РЕФЕРАТ

ІНТЕЛЕКТУАЛЬНА СИСТЕМА ПІДТРИМКИ ВОДІЯ, КОМП'ЮТЕРНИЙ ЗІР, ДОРОЖНІ ЗНАКИ, ДЕТЕКЦІЯ ОБ'ЄКТІВ, YOLOV8, CVAT, OPENCV, PYTHON, PYSIDE6.

Пояснювальна записка: 64 с., 6 рис., 8 дод., 20 джерел.

Метою кваліфікаційної роботи є розробка інтелектуальної системи підтримки водія для автоматичного розпізнавання дорожніх знаків у відеопотоці з використанням комп'ютерного зору та нейронних мереж.

У роботі розглянуто проблеми безпеки дорожнього руху, системи підтримки водія, методи комп'ютерного зору та моделі детекції об'єктів. Обґрунтовано вибір YOLOv8 як моделі, придатної для розпізнавання дорожніх знаків у режимі, наближеному до реального часу.

Для підготовки датасету використано CVAT, розгорнутий у Docker-середовищі. Розмітка виконувалась за класами: crosswalk, children_on_road, speed_limit, speed_bump, turn_sign, bus_stop, patrol, parking та parking_ban. Після цього анотації було експортовано та підготовлено до навчання моделі YOLOv8.

Програмну реалізацію виконано мовою Python із використанням Ultralytics YOLOv8, OpenCV та PySide6. Розроблено модуль навчання, модуль валідації, графічний інтерфейс для перегляду відео з результатами детекції та вікно текстових підказок для водія.

Проведене тестування показало, що система здатна виявляти основні категорії дорожніх знаків, однак якість роботи залежить від збалансованості датасету, освітлення, відстані до об'єкта та якості відеозапису.

Розроблена система може бути використана як прототип інтелектуальної системи підтримки водія та основа для подальшого розвитку засобів аналізу дорожньої ситуації.

ABSTRACT

INTELLIGENT DRIVER ASSISTANCE SYSTEM, COMPUTER VISION, TRAFFIC SIGNS, OBJECT DETECTION, YOLOV8, CVAT, OPENCV, PYTHON, PYSIDE6.

Thesis explanatory note: 64 pages, 6 figures, 8 appendices, 20 references.

The purpose of the qualification work is to develop an intelligent driver assistance system for automatic traffic sign recognition in a video stream using computer vision methods and neural networks.

The thesis considers road safety problems, driver assistance systems, computer vision methods, and object detection models. The choice of YOLOv8 is justified as a model suitable for traffic sign recognition in near real-time mode.

The dataset was prepared using the CVAT annotation platform deployed in a Docker environment. Traffic signs were annotated according to four classes: crosswalk, children_on_road, speed_limit, speed_bump, turn_sign, bus_stop, patrol, parking and parking_ban. The annotations were then exported and prepared for YOLOv8 model training.

The software implementation was developed in Python using Ultralytics YOLOv8, OpenCV, and PySide6. The system includes a training module, a validation module, a graphical video viewer with detection results, and a separate window with textual driving hints.

Testing showed that the system is capable of detecting the main categories of traffic signs. However, detection quality depends on dataset balance, lighting conditions, object distance, and video quality.

The developed system can be used as a prototype of an intelligent driver assistance system and as a basis for further development of road scene analysis tools.

ЗМІСТ

ВСТУП

Безпека дорожнього руху залишається однією з важливих соціальних і технічних завдань сучасного суспільства. За даними Всесвітньої організації охорони здоров'я, кількість щорічних смертей унаслідок дорожньо-транспортних пригод у світі становить близько 1,19 млн осіб [1]. Це свідчить про те, що, незважаючи на розвиток транспортної інфраструктури, систем регулювання руху та засобів активної безпеки автомобілів, проблема зниження аварійності залишається актуальною.

Однією з причин виникнення дорожньо-транспортних пригод є високе інформаційне навантаження на водія. Під час руху водій повинен одночасно контролювати швидкість транспортного засобу, дорожні знаки, розмітку, сигнали світлофорів, поведінку інших учасників руху та зміну дорожньої ситуації. У складних умовах, зокрема під час інтенсивного руху, поганої видимості або втоми, зростає ймовірність того, що водій може несвоєчасно помітити важливий дорожній знак або неправильно оцінити ситуацію.

У зв'язку з цим особливого значення набувають системи підтримки водія. Європейська комісія розглядає advanced driver assistance systems як технології, що можуть надавати водієві допомогу в дорожньому середовищі з урахуванням можливостей і обмежень людини [2]. Такі системи не замінюють водія повністю, але можуть зменшувати інформаційне навантаження, своєчасно попереджати про потенційні небезпеки та підвищувати рівень безпеки руху.

Одним із перспективних напрямів розвитку систем підтримки водія є використання комп'ютерного зору та нейронних мереж для аналізу відеопотоку з камери. Такий підхід дозволяє автоматично виявляти дорожні знаки, транспортні засоби та інші об'єкти дорожньої інфраструктури. Для задачі розпізнавання дорожніх знаків це має особливе значення, оскільки

дорожні знаки містять критично важливу інформацію щодо обмежень, пріоритету руху, небезпечних ділянок дороги та рекомендованих дій водія.

Таким чином, розробка інтелектуальної системи підтримки водія, здатної автоматично розпізнавати дорожні знаки у відеопотоці та надавати водієві допоміжну інформацію, є актуальною задачею. Така система може бути використана як прототип програмного засобу для аналізу дорожньої ситуації та подальшого розвитку технологій підтримки водія.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Мета та завдання дослідження

Метою даної дипломної роботи є розробка інтелектуальної системи підтримки водія, здатної виконувати розпізнавання дорожніх знаків у відеопотоці в режимі реального часу із використанням методів комп'ютерного зору та машинного навчання.

Для досягнення поставленої мети у роботі необхідно вирішити такі завдання:

– проаналізувати сучасні методи та засоби розпізнавання дорожніх знаків у системах підтримки водія;

– дослідити можливості використання нейронних мереж для задач детекції об'єктів;

– виконати підготовку та розмітку датасету дорожніх знаків за допомогою системи CVAT;

– реалізувати процес конвертації анотацій у формат, придатний для навчання моделі YOLOv8;

– провести навчання моделі детекції дорожніх знаків на підготовленому датасеті;

– розробити програмний інтерфейс для перегляду відео та відображення результатів детекції;

– реалізувати механізм текстових підказок для водія на основі виявлених дорожніх знаків;

– виконати тестування роботи системи на різних відеозаписах дорожнього руху та оцінити якість роботи моделі.

1.2 Об’єкт та предмет дослідження

Об’єктом дослідження є процес автоматизованого розпізнавання дорожніх знаків у відеопотоці під час руху транспортного засобу.

Предметом дослідження є методи комп’ютерного зору, алгоритми детекції об’єктів та програмні засоби реалізації інтелектуальних систем підтримки водія на основі нейронної мережі YOLOv8.

1.3 Проблеми безпеки дорожнього руху

Одним із важливих завдань сучасного суспільства є забезпечення безпеки дорожнього руху. З розвитком транспортної інфраструктури та постійним збільшенням кількості автомобілів на дорогах значно ускладнюються умови керування транспортними засобами. Водій під час руху змушений постійно аналізувати велику кількість інформації: дорожні знаки, сигнали світлофорів, дорожню розмітку, поведінку інших учасників дорожнього руху, швидкість транспортного потоку та можливі небезпечні ситуації. Високе інформаційне навантаження підвищує ризик виникнення помилок, які можуть призвести до ДТП.

Однією з головних причин аварій на дорогах залишається людський фактор. Навіть досвідчені водії можуть припускатися помилок через втому, неуважність, стрес або перевантаження інформацією. Під час тривалого перебування за кермом знижується концентрація уваги, погіршується швидкість реакції та здатність швидко оцінювати дорожню обстановку.

Особливо небезпечними є ситуації, коли водій не помічає дорожній знак, запізно реагує на зміну дорожньої ситуації або неправильно оцінює дії інших учасників руху.

