складає 1,1;

$Т_р$ — середня кількість робочих днів в місяці, приблизно $Т_р$ = 21…23 дні, приймаємо 22 дні; $t_{зм}$ – тривалість зміни, год., приймаємо 8 год.

Додаткова заробітна плата $З_д$ всіх розробників та робітників, які брали участь у виконанні даного етапу роботи, розраховується як (від 10 до 12)% від суми основної заробітної плати всіх розробників та робітників, тобто:

$$З_д = 0{,}1 \cdot (З_о + З_р) = 0{,}1 \cdot (81818 + 3502) = 8534 \text{ грн.}$$

Таблиця 5.3 — Витрати на заробітну плату робітників

Найменування робіт	Трудомісткість, н-год.	Розряд роботи	Погодинна тарифна ставка	Тариф. коеф.	Величина, грн.
Монтаж і налаштування відеокамери	12	3	59	1,18	708
Розгортання серверної частини та бази даних	24	4	63,5	1,27	1524
Тестування програмної системи в умовах експлуатації	20	4	63,5	1,27	1270
Всього					3502

Нарахування на заробітну плату $Н_{зп}$ розробників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою:

$$Н_{зп} = \beta \cdot (З_о + З_р + З_д) =$$

$$= 0{,}22 \cdot (81818 + 3502 + 8534) = 20647 \text{ грн.}$$

де $З_о$ — основна заробітна плата розробників, грн.;

$З_р$ — основна заробітна плата робітників, грн.;

$З_д$ — додаткова заробітна плата всіх розробників та робітників, грн.;

β — ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, % (приймаємо для 1-го класу професійності ризику 22%).

Витрати на матеріали М, що були використані під час виконання даного етапу роботи, розраховуються за формулою:

$$М = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i - \sum_{1}^{n} В_i \cdot Ц_i,$$

де $Н_i$ — кількість матеріалів і-го виду, шт.;

$Ц_i$ — ціна матеріалів і-го виду, грн.;

$К_i$ — коефіцієнт транспортних витрат,

$К_i$ = (від 1,1 до 1,15);

n — кількість видів матеріалів.

Таблиця 5.4 – Матеріали, що використані на розробку

Найменування матеріалів	Ціна за одиницю, грн.	Витрачено	Вартість витрачених матеріалів, грн.
USB-камера Full HD	2500	1	2500
Жорсткий диск 1ТБ	1800	1	1800
Мережевий кабель UTP Cat5e	300	1	300
Всього, з врахуванням коефіцієнта транспортних витрат			5060

Витрати на комплектуючі К, що були використані під час виконання даного етапу роботи, розраховуються за формулою:

$$К = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i,$$

де $Н_i$ — кількість комплектуючих і-го виду, шт.;

$Ц_i$ — ціна комплектуючих і-го виду, грн.;

$К_i$ — коефіцієнт транспортних витрат,

$К_i$ = (від 1,1 до 1,15);

n — кількість видів комплектуючих.

Таблиця 5.5 – Комплектуючі, що використані на розробку

Найменування комплектуючих	Ціна за одиницю, грн.	Витрачено	Вартість витрачених комплектуючих, грн.
Системний блок (CPU, RAM, SSD тощо)	25000	1	25000
Відеокарта NVIDIA GeForce RTX 3060	18000	1	18000
Монітор 24" Full HD	5500	1	5500
Маршрутизатор з підтримкою IP-камер	3000	1	3000
Всього, з врахуванням коефіцієнта транспортних витрат			57165

Вартість спецустаткування визначається за прейскурантом гуртових цін або за даними базових підприємств за відпускними і договірними цінами.

$$В_{спец} = \sum_{1}^{к} Ц_i \cdot С_{пр.i} \cdot К_i,$$

де $Ц_i$ — ціна придбання спецустаткування і-го виду, грн.;

$С_{пр.і}$ — кількість одиниць спецустаткування відповідного виду, шт.;

$К_і$ — коефіцієнт транспортних витрат,

$К_і$ = (з 1,1 до 1,15);

n — кількість видів спецустаткування.

Таблиця 5.6 — Витрати на придбання спецустаткування

Найменування спецустаткування	Ціна за одиницю, грн.	Витрачено	Вартість спецустаткування, грн.
IP-відеокамера вулична Full HD	3500	1	3500
Велосипедний шолом (тестовий зразок)	1200	1	1200
Кронштейн/штатив для кріплення відеокамери	800	1	800
Всього, з врахуванням коефіцієнта транспортних витрат			6050

До балансової вартості програмного забезпечення входять витрати на його інсталяцію, тому ці витрати беруться додатково в розмірі з 10% до 12% від вартості програмного забезпечення. Балансову вартість програмного забезпечення розраховують за формулою:

$$В_{прг} = \sum_{1}^{к} Ц_{іпрг} \cdot С_{прг.і} \cdot К_і,$$

де $Ц_{іпрг}$ — ціна придбання програмного забезпечення і-го виду, грн.;

$С_{прг.і}$ — кількість одиниць програмного забезпечення відповідного виду, шт.;

$К_і$ — коефіцієнт, що враховує інсталяцію, налагодження програмного забезпечення, $К_і$ = (з 1,1 до 1,12);

к — кількість видів програмного забезпечення.

Таблиця 5.7 — Витрати на придбання програмного забезпечення

Найменування програмного забезпечення	Ціна за одиницю, грн.	Витрачено	Вартість програмного забезпечення, грн.
Операційна система Windows 11 Pro	7 000	1	7000
MATLAB (академічна ліцензія)	12 000	1	12000
Microsoft Office	4 000	1	4000
Всього, з врахуванням коефіцієнта інсталяції та налагодження			25530

Амортизація обладнання, комп'ютерів та приміщень, які використовувались під час (чи для) виконання даного етапу роботи.

У спрощеному вигляді амортизаційні відрахування А в цілому бути розраховані за формулою:

$$\text{А} = \frac{\text{Ц}_\text{б}}{\text{Т}_\text{в}} \cdot \frac{t}{12},$$

де $\text{Ц}_\text{б}$ – загальна балансова вартість всього обладнання, комп'ютерів, приміщень тощо, що використовувались для виконання даного етапу роботи, грн.;

t – термін використання основного фонду, місяці;

$\text{Т}_\text{в}$ – термін корисного використання основного фонду, роки.

Таблиця 5.8 — Амортизаційні відрахування за видами основних фондів

Найменування	Балансова вартість, грн.	Строк корисного використання, років	Термін використання, місяців	Сума амортизації, грн.
Робоча станція для обробки відео	25000	3	36	15000,0
Сервер (для зберігання й аналітики)	35000	3	36	21000,0
ДБЖ (джерело безперебійного живлення)	5000	3	24	2000,0
Всього	38000			

Витрати на силову електроенергію $\text{В}_\text{е}$, якщо ця стаття має суттєве значення для виконання даного етапу роботи, розраховуються за формулою:

$$\text{Ве} = \sum \frac{W_i \cdot t_i \cdot \text{Ц}_\text{е} \cdot \text{К}_\text{впі}}{\text{ККД}}$$

$$= \frac{0{,}5 \cdot 160 \cdot 4{,}32 \cdot 0{,}75}{0{,}98} + \frac{0{,}08 \cdot 160 \cdot 4{,}32 \cdot 0{,}75}{0{,}98}$$

$$+ \frac{0{,}015 \cdot 160 \cdot 4{,}32 \cdot 0{,}75}{0{,}98} = 314{,}7 \text{ грн.},$$

де W_i – встановлена потужність обладнання, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год.;

Це – вартість 1 кВт електроенергії, 4,32 грн.;

$К_{впі}$ – коефіцієнт використання потужності;

ККД – коефіцієнт корисної дії обладнання.

Таблиця 5.9 — Витрати на електроенергію

Найменування обладнання	Потужність, кВт	Тривалість годин роботи
Робоча станція (системний блок)	0,5	160
Монітор 24"	0,08	160
IP-відеокамера	0,015	160

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$Ів = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%} = (81818 + 3502) \cdot \frac{65}{100} = 55458{,}12 \text{ грн.},$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати».

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…200% від суми основної заробітної плати дослідників та робітників за формулою:

$$\text{Внзв} = \left(З_о + З_р\right) \cdot \frac{Н_{нзв}}{100\%} = (81818 + 3502) \cdot \frac{190}{100} = 162108{,}3 \text{ грн.},$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати».

Витрати на проведення розробки комп'ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж. Витрати на проведення науково-дослідної роботи розраховуються як сума всіх попередніх статей витрат за формулою:

$$В_{заг} = 81818 + 3502 + 8532 + 20647 + 5060 + 57165 + 605025530 + 38000314{,}7 + 55458{,}12 + 162108{,}3 = 464185{,}9 \text{ грн.}$$

Загальні витрати ЗВ на завершення науково-дослідної (науково-технічної) роботи з розробки комп'ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{В_{заг}}{\eta} = \frac{464185{,}9}{0{,}5} = 928371{,}8 \text{ грн.},$$

де η – коефіцієнт, що характеризує етап виконання науково-дослідної роботи.

Оскільки, якщо науково-технічна розробка знаходиться на стадії розробки дослідного зразка, то η=0,5.

5.3 Розрахунок економічної ефективності науково-технічної розробки комп'ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж за її можливої комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки комп'ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж, є збільшення у потенційного інвестора величини чистого прибутку.

В даному випадку відбувається розробка засобу, тому основу майбутнього економічного ефекту буде формувати: ΔN – збільшення кількості споживачів, яким надається відповідна інформаційна послуга в аналізовані періоди часу; N – кількість споживачів, яким надавалась відповідна інформаційна послуга у році до впровадження результатів нової науково-технічної розробки; $Ц_б$ – вартість послуги у році до впровадження інформаційної системи; $\pm\Delta Ц_о$ – зміна вартості послуги (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу.

Можливе збільшення чистого прибутку у потенційного інвестора ΔПі для кожного із років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховується за формулою:

$$\Delta П = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N_i)_i \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\vartheta}{100}\right),$$

де ±ΔЦ — зміна основного якісного показника від впровадження результатів науково-технічної розробки в аналізованому році. Зазвичай, таким показником може бути зміна ціни реалізації одиниці нової розробки в аналізованому році (відносно року до впровадження цієї розробки);

$\pm\Delta Ц_о$ може мати як додатне, так і від'ємне значення (від'ємне – при зниженні ціни відносно року до впровадження цієї розробки, додатне – при зростанні ціни);

N — основний кількісний показник, який визначає величину попиту на аналогічні чи подібні розробки у році до впровадження результатів нової науково-технічної розробки;

$Ц_о$ — основний якісний показник, який визначає ціну реалізації нової науково-технічної розробки в аналізованому році;

$Ц_б$ — основний якісний показник, який визначає ціну реалізації існуючої (базової) науково-технічної розробки у році до впровадження результатів;

ΔN — зміна основного кількісного показника від впровадження результатів науково-технічної розробки в аналізованому році. Зазвичай таким показником може бути зростання попиту на науково-технічну розробку в аналізованому році (відносно року до впровадження цієї розробки);

λ — коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2025 році ставка податку на додану вартість становить 20%, а коефіцієнт $\lambda = 0{,}8333$;

ρ — коефіцієнт, який враховує рентабельність інноваційного продукту (послуги). Рекомендується брати $\rho = 0{,}2...0{,}5$; ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2025 році $\vartheta = 18\%$.

Очікуваний термін життєвого циклу розробки 3 роки, тому:

Таблиця 5.10 — Життєвий цикл

1-й рік	ΔП1	307377	грн.
2-й рік	ΔП2	409836	грн.
3-й рік	ΔП3	512295	грн.

Таблиця 5.11 — Детальний життєвий цикл

Рік	Ц0, грн.	N, шт.	ΔЦ0, грн.	ΔN, шт.	Цб, грн.	N0, шт.
1	100000	20	50000	10	150000	10
2	100000	25	50000	15	150000	10
3	100000	30	50000	20	150000	10

Далі розраховують приведену вартість збільшення всіх чистих прибутків ПП, що їх може отримати потенційний інвестор від можливого впровадження та

комерціалізації науково-технічної розробки комп’ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t} = \frac{307377}{(1+0{,}1)^1} + \frac{409836}{(1+0{,}1)^2} + \frac{512295}{(1+0{,}1)^3} = 1003036 \text{ грн.},$$

де ΔПі — збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн.;

Т — період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки (приймаємо Т=3 роки);

τ — ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,05…0,15;

t — період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

Далі розраховують величину початкових інвестицій PV , які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки комп’ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж. Для цього можна використати формулу:

$$PV = k_{\text{інв}} \cdot ЗВ = 1 \cdot 928371{,}8 = 434508{,}4 \text{ грн.}$$

де $k_{\text{інв}}$ — коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки комп’ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай kінв=1...5, але може бути і більшим;

ЗВ — загальні витрати на проведення науково-технічної розробки та оформлення

її результатів, грн.

Тоді абсолютний економічний ефект $Е_{абс}$ або чистий приведений дохід для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки комп'ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж становитиме:

$$Е_{абс} = ПП - PV = 1003036 - 928372 = 74664 \text{ грн.},$$

де ПП – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки комп'ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж, грн.;

PV — теперішня вартість початкових інвестицій, грн.

Оскільки $Е_{абс}>0$, то можемо припустити про потенційну зацікавленість у розробці комп'ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж.

Для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність Ев або показник внутрішньої норми дохідності вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку комп'ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж вкладати буде економічно недоцільно.

Внутрішня економічна дохідність інвестицій Ев, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки комп'ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж, розраховується за формулою:

$$Е_в = \sqrt[Т_ж]{1 + \frac{Еабс}{PV}} = \sqrt[з]{1 + \frac{74664}{928372}} = 0{,}36,$$

де $Т_ж$ — життєвий цикл розробки комп’ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж, роки.

Далі розраховуємо період окупності інвестицій $Т_о$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки комп’ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж:

$$Т_о = \frac{1}{Е_в} = \frac{1}{0{,}36} = 2{,}78 \text{ роки.}$$

Оскільки $Т_о$<1…3-х років, то це свідчить про комерційну привабливість науково-технічної розробки комп’ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж і може спонукати потенційного інвестора профінансувати впровадження цієї розробки комп’ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж та виведення її на ринок.

Проведений комерційний і технологічний аудит підтвердив, що розробка комп’ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж має науково-технічний рівень та комерційний потенціал вище середнього (середня оцінка експертів — 32 бали). Технологічний аудит засвідчив технічну здійсненність проєкту, можливість його інтеграції у міську інфраструктуру та перспективи масштабування, тоді як комерційний аудит визначив реальну ринкову нішу й конкурентні переваги розробки.

Загальна сума витрат на створення системи становить 928,37 тис. грн. Розрахунки економічної ефективності показали, що протягом трирічного життєвого циклу очікуване збільшення чистого прибутку може досягти 1 003 036 грн, що перевищує обсяг інвестицій та забезпечує економічну доцільність упровадження.

Отже, проєкт є перспективним як з технологічного, так і з комерційного погляду та може бути рекомендований до подальшого розвитку і комерціалізації.

ВИСНОВКИ

У магістерській кваліфікаційній роботі розв’язано задачу розробки комп’ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж. Метою роботи було створення програмного рішення, яке забезпечує автоматизоване виявлення велосипедистів у відеопотоці та визначення наявності/відсутності захисного шолома з подальшим накопиченням статистики для аналізу дотримання вимог безпеки.

У першому розділі виконано огляд та аналіз предметної галузі, сформульовано постановку задачі та розглянуто типові сценарії ризиків для велосипедистів, що обґрунтовують необхідність відеомоніторингу. Визначено цілі моніторингу та критерії успіху системи, а також проаналізовано сучасні методи комп’ютерного зору для виявлення та класифікації об’єктів, зокрема підходи на основі CNN і трансформерів. Проведено порівняння детекторів (YOLO, SSD, EfficientDet, DETR) і класифікаторів (ResNet, EfficientNet, ViT), описано принципи підготовки даних, формати розмітки та вимоги до якості анотацій. Також у межах розділу наведено методи оцінювання результатів розпізнавання, використано метрики Precision, Recall, mAP і F1, а окремо розглянуто існуючі системи моніторингу використання шоломів та відеоконтролю велосипедистів.

У другому розділі виконано теоретичні дослідження та сформовано вимоги до системи і критерії вибору архітектури нейронної мережі. Розглянуто підхід до вдосконалення методу розпізнавання об’єктів на зображенні під час дорожнього руху, визначено етапи підготовки даних, доповнення датасету та застосування аугментації. Виконано навчання моделі з налаштуванням ключових гіперпараметрів, використанням прийомів регуляризації та механізмів ранньої зупинки, а також проведено донавчання моделі на підготовлених даних. За результатами оцінювання якості визначено робочий поріг прийняття рішення, який забезпечує узгоджений баланс між точністю та повнотою виявлення.

У третьому розділі розроблено та реалізовано систему моніторингу та оцінки безпеки руху велосипедистів. Сформовано архітектуру і структурну схему комп’ютерної системи, визначено вимоги до апаратної та програмної частини,

обґрунтовано вибір програмних засобів реалізації. Реалізовано модуль детекції на базі навченої моделі з постобробкою результатів, а також модуль моніторингу відеопотоку в реальному часі, який забезпечує підрахунок порушень і формування подій. Окремо розроблено інтерфейс користувача в одному робочому вікні, що поєднує перегляд відео з накладеними результатами детекції, елементи керування та відображення поточної статистики і журналу подій, що підвищує зручність експлуатації системи оператором.

У четвертому розділі виконано перевірку працездатності та тестування розробленої системи на різних вхідних відеоданих. Оцінено стабільність роботи під час безперервної обробки потоку, коректність візуалізації та підрахунку подій, а також узгодженість результатів детекції зі статистикою в журналі. Визначено напрями подальшого вдосконалення, зокрема підвищення стійкості до складного освітлення, часткових перекриттів та рухових розмиттів, а також оптимізацію продуктивності для роботи на менш потужних обчислювальних платформах.

У п’ятому розділі виконано економічне обґрунтування розробки: проведено комерційний і технологічний аудит, визначено витрати на створення та впровадження, а також розраховано економічну ефективність науково-технічної розробки за умови можливої комерціалізації потенційним інвестором. Отримані результати підтверджують доцільність практичного застосування системи як інструменту підтримки контролю безпеки дорожнього руху та основи для подальшого розвитку функціональності розширення набору порушень, інтеграція з централізованими сервісами, аналітика за періодами, удосконалення моделі).

Таким чином, у роботі отримано комплексне рішення, що поєднує теоретичні засади комп’ютерного зору, навчання нейронної мережі та практичну програмну реалізацію системи відеомоніторингу. Запропонований підхід дозволяє автоматизувати контроль використання захисного шолома велосипедистами, формувати статистику порушень і забезпечувати зручне представлення результатів для оператора в режимі, наближеному до реального часу. Розроблена система може бути використана як прототип для подальшого впровадження на контрольованих ділянках дорожньої інфраструктури та масштабування функціоналу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1) Ultralytics YOLOv8 — офіційна документація [Електронний ресурс]. — Режим доступу: https://docs.ultralytics.com/ (дата звернення: 01.10.2025). — Назва з екрана.

2) PyTorch — офіційний сайт та документація [Електронний ресурс]. — Режим доступу: https://pytorch.org/ (дата звернення: 04.09.2025). — Назва з екрана.

3) OpenCV Documentation — офіційна документація OpenCV [Електронний ресурс]. — Режим доступу: https://docs.opencv.org/ (дата звернення: 08.10.2025). — Назва з екрана.

4) Helmet Detection (Kaggle Dataset) — Andrew Mvd [Електронний ресурс]. — Режим доступу: https://www.kaggle.com/datasets/andrewmvd/helmet-detection (дата звернення: 12.10.2025). — Назва з екрана.

5) LabelImg — Graphical Image Annotation Tool (GitHub) [Електронний ресурс]. — Режим доступу: https://github.com/tzutalin/labelImg (дата звернення: 12.10.2025). — Назва з екрана.