Суттєвою проблемою сучасного дорожнього середовища є необхідність прийняття рішень у дуже короткий проміжок часу. Під час руху автомобіля дорожня ситуація може змінюватися за лічені секунди. Наприклад, поява пішохода на проїжджій частині, різке гальмування автомобіля попереду або

зміна дорожніх умов вимагають від водія миттєвої реакції. У подібних ситуаціях навіть незначна затримка може мати серйозні наслідки.

Окрему небезпеку становить рух у великих містах із високою інтенсивністю транспортного потоку. У міських умовах водій одночасно взаємодіє з великою кількістю об’єктів та джерел інформації. Значна кількість дорожніх знаків, складні перехрестя, велика кількість пішоходів та постійна зміна дорожньої ситуації створюють додаткове навантаження на увагу водія. У результаті підвищується ймовірність пропуску важливої інформації або неправильного сприйняття дорожньої обстановки.

Ще одним важливим фактором є вплив зовнішніх умов на здатність водія контролювати ситуацію на дорозі. Якість сприйняття дорожньої інформації може погіршуватись через недостатнє освітлення, дощ, туман, сніг або поганий стан дорожнього покриття. У таких умовах водієві стає складніше своєчасно помічати дорожні знаки та оцінювати відстань до об’єктів. Це особливо небезпечно під час руху на високій швидкості або в умовах обмеженої видимості.

Проблема безпеки дорожнього руху також пов’язана зі зростанням кількості транспортних засобів. Збільшення щільності транспортного потоку призводить до виникнення заторів, ускладнення маневрування та підвищення ризику виникнення аварійних ситуацій. У складних дорожніх умовах водій змушений постійно контролювати положення інших транспортних засобів та прогнозувати їхні дії, що створює додаткове психологічне навантаження.

Для зменшення кількості дорожньо-транспортних пригод у сучасних автомобілях активно впроваджуються різноманітні системи допомоги водієві. Такі системи спрямовані на підвищення рівня безпеки шляхом автоматичного контролю окремих аспектів дорожньої ситуації. Вони можуть попереджати водія про потенційні небезпеки, допомагати під час маневрування або контролювати дотримання правил дорожнього руху.

Особливу роль у розвитку подібних систем відіграють технології комп’ютерного зору та машинного навчання. Завдяки використанню сучасних алгоритмів аналізу зображень стало можливим автоматичне розпізнавання дорожніх знаків, транспортних засобів та інших об’єктів дорожньої інфраструктури. Це дозволяє створювати інтелектуальні системи підтримки водія, здатні аналізувати дорожню ситуацію в режимі реального часу та своєчасно інформувати водія про потенційні ризики.

Використання інтелектуальних систем підтримки водія дозволяє знизити вплив людського фактора та підвищити рівень безпеки дорожнього руху. Подібні системи не замінюють водія повністю, однак можуть суттєво допомагати під час керування транспортним засобом, зменшуючи інформаційне навантаження та підвищуючи швидкість реагування на небезпечні ситуації.

1.4 Системи підтримки водія

Системи підтримки водія є класом програмно-апаратних рішень, призначених для допомоги водієві під час керування транспортним засобом. Їх основне завдання пAtomegaолягає не в повній заміні людини, а в підвищенні безпеки руху за рахунок додаткового контролю дорожньої ситуації, попередження про потенційні ризики та зменшення ймовірності помилки водія.

У сучасних автомобілях такі системи можуть виконувати різні функції залежно від призначення та рівня складності. До них належать:

- адаптивний круїз-контроль;
- система автоматичного екстреного гальмування;
- контроль дистанції;
- попередження про виїзд зі смуги руху;
- моніторинг сліпих зон;
- контроль стану водія;
- розпізнавання дорожніх знаків.

Європейська комісія розглядає “advanced driver assistance systems” як засоби, що допомагають водієві під час руху та спрямовані на підвищення безпеки транспортних засобів [2].

За характером роботи системи підтримки водія можна умовно поділити на інформаційні, попереджувальні та системи активного втручання. Інформаційні системи надають водієві додаткові дані про дорожню ситуацію, наприклад про обмеження швидкості або наявність дорожнього знака. Попереджувальні системи повідомляють про можливу небезпеку, таку як скорочення дистанції або вихід зі смуги руху. Системи активного втручання можуть частково впливати на керування транспортним засобом, наприклад виконувати автоматичне гальмування у критичній ситуації.

У межах даної роботи основна увага приділяється інформаційно-попереджувальному підходу. Розроблювана система не втручається безпосередньо в керування автомобілем, а виконує аналіз відеопотоку, визначає дорожні знаки та відображає водієві допоміжну інформацію. Такий підхід є доцільним для прототипу, оскільки дозволяє перевірити працездатність алгоритмів комп’ютерного зору без необхідності інтеграції з реальними системами керування транспортним засобом.

Окреме місце серед систем підтримки водія займають системи розпізнавання дорожніх знаків. Дорожні знаки містять важливу інформацію, щодо обмежень швидкості, пріоритету руху, напрямків, заборон, попереджень про небезпечні ділянки та інших умов руху. Якщо водій не помічає знак або помічає його занадто пізно, це може призвести до порушення правил дорожнього руху або створення небезпечної ситуації.

Таким чином, системи підтримки водія є важливим елементом сучасних транспортних технологій. У контексті цієї роботи вони розглядаються, як програмні засоби, що аналізують дорожню сцену за допомогою відеопотоку та надають водієві додаткову інформацію для своєчасного реагування на дорожні знаки.

1.5 Методи комп’ютерного зору

Комп’ютерний зір є напрямом штучного інтелекту, що займається автоматичним аналізом зображень і відеопотоку з метою отримання корисної інформації про об’єкти, їх розташування та взаємодію у сцені. У транспортних системах комп’ютерний зір використовується для аналізу дорожньої обстановки, зокрема для виявлення дорожніх знаків, транспортних засобів, пішоходів, дорожньої розмітки та інших елементів інфраструктури.

Основним джерелом даних у таких системах є камера, яка формує послідовність кадрів дорожньої сцени. Кожен кадр може бути оброблений окремо або в контексті відеопотоку. У задачах підтримки водія важливо не лише визначити наявність певного об’єкта, але й оцінити його положення у кадрі, оскільки від цього залежить своєчасність реакції системи та коректність відображення інформації для водія.

Традиційні методи комп’ютерного зору базувалися на використанні заздалегідь визначених ознак: контурів, кольору, форми, текстури, градієнтів або геометричних характеристик об’єкта. Наприклад, для дорожніх знаків могли враховуватись форми знаку, контрастність, колірна область або наявність характерних меж. Такі підходи можуть бути ефективними у контрольованих умовах, однак їхня надійність знижується при зміні освітлення, частковому перекритті знака, поганій якості відео або нестандартному ракурсі.

Сучасні підходи до комп’ютерного зору дедалі частіше ґрунтуються на методах машинного навчання та глибоких нейронних мережах. На відміну від класичних алгоритмів, нейронні мережі не потребують ручного задання всіх ознак об’єкта, а навчаються виділяти їх автоматично на основі розмічених даних. Це дозволяє підвищити стійкість системи до різних умов зйомки та зробити її придатнішою для роботи з реальними дорожніми сценами.

У задачі розпізнавання дорожніх знаків комп’ютерний зір виконує декілька послідовних функцій: отримання кадру з відеопотоку, виявлення

потенційних об’єктів, визначення їхнього класу та відображення результатів для користувача. У межах інтелектуальної системи підтримки водія ці результати можуть використовуватись не лише для візуалізації рамки навколо дорожнього знака, але й для формування додаткових текстових підказок.

Для систем підтримки водія особливо важливою є швидкість обробки відеопотоку. Якщо аналіз кадру виконується із суттєвою затримкою, інформація може втратити актуальність, оскільки дорожня ситуація змінюється дуже швидко. Саме тому в таких системах перевага надається методам, здатним забезпечити баланс між точністю розпізнавання та швидкодією.

У наукових роботах, присвячених розпізнаванню дорожніх знаків, підкреслюється, що складність цієї задачі пов’язана з малим розміром об’єктів у кадрі, зміною освітлення, різними відстанями до знаків, частковим перекриттям та різною якістю зображення [12]. Тому для практичної реалізації системи підтримки водія доцільно використовувати методи комп’ютерного зору, які здатні працювати з відеопотоком у змінних умовах дорожнього середовища.