6) Tan M., Le Q. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks (arXiv) [Електронний ресурс]. — Режим доступу: https://arxiv.org/abs/1905.11946 (дата звернення: 13.10.2025). — Назва з екрана.

7) COCO — Common Objects in Context (офіційний сайт) [Електронний ресурс]. — Режим доступу: https://cocodataset.org/ (дата звернення: 15.10.2025). — Назва з екрана.

8) PASCAL Visual Object Classes (VOC) Challenge (офіційний сайт) [Електронний ресурс]. — Режим доступу: http://host.robots.ox.ac.uk/pascal/VOC/ (дата звернення: 16.10.2025). — Назва з екрана.

9) Bochkovskiy A., Wang C.-Y., Liao H.-Y. M. YOLOv4: Optimal Speed and Accuracy of Object Detection (arXiv) [Електронний ресурс]. — Режим доступу: https://arxiv.org/abs/2004.10934 (дата звернення: 20.10.2025). — Назва з екрана.

10) Wang C.-Y., Bochkovskiy A., Liao H.-Y. M. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors (arXiv) [Електронний

ресурс]. — Режим доступу: https://arxiv.org/abs/2207.02696 (дата звернення: 20.10.2025). — Назва з екрана.

11) Redmon J., Farhadi A. YOLOv3: An Incremental Improvement (arXiv) [Електронний ресурс]. — Режим доступу: https://arxiv.org/abs/1804.02767 (дата звернення: 21.10.2025). — Назва з екрана.

12) Liu W. та ін. SSD: Single Shot MultiBox Detector (arXiv) [Електронний ресурс]. — Режим доступу: https://arxiv.org/abs/1512.02325 (дата звернення: 22.10.2025). — Назва з екрана.

13) Tan M., Pang R., Le Q. V. EfficientDet: Scalable and Efficient Object Detection (arXiv) [Електронний ресурс]. — Режим доступу: https://arxiv.org/abs/1911.09070 (дата звернення: 22.10.2025). — Назва з екрана.

14) Szegedy C., Ioffe S., Vanhoucke V., Alemi A. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning (arXiv) [Електронний ресурс]. — Режим доступу: https://arxiv.org/abs/1602.07261 (дата звернення: 23.10.2025). — Назва з екрана.

15) Albumentations — документація з аугментацій зображень [Електронний ресурс]. — Режим доступу: https://albumentations.ai/docs/ (дата звернення: 24.10.2025). — Назва з екрана.

16) CVAT — Computer Vision Annotation Tool (офіційний сайт) [Електронний ресурс]. — Режим доступу: https://www.cvat.ai/ (дата звернення: 24.10.2025). — Назва з екрана.

17) FiftyOne — документація для аналізу та валідації датасетів/моделей [Електронний ресурс]. — Режим доступу: https://docs.voxel51.com/ (дата звернення: 26.10.2025). — Назва з екрана.

18) Weights & Biases — офіційна документація (експерименти, логування, моніторинг) [Електронний ресурс]. — Режим доступу: https://docs.wandb.ai/ (дата звернення: 27.10.2025). — Назва з екрана.

19) NVIDIA TensorRT — офіційна документація (оптимізація інференсу) [Електронний ресурс]. — Режим доступу:

https://docs.nvidia.com/deeplearning/tensorrt/ (дата звернення: 28.10.2025). — Назва з екрана.

20) ResearchGate — порівняння моделей [Електронний ресурс]. — Режим доступу: https://www.researchgate.net/figure/Comparison-of-accuracy-Faster-R-CNN-R-FCN-SSD-and-YOLO-models-using-the-database-MS_fig7_342570032 (дата звернення: 30.10.2025). — Назва з екрана

21) Roboflow Bike Helmet Detection Computer Vision [Електронний ресурс]. — Режим доступу: https://universe.roboflow.com/bike-helmets/bike-helmet-detection-2vdjo (дата звернення: 12.11.2025). — Назва з екрана

22) Циганков О.А. Комп'ютерна система моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж / О.А. Циганков, С.В. Городецька // Матеріали МНПІК Молодь в науці: дослідження, проблеми, перспективи (МН-2026), 2026 р. Режим доступу: — https://conferences.vntu.edu.ua/index.php/mn/mn2026/paper/viewFile/26536/21886 (дата звернення: 16.11.2025).

23) Object Tracking with YOLOv8 and Python [Електронний ресурс]. — Режим доступу: https://pyimagesearch.com/2024/06/17/object-tracking-with-yolov8-and-python/ (дата звернення: 17.11.2025). — Назва з екрана.

24) Марчук Д. Аналіз сучасних алгоритмів виявлення і розпізнавання об'єктів з відеопотоку для систем управління паркуванням в реальному часі // Вісник Хмельницького національного університету. Технічні науки. — 2023. — № 3 (321). — С. 17–23. — DOI: 10.31891/2307-5732-2023-321-3-17-23 [Електронний ресурс]. — Режим доступу: https://journals.khnu.km.ua/vestnik/wp-content/uploads/2023/07/vknu-ts-2023-n3321-17-23.pdf (дата звернення: 18.11.2025). — Назва з екрана

25) Методичні вказівки до виконання магістерських кваліфікаційних робіт студентами спеціальності 123 «Комп'ютерна інженерія» / Укладачі О. Д. Азаров, О. В. Дудник, С. І. Швець. — Вінниця : ВНТУ, 2023. — 57 с.

ДОДАТОК А

Технічне завдання

Міністерство освіти та науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

проф., д.т.н. О. Д. Азаров

«25» вересня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

«Комп’ютерна система моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж»

Науковий керівник: к.т.н., доц. каф. ОТ

_____ Городецька О.С.

«____» _____ 2025 р.

Магістрант групи 2КІ-24м

Циганков О.А.

ВНТУ 2025

1 Підстава для виконання магістерської кваліфікаційної роботи (МКР)

Підставою для розробки даної магістерської кваліфікаційної роботи є наказ ВНТУ №_313_ від «24» вересня 2025 року та рішення засідання кафедри обчислювальної техніки (протокол №__ від «_____»__ 2025 року).

2 Мета і призначення МКР

2.1 Метою роботи є збільшення точності розпізнавання об’єктів на зображенні під час відеозйомки дорожнього руху із застосуванням нейронних мереж.

2.2 Призначенням розробки є створення комп’ютерної системи, що автоматично виявляє велосипедистів на відеопотоці та визначає наявність або відсутність захисного шолома із застосуванням нейронних мереж. Система забезпечує відображення результатів оператору, підрахунок спрацювань і накопичення статистики порушень у базі даних для подальшого аналізу та формування звітів.

3 Вихідні дані для виконання МКР

Вихідними даними для виконання МКР є відеопотік або окремі кадри з камер спостереження, на яких присутні велосипедисти. Додатково використовуються розмічені датасети зображень у форматі YOLO - класи «With Helmet» та «Without Helmet» для навчання і донавчання нейромережевої моделі. Також до вихідних даних належать параметри роботи системи:інтервал обробки кадрів, пороги confidence/IoU, шлях до ваг моделі, що задаються під час запуску. Результати детекції фіксуються у вигляді записів у базі даних (час події та кількість спрацювань по кожному класу), які надалі використовуються для побудови статистики й графіків.

4 Вимоги до виконання МКР

Вимоги до виконання МКР:

—МКР повинна містити вступ, розділи 1–5, висновки, перелік джерел та додатки, оформлені з наскрізною нумерацією і логічними переходами між розділами.

—У роботі мають бути чітко сформульовані мета, задачі, об’єкт і предмет дослідження, а також обґрунтовано вибір методів, моделей та програмних засобів реалізації.

— Обов’язково необхідно навести опис даних і процесу їх підготовки (формат розмітки, якість анотацій, аугментація), а також методику оцінювання результатів із використанням метрик Precision, Recall, mAP та F1.

— Практична частина повинна містити реалізацію модулів системи (детекція, моніторинг, інтерфейс), демонстрацію працездатності через тестування та подання результатів у вигляді рисунків/таблиць/графіків із посиланнями в тексті.

— Список використаних джерел і всі запозичення мають бути оформлені згідно з методичними вимогами, а робота повинна пройти перевірку на академічну доброчесність і бути підготовленою до захисту з повним комплектом додатків та супровідних матеріалів.5. Етапи МКР та очікувані результати

Етапи роботи та очікувані результати приведено в таблиці А.1.

Таблиця А.1 — Етапи МКР

№ етапу	Назва етапу	Термін виконання		Очікуванні результати
		початок	кінець	
1	Огляд і аналіз джерел інформації	26.09.2025	6.10.2025	Перший розділ
2	Теоретичні дослідження технологій побудови системи та вибір засобів реалізації	7.10.2025	10.10.2025	Другий розділ
3	Розробка комп’ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж	11.10.2025	9.11.2025	Третій розділ
4	Тестування комп’ютерної системи	10.11.2025	16.11.2025	Четвертий розділ
5	Економічна частина	17.11.2025	5.12.2025	П’ятий розділ

6. Матеріали, що подаються до захисту МКР

До захисту подаються: пояснювальна записка МКР, графічні і ілюстративні матеріали, протокол попереднього захисту МКР на кафедрі, відгук наукового

керівника, відгук опонента, протоколи складання державних екзаменів, анотації до

МКР українською та іноземною мовами, нормоконтроль про відповідність

оформлення МКР діючим вимогам.

7. Порядок контролю виконання та захисту МКР

Виконання етапів графічної та розрахункової документації МКР

контролюється науковим керівником згідно зі встановленими термінами. Захист

МКР відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

8. Вимоги до оформлювання та порядок виконання МКР 8.1 При оформлювання МКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— міждержавний ГОСТ 2.104—2006 «Єдина система конструкторської документації. Основні написи»;

— Методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 123 «Комп'ютерна інженерія» (освітня програма «Комп'ютерна інженерія»). Кафедра обчислювальної техніки ВНТУ 2023;

— документами на які посилаються у вище вказаних.

8.2 Порядок виконання МКР викладено в «Положення про кваліфікаційні роботи на другому (магістерському) рівні вищої освіти СУЯ ВНТУ—03.02.02—П.001.01:21».

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: КОМП'ЮТЕРНА СИСТЕМА МОНІТОРИНГУ ТА ОЦІНКИ БЕЗПЕКИ РУХУ ВЕЛОСИПЕДИСТІВ ІЗ ЗАСТОСУВАННЯМ НЕЙРОННИХ МЕРЕЖ

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

Підрозділ кафедра ОТ, ФІТКІ, 2КІ-24м
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism (КП1) 1 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

☐ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак академічного плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту.

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

☐ У роботі виявлено ознаки академічного плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень. Робота до захисту не приймається.

Експертна комісія:

Азаров О.Д. зав. каф. ОТ _____
(прізвище, ініціали, посада) (підпис)

Мартинюк Т.Б. гар. осв. пр. _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ _____
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Городецька О.С. к.т.н., доц. каф. ОТ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Циганков О.А.
(підпис) (прізвище, ініціали)

ДОДАТОК В

Структура нейронної мережі

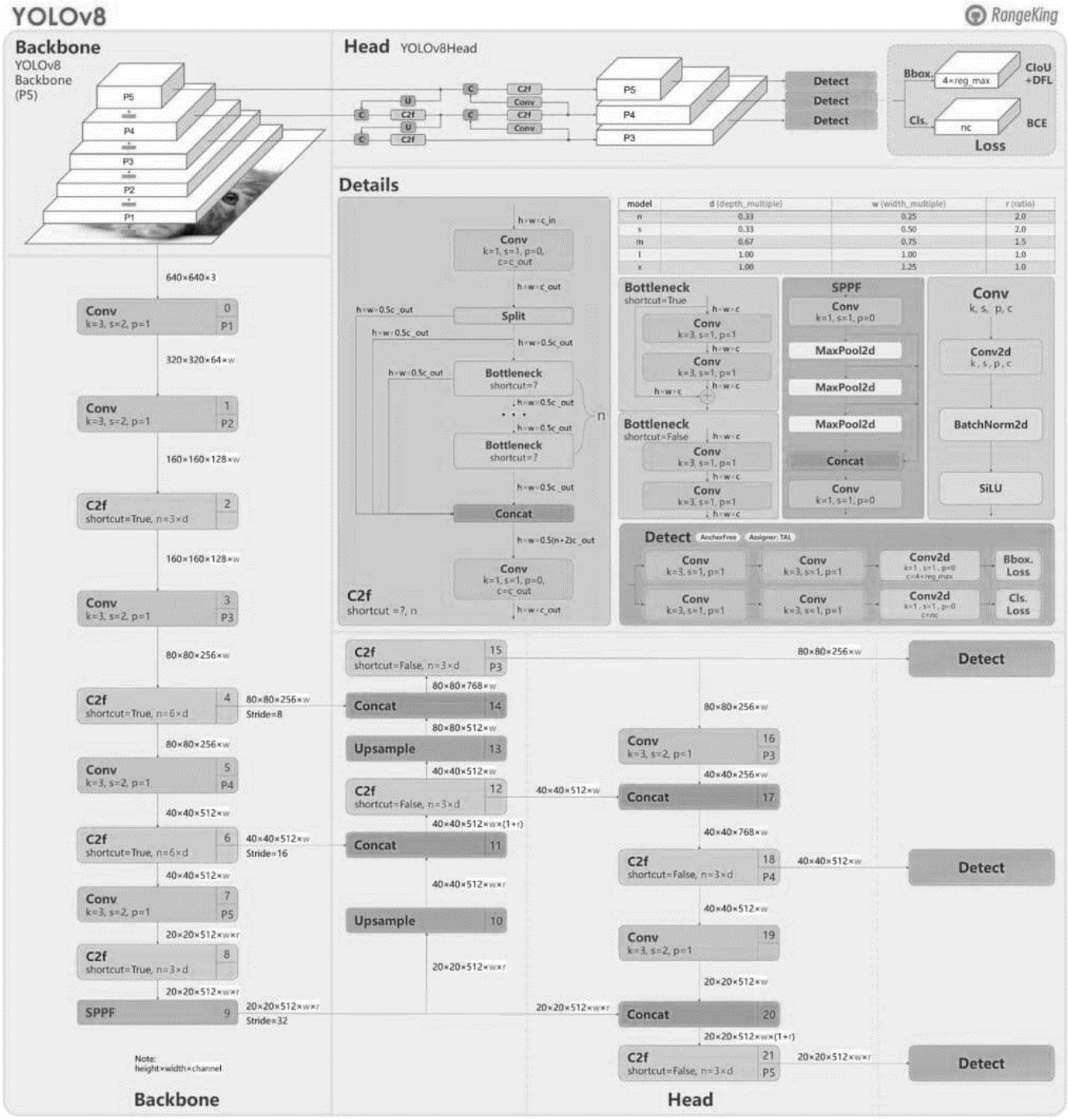


Рисунок В.1 — Структура нейронної мережі

ДОДАТОК Г

Блок-схема алгоритму донавчання нейронної мережі

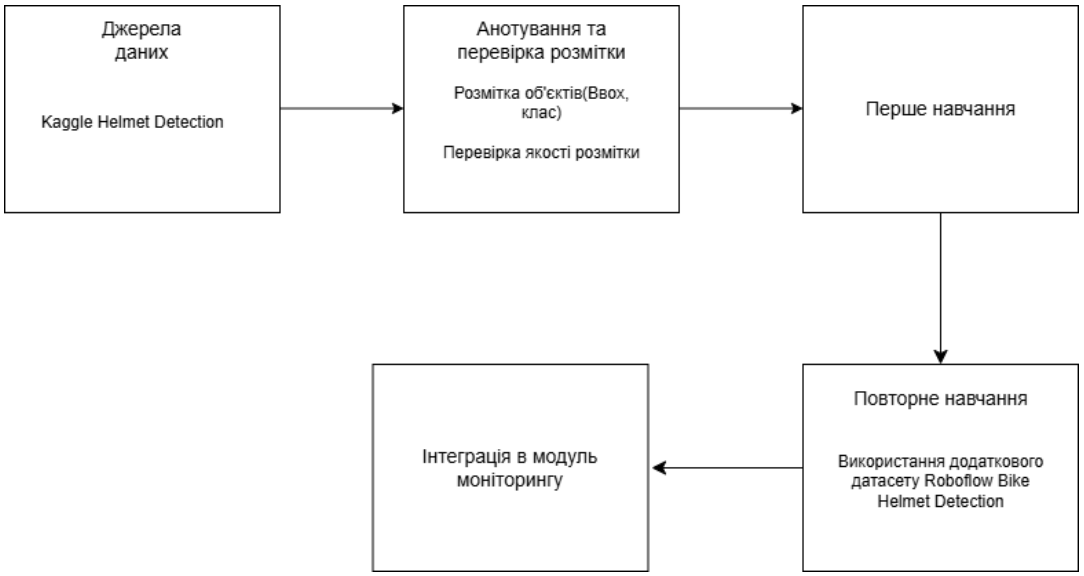


Рисунок Г.1 — Блок-схема алгоритму донавчання нейронної мережі

ДОДАТОК Д

Матриця помилок моделі YOLOv8

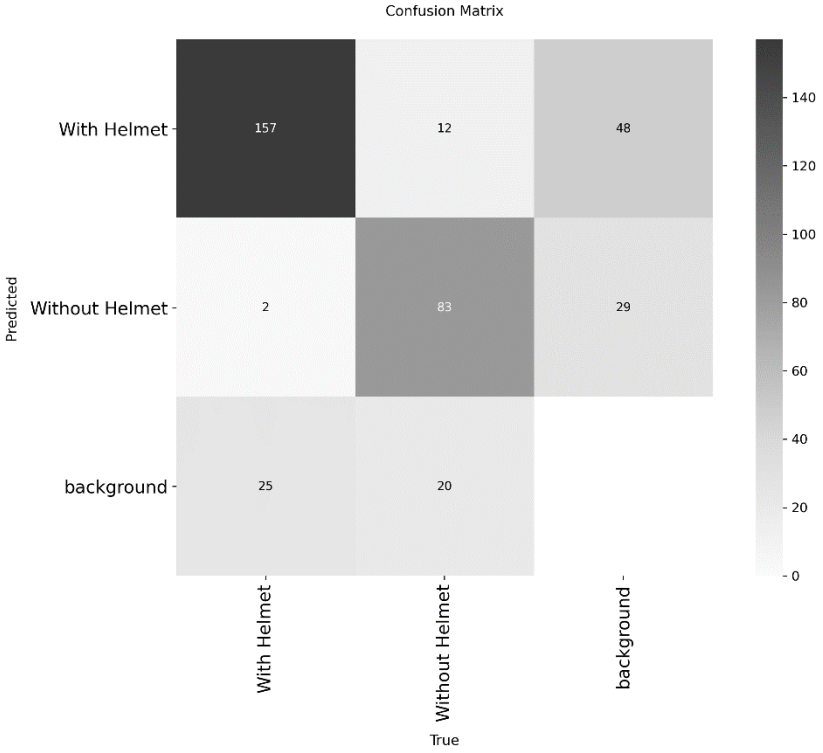


Рисунок Д.1 — Матриця помилок моделі YOLOv8

ДОДАТОК Е

Архітектура комп’ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж

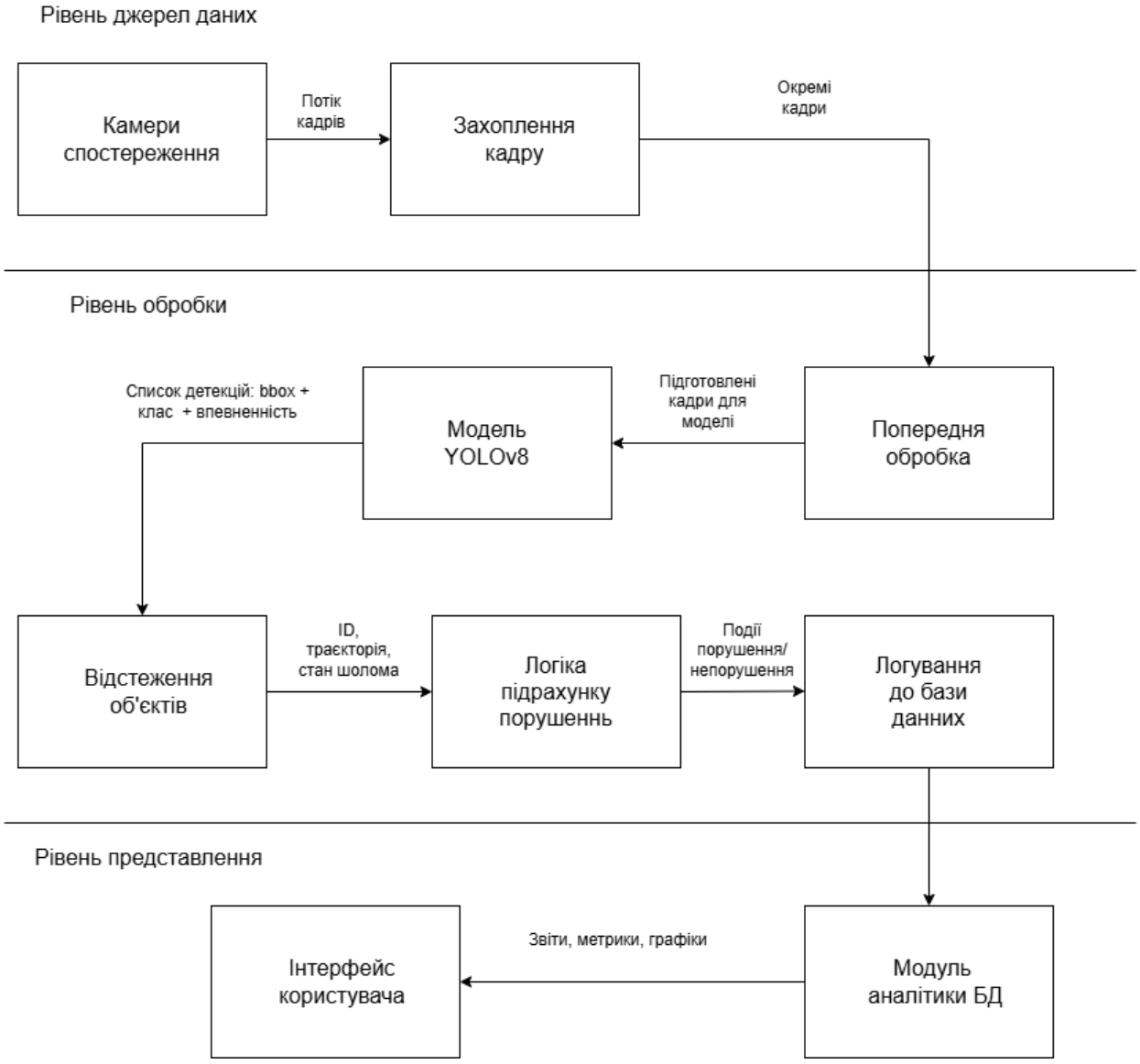


Рисунок Е.1 — Архітектура системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж

ДОДАТОК Ж

Структурна схема комп’ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж

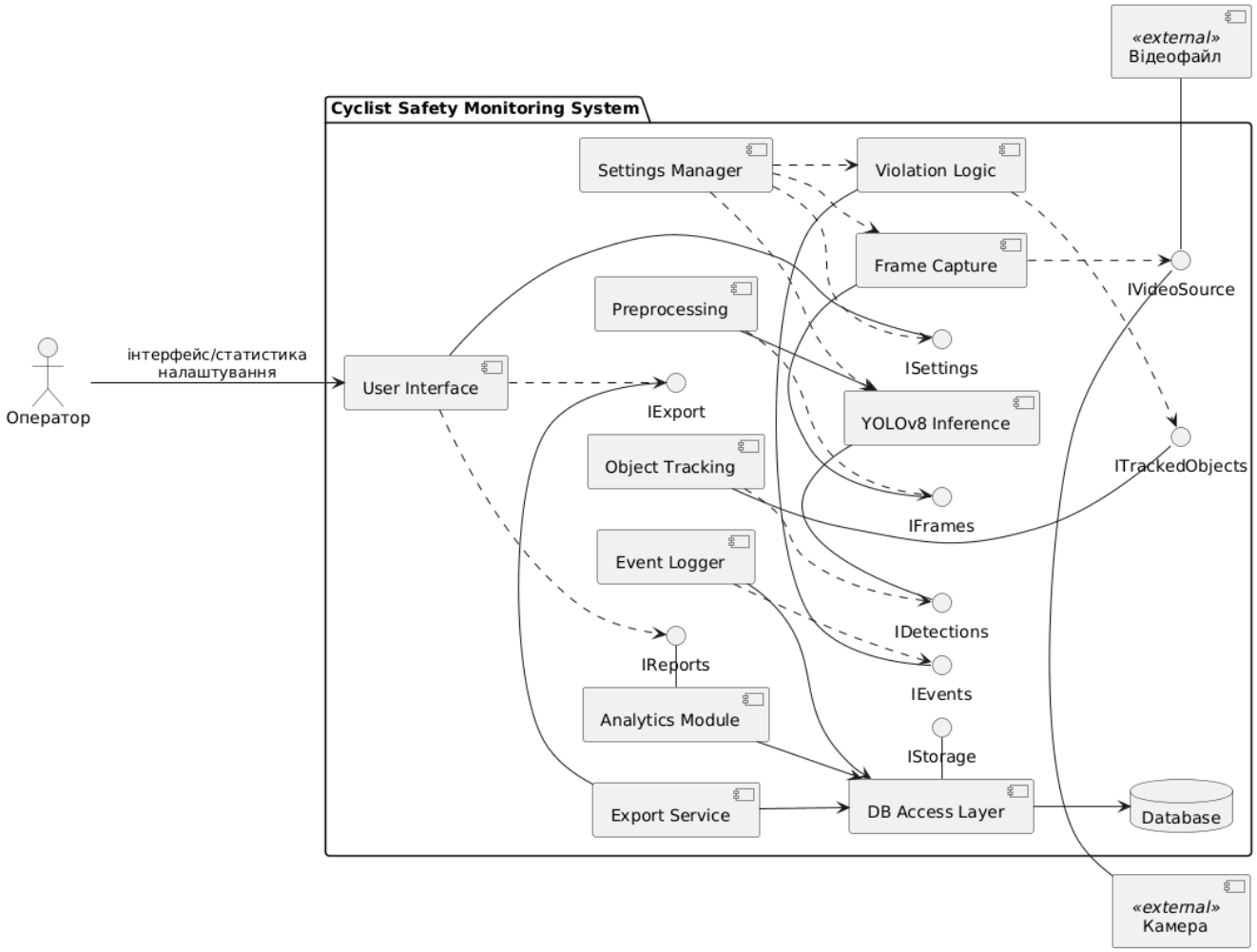


Рисунок Ж.1 — Структурна схема комп’ютерної системи моніторингу та оцінки безпеки руху велосипедистів із застосуванням нейронних мереж

ДОДАТОК Л
Лістинг програмного коду

Лістинг Л.1 — Зміна розмітки анотацій

```
# VOC -> YOLO converter with autosplit (ASCII only)
import os, random, shutil, xml.etree.ElementTree as ET
from pathlib import Path

# === EDIT THESE PATHS ===
IMAGES_DIR = r"C:\ml\helmet\images"
ANN_DIR    = r"C:\ml\helmet\annotations"
OUT_ROOT   = r"C:\ml\helmet\datasets\helmet"      # output root

TRAIN_SPLIT = 0.80
RANDOM_SEED = 42

# Optional class remap, e.g.:
# MAP_CLASSES = {"with_helmet":"helmet", "without_helmet":"no_helmet"}
MAP_CLASSES = None
# ========================

def parse_voc(xml_path):
    tree = ET.parse(xml_path)
    root = tree.getroot()
    size = root.find('size')
    if size is None:
        return None, []

    w = float(size.find('width').text)
    h = float(size.find('height').text)

    objs = []
    for obj in root.findall('object'):
        name_el = obj.find('name')
        box = obj.find('bndbox')
        if name_el is None or box is None:
            continue
        cls = name_el.text.strip()

        try:
            xmin = float(box.find('xmin').text)
            ymin = float(box.find('ymin').text)
            xmax = float(box.find('xmax').text)
            ymax = float(box.find('ymax').text)
        except:
            continue

        cx = ((xmin + xmax) / 2.0) / w
        cy = ((ymin + ymax) / 2.0) / h
        bw = (xmax - xmin) / w
        bh = (ymax - ymin) / h
```

```
        if bw <= 0 or bh <= 0:
            continue
        objs.append((cls, cx, cy, bw, bh))

    fname_el = root.find('filename')
    fname = fname_el.text.strip() if fname_el is not None else None
    return fname, objs

def ensure_dirs():
    for p in [
        Path(OUT_ROOT, "train", "images"),
        Path(OUT_ROOT, "train", "labels"),