Таким чином, методи комп’ютерного зору є основою для автоматичного аналізу дорожньої сцени. Вони дозволяють перейти від простого перегляду відео до автоматизованого виявлення важливих дорожніх об’єктів, що є необхідним етапом для створення інтелектуальної системи підтримки водія.

1.6 Нейронні мережі для детекції об’єктів

Нейронні мережі є одним із основних інструментів сучасного комп’ютерного зору. Вони використовуються для аналізу зображень, виявлення об’єктів, визначення їхнього положення у кадрі та класифікації. На відміну від класичних методів обробки зображень, де ознаки об’єкта задаються вручну, нейронні мережі формують внутрішні ознаки автоматично в процесі навчання на розмічених даних.

У задачах детекції об’єктів модель повинна одночасно вирішувати дві пов’язані задачі: визначити, який об’єкт присутній на зображенні, та встановити його розташування. Результатом роботи моделі є не лише клас об’єкта, а й координати області, у якій він знаходиться. Для кожного знайденого об’єкта модель формує набір параметрів:

$$B = (x, y, w, h, c, p_1, p_2, \ldots, p_n),$$

де x та y — координати центра обмежувальної рамки, w і h — її ширина та висота, c — рівень впевненості моделі у наявності об’єкта, а $p_1, p_2, \ldots, p_n$ — ймовірності належності об’єкта до відповідних класів.

У межах цієї роботи такими класами є crosswalk, children_on_road, speed_limit, speed_bump, turn_sign, bus_stop, patrol, parking та parking_ban. Наприклад, якщо модель виявляє дорожній знак, вона повинна визначити його положення у кадрі та віднести його до одного із зазначених класів. Після цього отримані координати використовуються для побудови рамки навколо об’єкта, а клас і рівень впевненості відображаються в інтерфейсі системи.

Основою багатьох сучасних моделей детекції є згорткові нейронні мережі. Згортковий шар аналізує зображення за допомогою фільтрів, які проходять по зображенню та виділяють певні ознаки. Математично операцію згортки можна подати у вигляді:

$$S(i,j) = \sum_m \sum_n I\,(i+m, j+n) \cdot K(m,n)$$

де I — вхідне зображення або карта ознак, K — ядро згортки, а $S(i,j)$ — значення нової карти ознак у точці (i,j).

На початкових шарах нейронна мережа зазвичай виділяє прості ознаки: контури, межі, переходи кольору та окремі геометричні елементи.

На глибших шарах ці ознаки об’єднуються у складніші структури, наприклад форму дорожнього знака, характерну кольорову область або комбінацію елементів, властиву певному класу. Саме завдяки цьому модель може розпізнавати об’єкти навіть тоді, коли вони мають різний розмір, розташування або освітлення.

Для задачі розпізнавання дорожніх знаків така властивість є особливо важливою. Один і той самий знак може виглядати по-різному залежно від відстані до камери, кута огляду, освітлення, якості відеозапису або часткового перекриття іншими об’єктами. У систематичному огляді методів розпізнавання дорожніх знаків на основі YOLO зазначається, що складність цієї задачі пов’язана з малим розміром знаків у кадрі, зміною умов зйомки, відстані та якості зображення [12]. Тому модель не повинна просто запам’ятовувати окремі зображення, а має навчитися узагальнювати характерні ознаки класів.

Процес навчання нейронної мережі полягає у поступовому налаштуванні її вагових коефіцієнтів. На вхід моделі подається зображення, для якого заздалегідь відомі правильні координати об’єктів та їхні класи. Модель виконує передбачення, після чого результат порівнюється з правильною розміткою. Різниця між передбаченням моделі та еталонними даними визначається за допомогою функції втрат.

У загальному вигляді функцію втрат для задачі детекції можна подати як суму декількох складових:

$$L = L_{box} + L_{cls} + L_{conf}$$

де L_{box} — помилка визначення координат обмежувальної рамки, L_{cls} — помилка класифікації об’єкта, а L_{conf} — помилка оцінки впевненості моделі.

Складова L_{conf} відповідає за те, наскільки точно модель визначила положення об’єкта. Якщо передбачена рамка сильно відрізняється від

розміченої вручну, значення цієї помилки зростає. Для оцінки збігу між передбаченою та правильною рамкою часто використовується показник IoU, тобто Intersection over Union [9]:

$$IoU = \frac{S_{intersection}}{S_{union}}$$

де $S_{intersection}$ — площа перетину передбаченої та правильної рамки, а S_{union} — площа їхнього об'єднання. Чим більше значення IoU, тим точніше модель визначила положення об'єкта.

Складова L_{cls} відповідає за правильність визначення класу. Наприклад, якщо дорожній знак класу warning модель помилково визначає як information, значення помилки класифікації збільшується. Складова L_{conf} пов'язана з тим, наскільки коректно модель оцінює наявність об'єкта в певній області зображення.

Після обчислення функції втрат виконується оновлення ваг нейронної мережі. Для цього використовується метод зворотного поширення помилки та оптимізаційний алгоритм, наприклад градієнтний спуск або його модифікації. У спрощеному вигляді оновлення ваг можна записати так:

$$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w}$$

де w_{old} — попереднє значення ваги, w_{new} — нове значення ваги, η — швидкість навчання, а $\frac{\partial L}{\partial w}$ — похідна функції втрат за відповідною вагою.

Це означає, що модель поступово змінює свої параметри так, щоб зменшити помилку на навчальних даних. Якщо модель неправильно визначила положення або клас об'єкта, функція втрат збільшується, а під час наступних ітерацій ваги коригуються. Після великої кількості таких ітерацій

модель починає краще виявляти характерні ознаки об’єктів і точніше виконувати детекцію.

У процесі навчання модель багаторазово проходить через навчальний набір даних. Один повний прохід через усі навчальні зображення називається — епохою. Протягом однієї епохи модель отримує на вхід усі зображення з навчальної вибірки, виконує передбачення для кожного з них, порівнює результат із правильною розміткою та коригує свої вагові коефіцієнти.

Кількість епох визначає, скільки разів модель буде повторно аналізувати навчальні дані. Якщо епох занадто мало, модель може не встигнути достатньо добре засвоїти характерні ознаки об’єктів. У такому випадку вона буде погано визначати дорожні знаки навіть на схожих зображеннях. Якщо ж кількість епох занадто велика, може виникнути перенавчання, коли модель надто добре запам’ятовує навчальні приклади, але гірше працює на нових відеозаписах.

Для задачі розпізнавання дорожніх знаків вибір кількості епох є важливим, оскільки модель повинна не просто запам’ятати конкретні кадри, а навчитися узагальнювати ознаки знаків: форму, колір, контури, розташування в кадрі та відмінність від схожих об’єктів. Саме тому під час навчання необхідно контролювати не лише результати на навчальній вибірці, але й якість роботи моделі на валідаційних даних.

Через це, епоха є одним із основних параметрів навчання нейронної мережі. Вона визначає кількість повторних проходів моделі через навчальний набір даних і впливає на здатність моделі вивчити потрібні ознаки без надмірного запам’ятовування окремих прикладів.

У сучасних системах розпізнавання важливе значення має не лише точність, але й швидкодія. Для систем підтримки водія результат обробки відеокадру повинен бути отриманий достатньо швидко, оскільки дорожня ситуація постійно змінюється. Якщо модель працює із суттєвою затримкою, виявлений дорожній знак може вже втратити актуальність для водія. Саме

тому для задач, пов’язаних із відеопотоком, часто використовуються моделі, орієнтовані на обробку в режимі, наближеному до реального часу.

Одним із відомих підходів до швидкої детекції об’єктів є сімейство моделей YOLO. У роботі J. Redmon та співавторів YOLO описано як підхід, у якому одна нейронна мережа за один прохід зображення передбачає координати об’єктів і ймовірності їхніх класів [3]. Така особливість робить подібні моделі придатними для задач, де важлива швидка обробка кадрів. У даній роботі використовується YOLOv8, оскільки бібліотека Ultralytics надає готові засоби для навчання, валідації та застосування моделі до зображень і відеопотоку [6]–[8].