        Path(OUT_ROOT, "valid", "images"),
        Path(OUT_ROOT, "valid", "labels"),
    ]:
        p.mkdir(parents=True, exist_ok=True)

def main():
    ann_files = [f for f in os.listdir(ANN_DIR) if f.lower().endswith(".xml")]
    ann_files.sort()
    if not ann_files:
        print("No .xml found in:", ANN_DIR)
        return

    cls_counts = {}
    pairs = []

    for x in ann_files:
        xml_path = os.path.join(ANN_DIR, x)
        fname, objs = parse_voc(xml_path)
        if fname is None:
            fname = os.path.splitext(x)[0] + ".jpg"
        for (cls, cx, cy, bw, bh) in objs:
            cls_counts[cls] = cls_counts.get(cls, 0) + 1
        pairs.append((fname, xml_path, objs))

    print("Classes found:")
    for k in sorted(cls_counts.keys()):
        print("  - {}: {}".format(k, cls_counts[k]))

    if MAP_CLASSES:
        remapped = {}
        for cls, cnt in cls_counts.items():
            if cls in MAP_CLASSES:
                newn = MAP_CLASSES[cls]
                remapped[newn] = remapped.get(newn, 0) + cnt
        class_names = sorted(remapped.keys())
        print("\nUsing mapped classes:", class_names)
    else:
        class_names = sorted(cls_counts.keys())
        print("\nUsing classes as-is:", class_names)
```

```
class_to_id = {c: i for i, c in enumerate(class_names)}

random.seed(RANDOM_SEED)
random.shuffle(pairs)
n_train = int(len(pairs) * TRAIN_SPLIT)
train_pairs = pairs[:n_train]
valid_pairs = pairs[n_train:]

ensure_dirs()

def process(split_name, split_pairs):
    img_out = Path(OUT_ROOT, split_name, "images")
    lbl_out = Path(OUT_ROOT, split_name, "labels")
    ok, skipped = 0, 0
    for (img_name, xml_path, objs) in split_pairs:
        src_img = None
        stem = os.path.splitext(img_name)[0]
        for ext in [".jpg", ".jpeg", ".png"]:
            cand = os.path.join(IMAGES_DIR, stem + ext)
            if os.path.exists(cand):
                src_img = cand
                break
        if src_img is None:
            skipped += 1
            continue

        lines = []
        for (cls, cx, cy, bw, bh) in objs:
            if MAP_CLASSES:
                if cls not in MAP_CLASSES:
                    continue
                mapped = MAP_CLASSES[cls]
                if mapped not in class_to_id:
                    continue
                cid = class_to_id[mapped]
            else:
                if cls not in class_to_id:
                    continue
                cid = class_to_id[cls]

            cx = min(max(cx, 0.0), 1.0)
            cy = min(max(cy, 0.0), 1.0)
            bw = min(max(bw, 1e-6), 1.0)
            bh = min(max(bh, 1e-6), 1.0)
            lines.append("{} {:.6f} {:.6f} {:.6f} {:.6f}".format(cid, cx, cy, bw, bh))

        dst_img = img_out / Path(src_img).name
        shutil.copy2(src_img, dst_img)
        dst_lbl = lbl_out / (dst_img.stem + ".txt")
        with open(dst_lbl, "w", encoding="utf-8") as f:
            f.write("\n".join(lines))
        ok += 1
```

```
        return ok, skipped

    ok_tr, sk_tr = process("train", train_pairs)
    ok_va, sk_va = process("valid", valid_pairs)

    print("\nDone. Train: {} (skipped {}); Valid: {} (skipped {})".format(ok_tr, sk_tr, ok_va, sk_va))

    data_yaml = "train: {}/train/images\nval: {}/valid/images\n\nnc: {}\nnames: {}\n".format(
        OUT_ROOT.replace(os.sep, '/'),
        OUT_ROOT.replace(os.sep, '/'),
        len(class_names),
        class_names
    )
    yaml_path = Path(OUT_ROOT, "data.yaml")
    with open(yaml_path, "w", encoding="utf-8") as f:
        f.write(data_yaml)
    print("\nWrote:", yaml_path)
    print(data_yaml)

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

Лістинг Л.2 — Програмний модуль розпізнавання