Якість роботи нейронної мережі значною мірою залежить від навчального набору даних. Для коректного навчання необхідно мати достатню кількість розмічених зображень, які відображають різні умови зйомки: різне освітлення, відстань до об’єктів, ракурси, часткове перекриття та різну якість відео. Якщо певний клас представлений у датасеті недостатньо, модель може гірше розпізнавати його під час тестування. Проблема якості та різноманітності даних особливо важлива для дорожніх знаків, оскільки вони часто мають малий розмір у кадрі та можуть бути частково перекриті або зняті під різним кутом [12], [14].

Крім того, важливо враховувати не лише кількість прикладів дорожніх знаків, а й різноманітність об’єктів, які не є знаками, але можуть бути схожими на них. У практичному тестуванні може виникати ситуація, коли модель помилково позначає як дорожній знак вікно, табличку, елемент автомобіля, знак навчального транспортного засобу або інший об’єкт схожої форми чи кольору. Такі хибні спрацьовування виникають через те, що модель орієнтується на візуальні ознаки, отримані під час навчання. Якщо в навчальному наборі недостатньо прикладів складних сцен або схожих об’єктів, модель може неправильно узагальнювати ознаки.

Для зменшення таких помилок необхідно розширювати датасет, додавати більше різноманітних дорожніх сцен, збалансовувати кількість прикладів між класами та включати до навчання складні випадки, у яких присутні об’єкти, схожі на дорожні знаки. Це дозволяє моделі краще відрізняти реальні знаки від випадкових схожих елементів зображення.

Таким чином, нейронні мережі є ефективним інструментом для детекції дорожніх знаків у відеопотоці. Вони дозволяють автоматично визначати положення об’єктів у кадрі, класифікувати їх та передавати результати до системи візуалізації або формування підказок для водія. У межах розробленої системи нейронна мережа виконує роль основного модуля аналізу відеокадру, а результати її роботи використовуються для відображення рамок, класів, рівня впевненості та текстових рекомендацій для користувача. Саме тому використання нейронних мереж є доцільним для реалізації інтелектуальної системи підтримки водія.

1.7 Порівняння сучасних моделей

У задачах комп’ютерного зору, пов’язаних із розпізнаванням об’єктів, використовується багато моделей, які відрізняються архітектурою, точністю,швидкістю обробки та вимогами до обчислювальних ресурсів. Вибір конкретної моделі залежить від призначення системи. Для інтелектуальної системи підтримки водія особливо важливими є швидкодія, можливість обробки відеопотоку та достатня точність виявлення об’єктів у змінних дорожніх умовах.

Сучасні моделі детекції об’єктів умовно поділяють на двоетапні та одноетапні. Двоетапні моделі спочатку визначають області зображення, у яких потенційно можуть знаходитися об’єкти, а потім виконують класифікацію цих областей. Прикладом такого підходу є Faster R-CNN, у якому для формування областей-кандидатів використовується Region Proposal Network [4]. Перевагою двоетапних моделей є висока точність, однак їхнім недоліком є

більша обчислювальна складність, що може обмежувати використання у задачах обробки відео в режимі реального часу.

До одноетапних моделей належать SSD та моделі сімейства YOLO. На відміну від двоетапних підходів, такі моделі виконують локалізацію об’єктів і визначення їхніх класів за один прохід нейронної мережі. У роботі, присвяченій SSD, модель описується як підхід, що усуває необхідність окремого етапу генерації пропозицій областей і виконує детекцію безпосередньо за один прохід мережі [5]. Завдяки цьому ці моделі краще підходять для задач, де важливою є швидкість обробки. Сімейство YOLO належить до одноетапних моделей розпізнавання. У роботі J. Redmon та співавторів YOLO подається як єдина нейронна мережа, що передбачає координати об’єктів і ймовірності класів безпосередньо за повним зображенням [3]. Такий підхід дозволяє досягти високої швидкості роботи, що є важливим для аналізу відеопотоку в системах підтримки водія.

Для задачі розпізнавання дорожніх знаків важливо враховувати не лише загальну точність моделі, а й її здатність працювати з малими об’єктами, різними відстанями до камери, зміною освітлення та частковим перекриттям знаків. У систематичному огляді методів розпізнавання дорожніх знаків на основі YOLO зазначається, що такі задачі ускладнюються саме через варіативність дорожніх умов, розмір об’єктів та якість зображення [12]. Тому модель для такої системи повинна забезпечувати баланс між швидкодією та стійкістю до змін середовища.

Серед сучасних моделей сімейства YOLO у даній роботі було обрано YOLOv8. Згідно з документацією Ultralytics, YOLOv8 підтримує задачі детекції об’єктів і може використовуватись для навчання на наборах даних користувача [6]. Для дипломної роботи було обрано варіант yolov8n.pt, оскільки він має невеликий розмір і краще підходить для експериментального прототипу, де важливі швидкість роботи та можливість запуску на доступному апаратному забезпеченні.

Використання YOLOv8 також є зручним з погляду програмної реалізації. Бібліотека Ultralytics надає готові засоби для навчання, валідації та застосування моделі до зображень або відео. Це дозволяє зосередитися не лише на самій моделі, а й на побудові повного програмного комплексу: підготовці датасету, навчанні, тестуванні, візуалізації результатів і формуванні підказок для водія [6], [7], [8].

Таким чином, аналіз сучасних моделей детекції об’єктів показує, що для задачі розпізнавання дорожніх знаків у відеопотоці найбільш доцільним є використання одноетапних моделей, орієнтованих на швидку обробку кадрів.

У межах даної роботи модель YOLOv8 була обрана як компроміс між швидкодією, доступністю інструментів навчання та можливістю інтеграції у програмний прототип інтелектуальної системи підтримки водія

2 ПРОЄКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ

2.1 Постановка задачі

Метою розробки інтелектуальної системи підтримки водія є створення програмного комплексу, здатного виконувати аналіз дорожньої ситуації у режимі реального часу. Система повинна автоматично визначати дорожні знаки на відеопотоці, класифікувати їх та передавати водію додаткові текстові підказки.

Основними завданнями розробки стали підготовка датасету для навчання моделі, розробка механізму конвертації анотацій, навчання нейронної мережі YOLOv8 та створення програмного інтерфейсу для перегляду результатів роботи системи.

Під час створення системи особлива увага приділялась швидкодії обробки відео, оскільки затримки у визначенні дорожніх знаків можуть негативно впливати на ефективність підтримки водія.

2.2 Архітектура системи

Архітектура розробленої системи складається з декількох взаємопов’язаних модулів. Першим модулем є модуль отримання відеопотоку, який відповідає за зчитування кадрів із відеофайлу.

Далі кадр передається до моделі YOLOv8, де виконується необхідна підготовка зображення та детекція дорожніх знаків із визначенням їхнього класу.

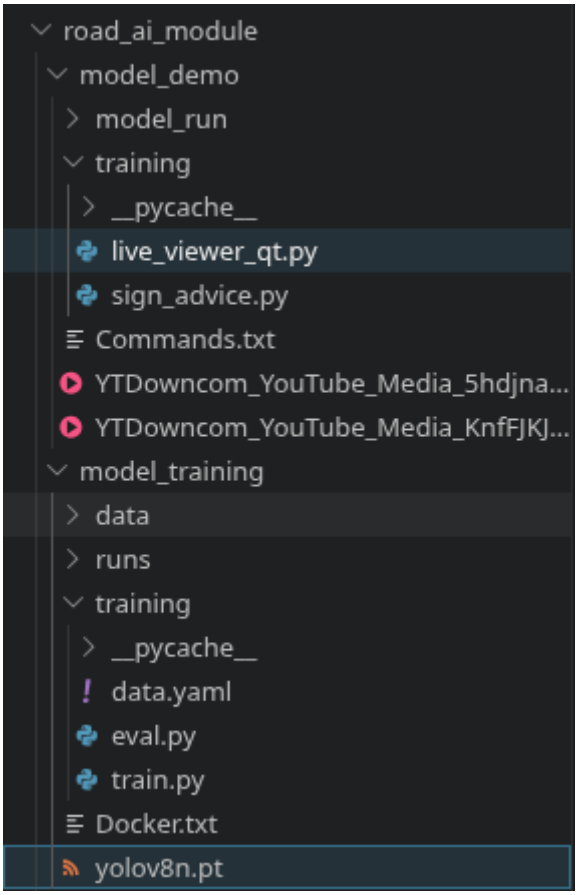

Рисунок 2.1 — Архітектура роботи

Результати роботи моделі передаються до модуля візуалізації. У графічному інтерфейсі відображаються межі детекції, назви класів та рівень впевненості моделі.