```
import time
from pathlib import Path
import sqlite3
from datetime import datetime
import cv2
from ultralytics import YOLO

# ---------- SETTINGS ----------
MODEL_PATH = r"C:\ml\helmet\best.pt"

INTERVAL  = 0.5   # seconds between inferences
IMG_SIZE  = 640
CONF_TH   = 0.30  # global confidence threshold

WINDOW_NAME = "Helmet monitor (q/Esc to quit)"

COLOR_HELMET   = (0, 200, 0)   # green
COLOR_NOHELMET = (0,   0, 200) # red
COLOR_OTHER    = (200,150,  0) # orange
COLOR_TEXT     = (255,255,255)
COLOR_UI       = (50, 255, 255)

def draw_label(img, text, x1, y1, color):
    (tw, th), bl = cv2.getTextSize(text, cv2.FONT_HERSHEY_SIMPLEX, 0.6, 2)
    cv2.rectangle(img, (x1, y1 - th - 8), (x1 + tw + 6, y1), color, -1)
    cv2.putText(img, text, (x1 + 3, y1 - 6),
                cv2.FONT_HERSHEY_SIMPLEX, 0.6, COLOR_TEXT, 2)
```

```
def init_db(db_path: str):
    """Create SQLite DB and events table if not exists."""
    conn = sqlite3.connect(db_path)
    cur = conn.cursor()
    cur.execute("""
        CREATE TABLE IF NOT EXISTS events (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            ts TEXT NOT NULL,
            helmet_count INTEGER NOT NULL,
            nohelmet_count INTEGER NOT NULL
        );
    """)
    conn.commit()
    return conn

def main():
    # ---- load model ----
    if not Path(MODEL_PATH).exists():
        raise FileNotFoundError(f"Weights not found: {MODEL_PATH}")

    model = YOLO(MODEL_PATH)
    names = {int(k): str(v) for k, v in model.names.items()}
    print("model.names:", names)

    # resolve class ids for With Helmet / Without Helmet
    helmet_id = None
    nohelmet_id = None
    for cid, name in names.items():
        low = name.lower()
        if "with helmet" in low or ("helmet" in low and "without" not in low and "no" not in low):
            helmet_id = cid
        if "without helmet" in low or "no helmet" in low:
            nohelmet_id = cid

    # fallback to {0,1}
    if helmet_id is None and 0 in names:
        helmet_id = 0
    if nohelmet_id is None and 1 in names:
        nohelmet_id = 1

    print(f"helmet_id = {helmet_id}, nohelmet_id = {nohelmet_id}")

    # ---- open camera ----
    cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)
    if not cap.isOpened():
        cap = cv2.VideoCapture(1, cv2.CAP_DSHOW)
    if not cap.isOpened():
        raise RuntimeError("Camera not opened. Try different index or check permissions.")

    cap.set(cv2.CAP_PROP_FRAME_WIDTH,  960)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 540)
```

```
cv2.namedWindow(WINDOW_NAME, cv2.WINDOW_NORMAL)
cv2.resizeWindow(WINDOW_NAME, 960, 540)

# ---- init DB ----
db_path = r"C:\ml\helmet\helmet_events.db"
db_conn = init_db(db_path)
db_cur = db_conn.cursor()

last_t = 0.0
last_result = None

# global counters for operator
total_helmet = 0
total_nohelmet = 0

try:
    while True:
        ok, frame = cap.read()
        if not ok:
            break

        now = time.time()

        # run inference every INTERVAL seconds
        if now - last_t >= INTERVAL:
            last_t = now
            result = model.predict(
                frame,
                imgsz=IMG_SIZE,
                conf=CONF_TH,
                verbose=False
            )[0]
            last_result = result

            # counts per current inference
            frame_helmet = 0
            frame_nohelmet = 0

            if last_result is not None and last_result.boxes is not None:
                boxes = last_result.boxes
                xyxy = boxes.xyxy.cpu().numpy()
                clss = boxes.cls.cpu().numpy().astype(int)
                confs = boxes.conf.cpu().numpy()

                for box, cid, cf in zip(xyxy, clss, confs):
                    if cf < CONF_TH:
                        continue
                    if cid == helmet_id:
                        frame_helmet += 1
                    elif cid == nohelmet_id:
                        frame_nohelmet += 1
```

```
            # update global counters
            total_helmet += frame_helmet
            total_nohelmet += frame_nohelmet

            # log to DB only if there is at least one detection
            if frame_helmet > 0 or frame_nohelmet > 0:
                ts = datetime.now().isoformat(timespec="seconds")
                db_cur.execute(
                    "INSERT INTO events (ts, helmet_count, nohelmet_count) VALUES (?, ?, ?)",
                    (ts, frame_helmet, frame_nohelmet)
                )
                db_conn.commit()
                print(f"[DB] {ts}  Helmet={frame_helmet}, NoHelmet={frame_nohelmet}")

            # save per-frame counts to attach into UI
            current_frame_helmet = frame_helmet
            current_frame_nohelmet = frame_nohelmet
        else:
            # if no new inference yet, just show zeros for "per 0.5s"
            current_frame_helmet = 0
            current_frame_nohelmet = 0

        # draw detections if any (using last_result)
        if last_result is not None and last_result.boxes is not None:
            boxes = last_result.boxes
            xyxy = boxes.xyxy.cpu().numpy()
            clss = boxes.cls.cpu().numpy().astype(int)
            confs = boxes.conf.cpu().numpy()

            for box, cid, cf in zip(xyxy, clss, confs):
                if cf < CONF_TH:
                    continue

                x1, y1, x2, y2 = map(int, box)
                name = names.get(cid, str(cid))

                if cid == helmet_id:
                    color = COLOR_HELMET
                    text = f"Helmet {cf:.2f}"
                elif cid == nohelmet_id:
                    color = COLOR_NOHELMET
                    text = f"No helmet {cf:.2f}"
                else:
                    color = COLOR_OTHER
                    text = f"{name} {cf:.2f}"

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

        # ---- operator UI text ----
        # status line
```

```
            cv2.putText(frame, f"interval {INTERVAL:.2f}s  (q/Esc to quit)",
                        (10, 25), cv2.FONT_HERSHEY_SIMPLEX, 0.6, COLOR_UI, 2)

            # current frame counts
            cv2.putText(frame,
                        f"Frame: Helmet={current_frame_helmet}  No helmet={current_frame_nohelmet}",
                        (10, 55), cv2.FONT_HERSHEY_SIMPLEX, 0.6, COLOR_UI, 2)

            # total counts
            cv2.putText(frame,
                        f"Total: Helmet={total_helmet}  No helmet={total_nohelmet}",
                        (10, 85), cv2.FONT_HERSHEY_SIMPLEX, 0.6, COLOR_UI, 2)

            cv2.imshow(WINDOW_NAME, frame)

            key = cv2.waitKey(1) & 0xFF
            if key in (ord('q'), 27):  # q or Esc
                break
            if cv2.getWindowProperty(WINDOW_NAME, cv2.WND_PROP_VISIBLE) < 1:
                break

    finally:
        cap.release()
        cv2.destroyAllWindows()
        db_conn.close()