Додатково система формує текстові рекомендації для водія залежно від типу виявленого дорожнього знака.

2.3 Підготовка та розмітка датасету

Одним із ключових етапів створення системи стала підготовка навчального датасету. Для формування набору даних використовувались відеозаписи дорожнього руху, отримані з відкритих джерел.

Для розмітки кадрів було використано платформу CVAT. Під час розмітки для кожного дорожнього знака створювались анотації із зазначенням його положення на кадрі та належності до певного класу.

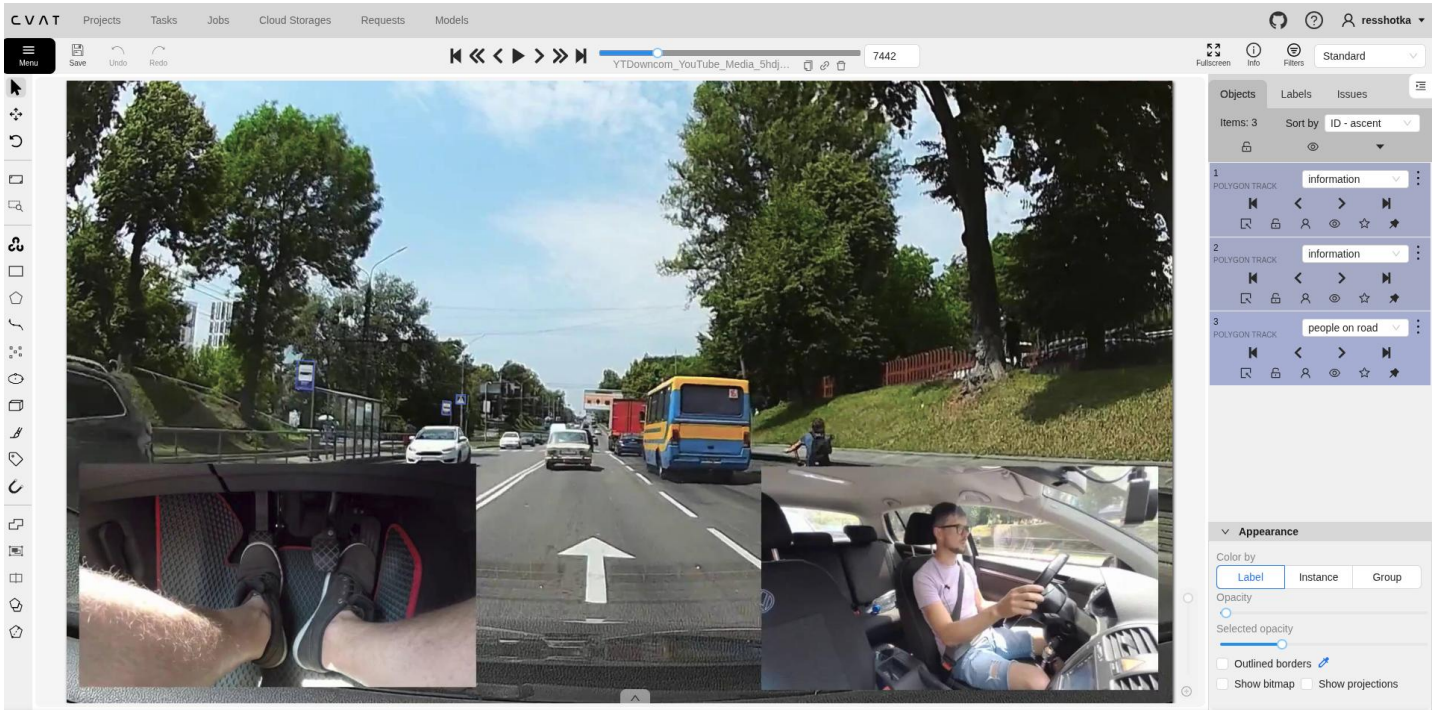

Рисунок 2.2 — Інтерфейс застосунку CVAT

У процесі роботи було використано класи regulatory, warning, priority та information. Такий поділ дозволив системі навчитися розрізняти основні типи дорожніх знаків.

2.4 Розгортання CVAT у Docker

Для організації процесу розмітки датасету система CVAT була розгорнута у середовищі Docker. Використання контейнеризації дозволило забезпечити автономну роботу системи розмітки в локальному середовищі без постійної залежності від доступу до зовнішніх сервісів або мережі Інтернет.

Запуск системи виконувався за допомогою docker compose. Після запуску користувач отримував доступ до веб-інтерфейсу CVAT через браузер, що дозволяло виконувати розмітку кадрів у зручному режимі.

Використання Docker також забезпечило стабільність роботи сервісу та можливість швидкого відновлення середовища у випадку помилок або пошкодження конфігурації.

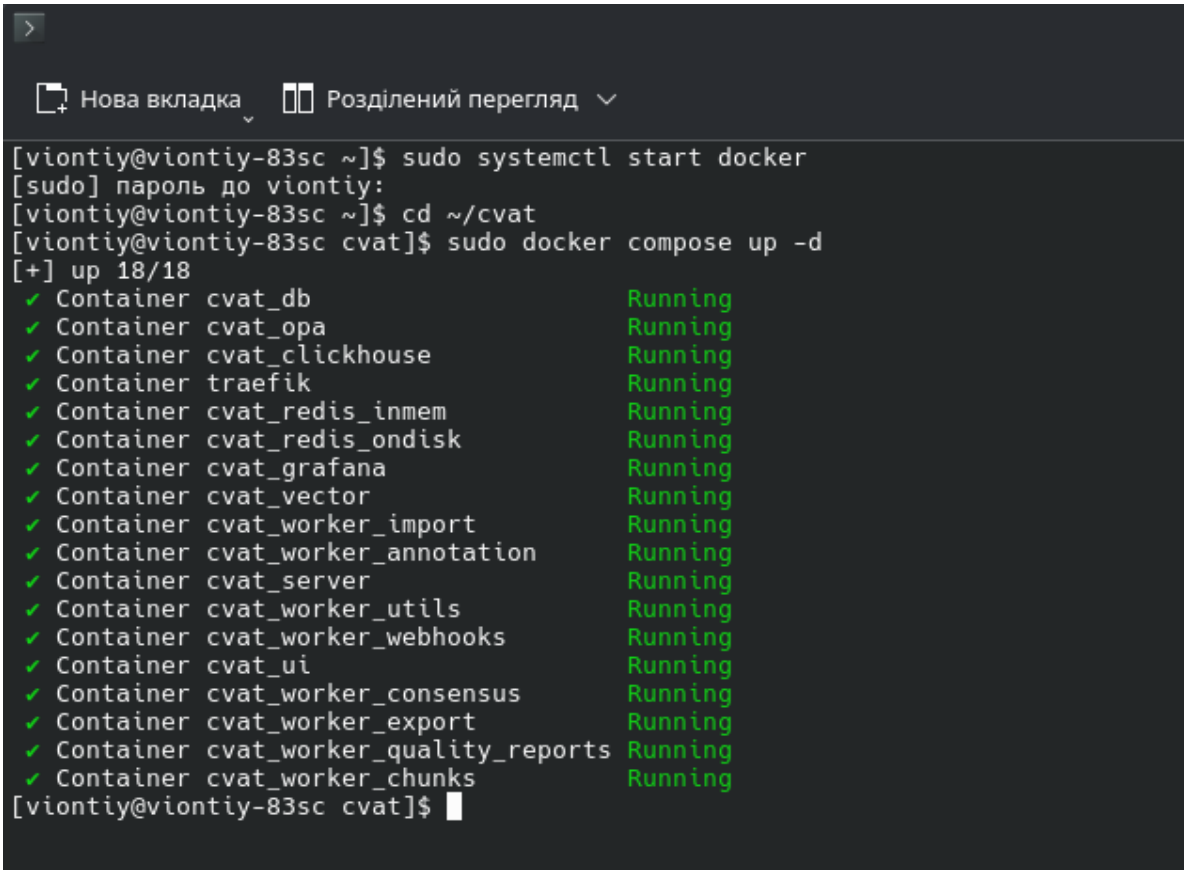

Рисунок 2.3 — Запуск Docker

2.5 Навчання моделі YOLOv8

Для навчання моделі розпізнавання дорожніх знаків було використано попередньо підготовлену модель yolov8n.pt із бібліотеки Ultralytics YOLOv8. Вибір саме цієї моделі пояснюється її невеликим розміром, високою швидкістю роботи та можливістю використання у задачах, наближених до режиму реального часу. Для розроблюваного прототипу це є важливим фактором, оскільки система повинна обробляти відеопотік без значних затримок.