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

Лістинг Л.3 — Програмний модуль аналітики

```
import argparse
import sqlite3
from datetime import datetime
from pathlib import Path

import matplotlib.pyplot as plt

DB_PATH = r"C:\ml\helmet\helmet_events.db"

def connect_db():
    path = Path(DB_PATH)
    if not path.exists():
        raise FileNotFoundError(f"Database not found: {path}")
    return sqlite3.connect(str(path))

def summary_all(conn):
    cur = conn.cursor()

    # Загальна статистика за весь час
    cur.execute("""
```

```
        SELECT
            IFNULL(SUM(helmet_count), 0),
            IFNULL(SUM(nohelmet_count), 0)
        FROM events;
    """)
    total_helmet, total_nohelmet = cur.fetchone()

    print("=== Total statistics (all time) ===")
    print(f"Helmet:     {total_helmet}")
    print(f"No helmet:  {total_nohelmet}")
    print()

    # Статистика по днях
    cur.execute("""
        SELECT
            date(ts) AS d,
            SUM(helmet_count) AS sum_helmet,
            SUM(nohelmet_count) AS sum_nohelmet
        FROM events
        GROUP BY date(ts)
        ORDER BY d;
    """)
    rows = cur.fetchall()
    if not rows:
        print("No data in DB.")
        return

    dates = [r[0] for r in rows]
    helmet_per_day = [r[1] for r in rows]
    nohelmet_per_day = [r[2] for r in rows]

    # Текстова табличка
    print("Date       | Helmet | No helmet")
    print("-----------------------------")
    for d, h, nh in rows:
        print(f"{d} | {h:6d} | {nh:9d}")

    # Графік: порушення по днях
    plt.figure(figsize=(10, 5))
    plt.plot(dates, helmet_per_day, marker="o", label="Helmet")
    plt.plot(dates, nohelmet_per_day, marker="o", label="No helmet")
    plt.xlabel("Date")
    plt.ylabel("Detections per day")
    plt.title("Helmet / No helmet statistics (per day, all time)")
    plt.xticks(rotation=45, ha="right")
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
    plt.show()

def summary_today(conn):
    cur = conn.cursor()
```

```
# Сьогоднішня дата (по локальному часу)
today_str = datetime.now().date().isoformat()
print(f"=== Statistics for today ({today_str}) ===")

# Загальні суми за сьогодні
cur.execute("""
   SELECT
      IFNULL(SUM(helmet_count), 0),
      IFNULL(SUM(nohelmet_count), 0)
   FROM events
   WHERE date(ts) = ?;
""", (today_str,))
total_helmet, total_nohelmet = cur.fetchone()

print(f"Helmet:     {total_helmet}")
print(f"No helmet:  {total_nohelmet}")
print()

# Статистика по годинах (0..23)
cur.execute("""
   SELECT
      strftime('%H', ts) AS hour,
      SUM(helmet_count) AS sum_helmet,
      SUM(nohelmet_count) AS sum_nohelmet
   FROM events
   WHERE date(ts) = ?
   GROUP BY hour
   ORDER BY hour;
""", (today_str,))
rows = cur.fetchall()
if not rows:
   print("No data for today yet.")
   return

hours = [int(r[0]) for r in rows]
helmet_per_hour = [r[1] for r in rows]
nohelmet_per_hour = [r[2] for r in rows]

print("Hour | Helmet | No helmet")
print("-------------------------")
for h, hh, nh in zip(hours, helmet_per_hour, nohelmet_per_hour):
   print(f"{h:02d}   | {hh:6d} | {nh:9d}")

# Графік: порушення по годинах за сьогодні
plt.figure(figsize=(10, 5))
plt.bar([h - 0.2 for h in hours], helmet_per_hour, width=0.4, label="Helmet")
plt.bar([h + 0.2 for h in hours], nohelmet_per_hour, width=0.4, label="No helmet")
plt.xlabel("Hour of day")
plt.ylabel("Detections per hour")
plt.title(f"Helmet / No helmet statistics for {today_str} (per hour)")
plt.xticks(hours)
```

```
    plt.grid(axis="y")
    plt.legend()
    plt.tight_layout()
    plt.show()

def main():
    parser = argparse.ArgumentParser(
        description="Analytics for helmet_events.db (helmet / no helmet statistics)."
    )
    parser.add_argument(
        "--mode",
        choices=["today", "all"],
        default="today",
        help="Analytics mode: 'today' (default) or 'all' (all time per day)."
    )
    args = parser.parse_args()

    conn = connect_db()
    try:
        if args.mode == "today":
            summary_today(conn)
        else:
            summary_all(conn)
    finally:
        conn.close()

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

Лістинг Л.4 — інтерфейс користувача

```
import sys
import time
import sqlite3
from pathlib import Path
from datetime import datetime

import cv2
from ultralytics import YOLO

from PySide6.QtCore import Qt, QThread, Signal
from PySide6.QtGui import QImage, QPixmap
from PySide6.QtWidgets import (
    QApplication, QMainWindow, QWidget, QLabel, QPushButton, QFileDialog,
    QHBoxLayout, QVBoxLayout, QGridLayout, QGroupBox, QTableWidget,
    QTableWidgetItem, QMessageBox, QSpinBox
)

# ====== SETTINGS ======
MODEL_PATH = r"C:\ml\helmet\best.pt"
DB_PATH    = r"C:\ml\helmet\helmet_events.db"
```

```
INTERVAL = 0.5
IMG_SIZE = 640
CONF_TH  = 0.30

COLOR_HELMET   = (0, 200, 0)
COLOR_NOHELMET = (0, 0, 200)
COLOR_OTHER    = (200, 150, 0)
COLOR_TEXT     = (255, 255, 255)


def draw_label(img, text, x1, y1, color):
    (tw, th), bl = cv2.getTextSize(text, cv2.FONT_HERSHEY_SIMPLEX, 0.6, 2)
    cv2.rectangle(img, (x1, y1 - th - 8), (x1 + tw + 6, y1), color, -1)
    cv2.putText(img, text, (x1 + 3, y1 - 6),
                cv2.FONT_HERSHEY_SIMPLEX, 0.6, COLOR_TEXT, 2)


def init_db(db_path: str):
    conn = sqlite3.connect(db_path)
    cur = conn.cursor()
    cur.execute("""
        CREATE TABLE IF NOT EXISTS events (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            ts TEXT NOT NULL,
            helmet_count INTEGER NOT NULL,
            nohelmet_count INTEGER NOT NULL
        );
    """)
    conn.commit()
    conn.close()


def fetch_last_events(db_path: str, limit: int = 50):
    conn = sqlite3.connect(db_path)
    cur = conn.cursor()
    cur.execute("SELECT ts, helmet_count, nohelmet_count FROM events ORDER BY id DESC LIMIT
?", (limit,))
    rows = cur.fetchall()
    conn.close()
    return list(reversed(rows))


class DetectorWorker(QThread):
    frame_ready = Signal(QImage)
    stats_ready = Signal(int, int, int, int)  # frame_helmet, frame_nohelmet, total_helmet, total_nohelmet
    event_logged = Signal(str, int, int)      # ts, helmet_count, nohelmet_count
    status = Signal(str)

    def __init__(self, interval: float, imgsz: int, conf_th: float):
        super().__init__()
        self.interval = interval
```

```
        self.imgsz = imgsz
        self.conf_th = conf_th

        self._running = False
        self._reset_requested = False

        self.model = None
        self.names = {}
        self.helmet_id = None
        self.nohelmet_id = None

        self.total_helmet = 0
        self.total_nohelmet = 0

        self.last_t = 0.0
        self.last_result = None

        self.cap = None

    def stop(self):
        self._running = False

    def request_reset(self):
        self._reset_requested = True

    def _resolve_class_ids(self):
        # (повторює твою логіку)
        helmet_id = None
        nohelmet_id = None

        for cid, name in self.names.items():
            low = name.lower()
            if "with helmet" in low or ("helmet" in low and "without" not in low and "no" not in low):
                helmet_id = cid
            if "without helmet" in low or "no helmet" in low:
                nohelmet_id = cid

        if helmet_id is None and 0 in self.names:
            helmet_id = 0
        if nohelmet_id is None and 1 in self.names:
            nohelmet_id = 1

        self.helmet_id = helmet_id
        self.nohelmet_id = nohelmet_id

    def _open_camera(self):
        # як у тебе: пробує 0, потім 1
        cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)
        if not cap.isOpened():
            cap = cv2.VideoCapture(1, cv2.CAP_DSHOW)
        if not cap.isOpened():
            return None
```

```
        cap.set(cv2.CAP_PROP_FRAME_WIDTH, 960)
        cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 540)
        return cap

    def run(self):
        # ---- load model ----
        if not Path(MODEL_PATH).exists():
            self.status.emit(f"❌ Weights not found: {MODEL_PATH}")
            return

        self.status.emit("Завантаження моделі…")
        self.model = YOLO(MODEL_PATH)
        self.names = {int(k): str(v) for k, v in self.model.names.items()}
        self._resolve_class_ids()

        if self.helmet_id is None or self.nohelmet_id is None:
            self.status.emit("⚠ Не вдалося визначити ID класів helmet/nohelmet. Перевір model.names.")
        else:
            self.status.emit(f"Модель OK. helmet_id={self.helmet_id}, nohelmet_id={self.nohelmet_id}")

        # ---- open camera ----
        self.cap = self._open_camera()
        if self.cap is None:
            self.status.emit("❌ Камера не відкрилась (0/1).")
            return

        init_db(DB_PATH)
        db_conn = sqlite3.connect(DB_PATH)
        db_cur = db_conn.cursor()

        self._running = True
        self.last_t = 0.0
        self.last_result = None

        try:
            while self._running:
                ok, frame = self.cap.read()
                if not ok:
                    self.status.emit("⚠ Не вдалося зчитати кадр.")
                    break

                if self._reset_requested:
                    self.total_helmet = 0
                    self.total_nohelmet = 0
                    self._reset_requested = False

                now = time.time()