Навчання виконувалось на підготовленому наборі даних, який складався із зображень дорожніх сцен та відповідних файлів анотацій. Для налаштування процесу навчання було використано конфігураційний файл data.yaml.

У цьому файлі задавались шлях до набору даних, розташування навчальної та валідаційної вибірок, а також перелік класів дорожніх знаків, які модель повинна розпізнавати.

```
path:
/mnt/storage/Program/road_ai_module/model_training/data/Detect

train: images/train
val: images/val

names:
  0: crosswalk
  1: children_on_road
  2: speed_limit
  3: speed bump
  4: turn_sign
  5: bus_stop
  6: patrol
  7: parking
  8: parking ban
```

Рисунок 2.4 — Код файлу конфігурації

У конфігураційному файлі навчальна вибірка була розміщена у директорії images/train, а валідаційна — у директорії images/val. Також у файлі було зазначено список класів, серед яких crosswalk, children_on_road, speed_limit, speed_bump, turn_sign, bus_stop, patrol, parking та parking_ban. Приклад структури конфігураційного файлу наведено на рисунку 2.4.

Під час навчання модель отримувала на вхід зображення та відповідні їм анотації у форматі YOLO. Для кожного об’єкта в анотації задавались клас дорожнього знака та координати обмежувальної рамки. На основі цих даних YOLOv8 порівнювала власні передбачення з правильною розміткою та поступово змінювала внутрішні ваги нейронної мережі.

Процес навчання полягав у тому, що модель багаторазово проходила через навчальний набір даних. Один повний прохід через усі навчальні зображення називається епохою. У даній роботі навчання виконувалось протягом 50 епох із розміром вхідного зображення 640 на 480 пікселів. Такий розмір було обрано як компроміс між якістю детекції та швидкістю обробки, оскільки більші зображення можуть підвищити точність для дрібних об’єктів, але потребують більше обчислювальних ресурсів.

У процесі навчання модель оптимізувала свої параметри для одночасного вирішення двох задач: визначення положення дорожнього знака на зображенні та класифікації цього знака за одним із заданих класів. Якщо передбачена рамка або клас не збігалися з розміткою, значення помилки збільшувалось, після чого ваги моделі коригувалися під час наступних ітерацій навчання. Завдяки цьому модель поступово навчалась виділяти характерні ознаки дорожніх знаків, зокрема форму, колір, контури та розташування об’єкта в кадрі.

Після завершення навчання було отримано два файли ваг моделі: best.pt та last.pt. Файл last.pt містить стан моделі після останньої епохи навчання, а best.pt зберігає варіант моделі з найкращими результатами на валідаційній вибірці. Надалі саме ці ваги можуть використовуватись для виконання детекції дорожніх знаків на відеозаписах дорожнього руху та в графічному інтерфейсі розробленої системи.

2.6 Розробка інтерфейсу перегляду

Для взаємодії користувача з розробленою системою було створено графічний інтерфейс на основі бібліотеки PySide6. Використання PySide6 дозволило реалізувати окреме вікно програми з елементами керування відео, вибором навченої моделі та відображенням результатів детекції безпосередньо під час перегляду відеопотоку.

Інтерфейс системи виконує роль проміжного рівня між користувачем і моделлю YOLOv8. Користувач може обрати файл навченої моделі, наприклад best.pt, а також відеофайл, на якому необхідно виконати розпізнавання дорожніх знаків. Після запуску відтворення відео система починає покадрову обробку: кожен кадр передається до моделі, модель виконує детекцію об’єктів, після чого результати повертаються до інтерфейсу для візуального відображення.

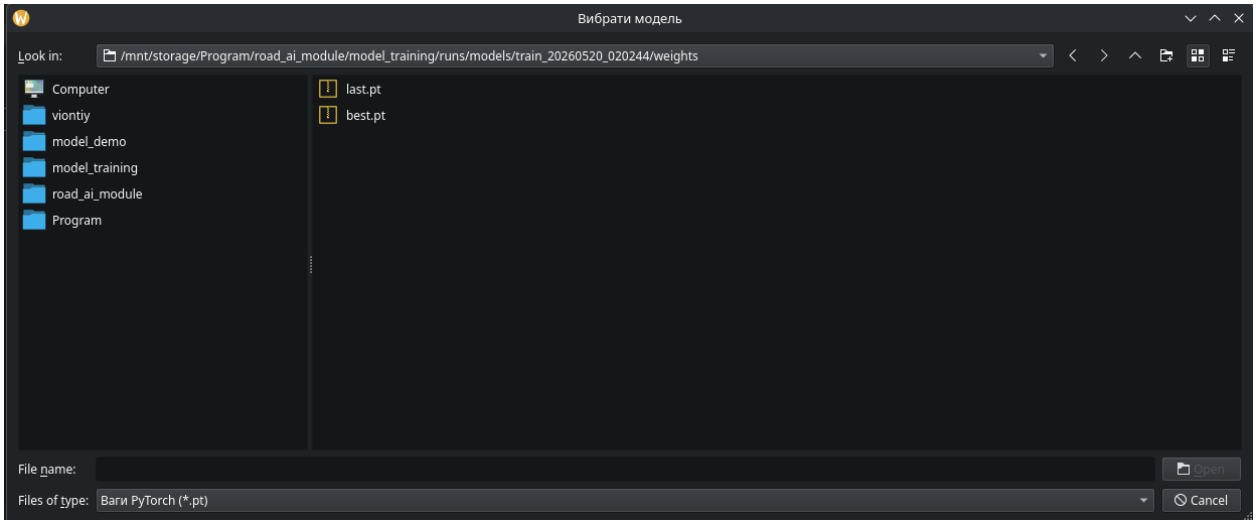

Рисунок 2.5 — Інтерфейс програвача відео з детекцією

Під час роботи програми на відео накладаються обмежувальні рамки навколо знайдених об’єктів. Для кожного виявленого дорожнього знака виводиться назва класу та значення впевненості моделі. Значення впевненості показує, наскільки модель оцінює власне передбачення як надійне. Наприклад, якщо біля рамки виводиться значення turn_sign 0.88, це означає, що модель віднесла знайдений об’єкт до класу turn_sign із рівнем впевненості 0.88.

Окрім основної області перегляду відео, інтерфейс містить елементи керування відтворенням. Користувач може запускати та зупиняти відео, перемотувати його, а також переглядати результати детекції на різних ділянках запису. Це зручно під час тестування моделі, оскільки дозволяє повторно переглядати фрагменти, де модель правильно або помилково визначила дорожній знак.

Додатково у правій частині інтерфейсу виводиться список знайдених об’єктів для поточного кадру. У цьому списку відображаються класи об’єктів та відповідні значення confidence. Такий підхід дозволяє не лише бачити рамки на відео, а й окремо аналізувати результати роботи моделі у текстовому вигляді. Приклад інтерфейсу програвача відео з результатами детекції наведено на рисунку 2.6.

Рисунок 2.6 — Інтерфейс програвача відео з детекцією

Розроблений інтерфейс є важливою частиною системи, оскільки він перетворює результати роботи нейронної мережі у зрозумілу для користувача форму. Без графічного відображення модель лише повертає координати, класи та числові значення впевненості, тоді як інтерфейс дозволяє наочно оцінити, які саме об’єкти були виявлені та наскільки коректно система працює на відеозаписі.

2.7 Система підказок для водія

Окремою частиною розробленої системи стала підсистема текстових підказок для водія. Її призначення полягає в тому, щоб не лише показати факт виявлення дорожнього знака, а й надати користувачу коротке пояснення або рекомендацію, пов’язану з цим типом знака.

Після того як модель YOLOv8 виявляє об’єкт у кадрі, система визначає його клас. Далі цей клас передається до підсистеми підказок, де для нього підбирається заздалегідь підготовлений текст рекомендації. Наприклад, для класу turn_sign система може виводити пораду заздалегідь зайняти потрібну смугу та уникати різкої зміни траєкторії руху. Для класу bus_stop

відображається підказка про необхідність зменшити швидкість і звернути увагу на пішоходів біля зупинки. Для класу crosswalk система нагадує про можливу появу пішоходів на дорозі.

Текстові підказки відображаються в окремому вікні, що дозволяє не перевантажувати основний відеоінтерфейс зайвою інформацією. У цьому вікні користувач бачить назву виявленого типу знака, клас моделі, час до зникнення підказки та короткі рекомендації щодо можливої поведінки водія. Приклад роботи вікна підказок наведено на рисунку 2.7.

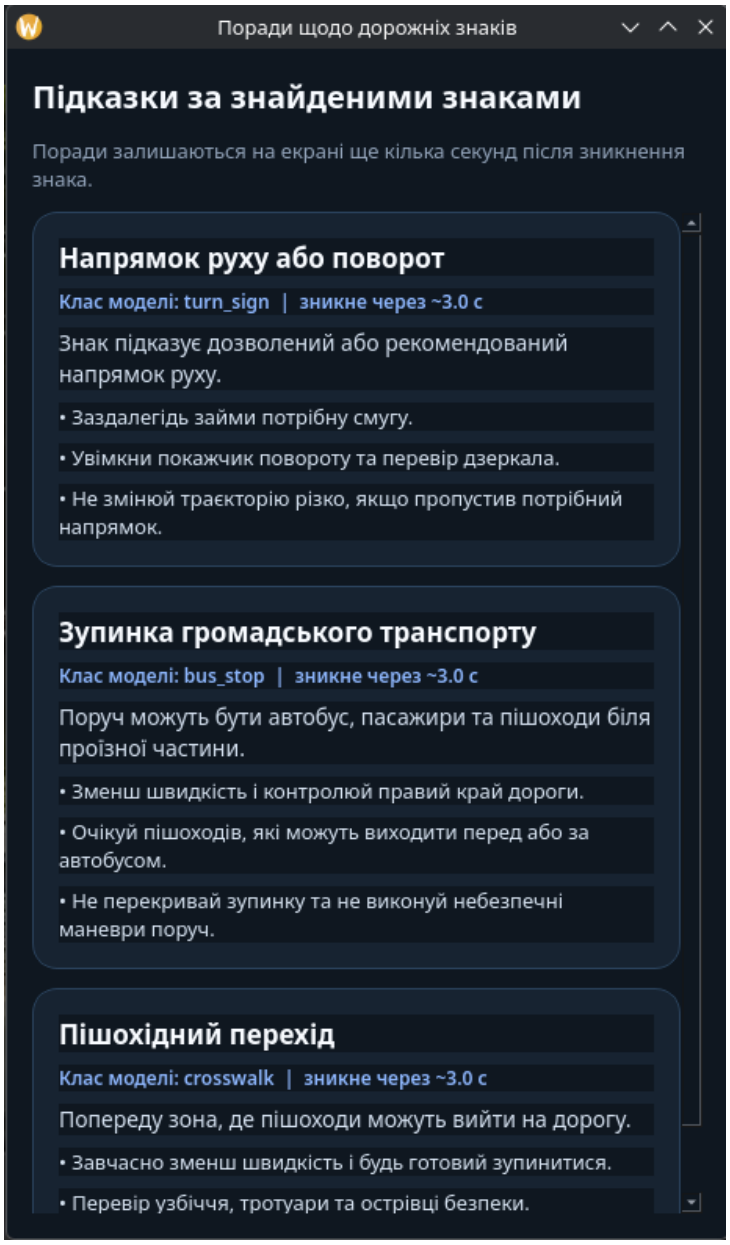

Рисунок 2.7 — Вікно текстових порад

Особливістю підсистеми є те, що підказки не зникають миттєво після того, як знак перестає бути видимим у кадрі. Вони залишаються активними протягом кількох секунд після зникнення знака. Це зроблено для того, щоб користувач мав час прочитати рекомендацію навіть у випадку, коли знак швидко з’явився та зник із поля зору камери.

Система підказок має допоміжний характер і не виконує автоматичного керування транспортним засобом. Її задача полягає у наданні додаткової інформації, яка може допомогти водієві швидше зорієнтуватися в дорожній ситуації. Такий підхід відповідає концепції інформаційно-попереджувальної системи підтримки водія, де остаточне рішення залишається за людиною.

Водночас варто враховувати, що якість підказок залежить від правильності роботи моделі детекції. Якщо модель помилково визначає об’єкт, схожий на дорожній знак, система також може сформувати підказку для цього хибного спрацьовування. Тому підсистема рекомендацій повинна розглядатися як частина прототипу, а не як повністю готове рішення для використання в реальному автомобілі без додаткової перевірки та вдосконалення.

Таким чином, розроблений інтерфейс перегляду та система підказок забезпечують зручну взаємодію користувача з моделлю детекції. Вони дозволяють переглядати відео, бачити знайдені дорожні знаки, оцінювати рівень впевненості моделі та отримувати короткі рекомендації, пов’язані з виявленими класами об’єктів.

3 ДОСЛІДЖЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Методика тестування

Для оцінки ефективності розробленої системи підтримки водія було проведено тестування моделі детекції об’єктів на відеоданих. Основною метою тестування було визначення здатності моделі коректно виявляти дорожні знаки в різних умовах та оцінка її точності і стабільності роботи.

Тестування проводилось на двох типах відеоданих:

- відео, що використовувалось під час формування навчального набору (контрольне середовище);
- сторонні відео, які не входили до навчальної вибірки (перевірка узагальнюючої здатності моделі).

Такий підхід дозволяє оцінити не лише якість запам’ятовування навчальних даних, але й здатність моделі працювати в нових умовах.

Для проведення тестування використовувався відеопотік, який оброблявся покадрово. Кожен кадр передавався до моделі, яка виконувала детекцію

об’єктів та повертала результати у вигляді координат обмежувальних областей і значень впевненості (confidence score). Отримані результати відображались у вигляді накладених на відео рамок та підписів класів.

Оцінка якості роботи моделі здійснювалась за такими критеріями:

- точність детекції — правильність визначення класу об’єкта;
- повнота виявлення — здатність знаходити всі релевантні об’єкти в кадрі;
- стійкість до умов середовища — робота при зміні освітлення, відстані та ракурсу;
- швидкодія — можливість обробки відео в режимі, наближеному до реального часу.

Окремо враховувався рівень впевненості моделі для кожного виявленогооб’єкта. Це дозволяло оцінити, наскільки стабільно модель приймає рішення та чи не виникає великої кількості хибних спрацьовувань.

Тестування проводилось у середовищі, що забезпечує візуалізацію результатів детекції. Це дозволило виконувати якісний аналіз роботи моделі, спостерігаючи за її поведінкою у реальному часі, а також оцінювати коректність виявлення об’єктів безпосередньо під час відтворення відео.

Таким чином, обрана методика тестування дозволяє комплексно оцінити ефективність роботи моделі детекції об’єктів як на знайомих, так і на нових даних, що є важливим для визначення її придатності до використання в системах підтримки водія.

3.2 Тестування на сторонніх відео

Після перевірки моделі на відеоданих, що використовувались під час навчання, було проведено додаткове тестування на сторонніх відеоматеріалах. Основною метою цього етапу є оцінка здатності моделі узагальнювати отримані знання та працювати в умовах, які відрізняються від навчального набору даних.