                # run inference every INTERVAL seconds
```

```
            if now - self.last_t >= self.interval:
                self.last_t = now
                self.last_result = self.model.predict(
                    frame,
                    imgsz=self.imgsz,
                    conf=self.conf_th,
                    verbose=False
                )[0]

                frame_helmet = 0
                frame_nohelmet = 0

                if self.last_result is not None and self.last_result.boxes is not None:
                    boxes = self.last_result.boxes
                    xyxy = boxes.xyxy.cpu().numpy()
                    clss = boxes.cls.cpu().numpy().astype(int)
                    confs = boxes.conf.cpu().numpy()

                    for box, cid, cf in zip(xyxy, clss, confs):
                        if cf < self.conf_th:
                            continue
                        if cid == self.helmet_id:
                            frame_helmet += 1
                        elif cid == self.nohelmet_id:
                            frame_nohelmet += 1

                self.total_helmet += frame_helmet
                self.total_nohelmet += frame_nohelmet

                # log to DB only if there is at least one detection
                if frame_helmet > 0 or frame_nohelmet > 0:
                    ts = datetime.now().isoformat(timespec="seconds")
                    db_cur.execute(
                        "INSERT INTO events (ts, helmet_count, nohelmet_count) VALUES (?, ?, ?)",
                        (ts, frame_helmet, frame_nohelmet)
                    )
                    db_conn.commit()
                    self.event_logged.emit(ts, frame_helmet, frame_nohelmet)

                self.stats_ready.emit(frame_helmet, frame_nohelmet, self.total_helmet,
self.total_nohelmet)
            else:
                # якщо інференсу ще не було — показуємо 0/0 для frame
                self.stats_ready.emit(0, 0, self.total_helmet, self.total_nohelmet)

            # draw detections using last_result
            if self.last_result is not None and self.last_result.boxes is not None:
                boxes = self.last_result.boxes
                xyxy = boxes.xyxy.cpu().numpy()
                clss = boxes.cls.cpu().numpy().astype(int)
                confs = boxes.conf.cpu().numpy()
```

```
                for box, cid, cf in zip(xyxy, clss, confs):
                    if cf < self.conf_th:
                        continue
                    x1, y1, x2, y2 = map(int, box)
                    name = self.names.get(cid, str(cid))

                    if cid == self.helmet_id:
                        color = COLOR_HELMET
                        text = f"Helmet {cf:.2f}"
                    elif cid == self.nohelmet_id:
                        color = COLOR_NOHELMET
                        text = f"No helmet {cf:.2f}"
                    else:
                        color = COLOR_OTHER
                        text = f"{name} {cf:.2f}"

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

                # convert to QImage
                rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                h, w, ch = rgb.shape
                qimg = QImage(rgb.data, w, h, ch * w, QImage.Format_RGB888).copy()
                self.frame_ready.emit(qimg)

        finally:
            try:
                db_conn.close()
            except Exception:
                pass
            try:
                if self.cap is not None:
                    self.cap.release()
            except Exception:
                pass
            self.status.emit("Зупинено.")


class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Helmet Monitor — 1 вікно (камера + статистика + журнал)")
        self.resize(1200, 720)

        # UI
        root = QWidget()
        self.setCentralWidget(root)
        layout = QHBoxLayout(root)

        # left video
        self.video = QLabel("Натисни Start")
        self.video.setAlignment(Qt.AlignCenter)
```

```
self.video.setStyleSheet("background:#111; color:#ddd; border-radius:8px;")
self.video.setMinimumSize(800, 600)
layout.addWidget(self.video, 2)

# right panel
right = QVBoxLayout()
layout.addLayout(right, 1)

gb_ctrl = QGroupBox("Керування")
g = QGridLayout(gb_ctrl)

self.btn_start = QPushButton("▶ Start")
self.btn_stop = QPushButton("■ Stop")
self.btn_reset = QPushButton("↺ Reset totals")
self.btn_snapshot = QPushButton("📸 Snapshot")
self.btn_export = QPushButton("⤓ Export DB → CSV")

self.spin_interval = QSpinBox()
self.spin_interval.setRange(1, 5000)
self.spin_interval.setValue(int(INTERVAL * 1000))

g.addWidget(self.btn_start, 0, 0)
g.addWidget(self.btn_stop, 0, 1)
g.addWidget(QLabel("Interval (ms):"), 1, 0)
g.addWidget(self.spin_interval, 1, 1)
g.addWidget(self.btn_reset, 2, 0, 1, 2)
g.addWidget(self.btn_snapshot, 3, 0, 1, 2)
g.addWidget(self.btn_export, 4, 0, 1, 2)

right.addWidget(gb_ctrl)

gb_stats = QGroupBox("Статистика")
sg = QGridLayout(gb_stats)

self.lbl_frame = QLabel("Frame: Helmet=0  NoHelmet=0")
self.lbl_total = QLabel("Total: Helmet=0  NoHelmet=0")
self.lbl_db = QLabel(f"DB: {DB_PATH}")
self.lbl_status = QLabel("Стан: готово")

sg.addWidget(self.lbl_frame, 0, 0)
sg.addWidget(self.lbl_total, 1, 0)
sg.addWidget(self.lbl_db, 2, 0)
sg.addWidget(self.lbl_status, 3, 0)

right.addWidget(gb_stats)

gb_log = QGroupBox("Журнал (останні 50 записів)")
v = QVBoxLayout(gb_log)
self.table = QTableWidget(0, 3)
self.table.setHorizontalHeaderLabels(["ts", "helmet", "nohelmet"])
self.table.horizontalHeader().setStretchLastSection(True)
self.table.setEditTriggers(QTableWidget.NoEditTriggers)
```

```
        self.table.setSelectionBehavior(QTableWidget.SelectRows)
        v.addWidget(self.table)
        right.addWidget(gb_log, 1)

        # worker
        self.worker = DetectorWorker(interval=INTERVAL, imgsz=IMG_SIZE, conf_th=CONF_TH)
        self.worker.frame_ready.connect(self.on_frame)
        self.worker.stats_ready.connect(self.on_stats)
        self.worker.event_logged.connect(self.on_event_logged)
        self.worker.status.connect(self.on_status)

        # buttons
        self.btn_start.clicked.connect(self.start_worker)
        self.btn_stop.clicked.connect(self.stop_worker)
        self.btn_reset.clicked.connect(self.reset_totals)
        self.btn_snapshot.clicked.connect(self.snapshot)
        self.btn_export.clicked.connect(self.export_csv)
        self.spin_interval.valueChanged.connect(self.change_interval)

        self.btn_stop.setEnabled(False)

        # load initial log table (if DB exists)
        if Path(DB_PATH).exists():
            init_db(DB_PATH)
            self.reload_table()

    def start_worker(self):
        if self.worker.isRunning():
            return
        init_db(DB_PATH)
        self.worker.interval = self.spin_interval.value() / 1000.0
        self.worker.start()
        self.btn_start.setEnabled(False)
        self.btn_stop.setEnabled(True)
        self.on_status("Запущено…")

    def stop_worker(self):
        if self.worker.isRunning():
            self.worker.stop()
            self.worker.wait(1500)
        self.btn_start.setEnabled(True)
        self.btn_stop.setEnabled(False)

    def reset_totals(self):
        if self.worker.isRunning():
            self.worker.request_reset()
        self.lbl_frame.setText("Frame: Helmet=0  NoHelmet=0")
        self.lbl_total.setText("Total: Helmet=0  NoHelmet=0")

    def change_interval(self, ms: int):
        if self.worker.isRunning():
            self.worker.interval = ms / 1000.0
```

```
def on_frame(self, img: QImage):
    pix = QPixmap.fromImage(img).scaled(
        self.video.width(), self.video.height(),
        Qt.KeepAspectRatio, Qt.SmoothTransformation
    )
    self.video.setPixmap(pix)

def on_stats(self, fh: int, fn: int, th: int, tn: int):
    self.lbl_frame.setText(f"Frame: Helmet={fh}  NoHelmet={fn}")
    self.lbl_total.setText(f"Total: Helmet={th}  NoHelmet={tn}")

def on_event_logged(self, ts: str, h: int, n: int):
    # додати рядок в таблицю (і тримати 50)
    row = self.table.rowCount()
    self.table.insertRow(row)
    for c, val in enumerate([ts, str(h), str(n)]):
        it = QTableWidgetItem(val)
        it.setTextAlignment(Qt.AlignCenter)
        self.table.setItem(row, c, it)

    if self.table.rowCount() > 50:
        self.table.removeRow(0)

    self.table.scrollToBottom()

def on_status(self, text: str):
    self.lbl_status.setText(f"Стан: {text}")

def reload_table(self):
    rows = fetch_last_events(DB_PATH, limit=50)
    self.table.setRowCount(0)
    for ts, h, n in rows:
        r = self.table.rowCount()
        self.table.insertRow(r)
        for c, val in enumerate([ts, str(h), str(n)]):
            it = QTableWidgetItem(val)
            it.setTextAlignment(Qt.AlignCenter)
            self.table.setItem(r, c, it)
    self.table.scrollToBottom()

def snapshot(self):
    if self.video.pixmap() is None:
        return
    path, _ = QFileDialog.getSaveFileName(self, "Save snapshot", "snapshot.png", "PNG (*.png)")
    if not path:
        return
    self.video.pixmap().save(path, "PNG")

def export_csv(self):
    if not Path(DB_PATH).exists():
        QMessageBox.warning(self, "DB", "База не знайдена.")
```

```
            return

        path, _ = QFileDialog.getSaveFileName(self, "Export CSV", "events.csv", "CSV (*.csv)")
        if not path:
            return

        conn = sqlite3.connect(DB_PATH)
        cur = conn.cursor()
        cur.execute("SELECT ts, helmet_count, nohelmet_count FROM events ORDER BY id ASC")
        rows = cur.fetchall()
        conn.close()

        with open(path, "w", encoding="utf-8") as f:
            f.write("ts,helmet_count,nohelmet_count\n")
            for ts, h, n in rows:
                f.write(f"{ts},{h},{n}\n")

        QMessageBox.information(self, "Export", f"Готово:\n{path}")

    def closeEvent(self, event):
        self.stop_worker()
        super().closeEvent(event)


def main():
    app = QApplication(sys.argv)
    w = MainWindow()
    w.show()
    sys.exit(app.exec())


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