Для тестування використовувались відео з різними характеристиками:

- різна якість зображення (висока та середня);
- різні погодні умови (ясна погода, знижена освітленість);
- зміна відстані до об’єктів;

Під час тестування модель обробляла відеопотік у покадровому режимі, виконуючи детекцію дорожніх знаків.

Отримані результати аналізувались за наступними параметрами:

- коректність визначення класу об’єкта;
- стабільність детекції при русі камери;
- рівень впевненості моделі;
- наявність хибних спрацьовувань.

Було встановлено, що на сторонніх відео точність роботи моделі знижується у порівнянні з навчальними даними. У середньому значення впевненості для виявлених об’єктів знаходилось у межах 0.60–0.70. При цьому модель продовжувала коректно визначати основні класи дорожніх знаків, що свідчить про її здатність до узагальнення.

Разом з тим, у складних умовах спостерігались наступні особливості:

- зниження точності при значному віддаленні об’єкта;
- труднощі при частковому перекритті знаків;
- нестабільність детекції при різких змінах освітлення.

Таким чином, результати тестування підтверджують, що модель здатна працювати не лише в умовах навчального набору, але й на нових відеоданих, хоча і з деяким зниженням точності.

3.3 Аналіз результатів

На основі проведеного тестування було виконано оцінку роботи розробленої системи підтримки водія. Аналіз проводився з урахуванням як числових показників моделі, зокрема рівня впевненості детекції, так і візуальної оцінки результатів під час перегляду відео.

На навчальних відеоматеріалах модель демонструвала достатньо стабільну роботу. У більшості випадків дорожні знаки визначались коректно, а середнє значення впевненості для виявлених об’єктів становило приблизно 0.80. Це свідчить про те, що модель засвоїла основні візуальні ознаки класів, використаних під час навчання.

Під час тестування на сторонніх відео якість детекції знижувалась. Значення впевненості для частини виявлених об’єктів знаходилось у межах 0.60–0.70. Це пояснюється тим, що умови на таких відео відрізнялись від навчального набору: змінювались освітлення, ракурс камери, відстань до знаків, якість зображення та загальна складність дорожньої сцени.

Окремою проблемою стали хибні спрацьовування на об’єктах, які візуально схожі на дорожні знаки. У деяких випадках модель могла помилково позначати як знак вікна, таблички, елементи кузова автомобіля, знак навчального транспортного засобу або інші об’єкти подібної форми чи кольору. Це пов’язано з тим, що модель орієнтується на візуальні ознаки, отримані під час навчання, і за недостатньої кількості різноманітних прикладів може плутати дорожні знаки з об’єктами, які мають схожу геометрію.

Попри наявні обмеження, система здатна працювати в режимі, наближеному до реального часу.

Результати детекції відображаються безпосередньо поверх відео, що дозволяє швидко оцінювати поведінку моделі та виявляти помилки її роботи. Графічний інтерфейс також дає можливість переглядати знайдені об’єкти, рівень впевненості моделі та аналізувати зміну результатів у різних умовах.

Важливим елементом розробленої системи є підсистема текстових підказок для водія. Вона дозволяє не лише показувати факт виявлення дорожнього знака, а й перетворювати результат детекції у зрозумілу рекомендацію. Це підвищує прикладну цінність системи, оскільки користувач отримує не тільки візуальну рамку на відео, а й коротке пояснення можливих дій.

Отримані результати показують, що обрана модель YOLOv8 є придатною для створення прототипу системи розпізнавання дорожніх знаків. Водночас для підвищення точності та зменшення кількості хибних спрацьовувань необхідно розширити навчальний датасет, додати більше прикладів складних дорожніх сцен, а також включити до набору даних об’єкти, схожі на дорожні знаки, але такими не є.

Таким чином, розроблена система демонструє технічну працездатність і може бути використана як основа для подальшого вдосконалення інтелектуальної системи підтримки водія.

3.4 Проблеми та обмеження системи

Незважаючи на отримані позитивні результати, розроблена система має ряд обмежень, які необхідно враховувати.

Основним обмеженням розробленої системи є залежність від якості та різноманітності навчального набору. Якщо певний тип знаків або умови зйомки представлені недостатньо, модель може нестабільно працювати на новому відео. Це проявляється у пропусках дрібних знаків, зниженні confidence при великій відстані до об’єкта та хибних спрацьовуваннях у складних сценах.

До основних обмежень системи можна віднести:

- зниження точності при зміні умов освітлення;
- залежність від якості відео;
- складність детекції дрібних об’єктів на великій відстані;
- можливість хибних спрацьовувань;
- обмежений набір класів.

Також варто зазначити, що система не враховує контекст дорожньої ситуації. Вона лише визначає наявність об’єктів, але не аналізує їх взаємодію між собою.

Окремим обмеженням є використання спрощеної моделі (YOLOv8n), яка має обмежену точність у порівнянні з більш складними варіантами.

Тому, для підвищення ефективності системи необхідне подальше вдосконалення як моделі, так і навчального набору даних.

3.5 Можливості подальшого розвитку

Подальший розвиток системи може бути спрямований на підвищення точності, розширення функціональності та покращення взаємодії з користувачем.

До основних напрямків розвитку можна віднести:

- розширення датасету за рахунок нових відеоданих;збільшення кількості класів об’єктів;
- використання більш складних моделей (YOLOv8m, YOLOv8l);
- оптимізацію швидкодії системи;
- інтеграцію з іншими сенсорами (GPS, датчики руху);
- впровадження аналізу дорожньої ситуації, а не лише окремих об’єктів.

Перспективним напрямком є також використання системи у режимі реального часу безпосередньо в автомобілі. Це дозволить створити повноцінну систему підтримки водія, здатну підвищити рівень безпеки дорожнього руху.

Розроблена система може бути використана як основа для створення більш складних інтелектуальних транспортних рішень.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено прототип інтелектуальної системи підтримки водія для розпізнавання дорожніх знаків у відеопотоці. Система поєднує підготовлений датасет, навчену модель YOLOv8, модуль обробки відео, графічний інтерфейс на PySide6 та підсистему текстових підказок для водія.

У роботі було проаналізовано предметну область, розглянуто сучасні системи підтримки водія, методи комп'ютерного зору та підходи до детекції об'єктів. Для реалізації системи було обрано модель YOLOv8, оскільки вона забезпечує достатню швидкість обробки відео та може використовуватись у задачах, наближених до режиму реального часу.

Для підготовки навчального набору даних було використано платформу CVAT, розгорнуту у Docker-середовищі. Дорожні знаки розмічались за чотирма узагальненими класами: regulatory, warning, priority та information. Після розмітки анотації було експортовано у форматі COCO та конвертовано у формат Ultralytics YOLO, придатний для навчання моделі.

Навчання моделі виконувалось із використанням бібліотеки Ultralytics YOLOv8. У результаті було отримано навчені ваги моделі, які використовувались для подальшої детекції дорожніх знаків на відеозаписах. Для взаємодії з користувачем було розроблено графічний інтерфейс, який дозволяє завантажувати модель і відеофайл, керувати відтворенням, переглядати результати детекції та отримувати текстові підказки залежно від виявленого класу знака.

Практичне тестування показало, що система здатна виявляти основні категорії дорожніх знаків у відеопотоці. Водночас було встановлено, що якість роботи моделі залежить від збалансованості датасету, умов освітлення, відстані до об'єкта, якості відеозапису та різноманітності навчальних прикладів.

Найбільш помітною проблемою стали хибні спрацьовування на об’єктах, які візуально схожі на дорожні знаки. Наприклад, модель може помилково позначати як знак вікна, таблички, елементи автомобілів, знак навчального транспортного засобу або інші об’єкти подібної форми та кольору. Це пов’язано з тим, що модель навчається знаходити характерні візуальні ознаки класів, і за недостатньої кількості різноманітних прикладів може сприймати схожі об’єкти як дорожні знаки.

Для зменшення кількості хибних спрацьовувань у подальшому доцільно розширити датасет, додати більше прикладів складних дорожніх сцен, збалансувати кількість зображень між класами та включити до навчання об’єкти, схожі на дорожні знаки, але такими не є. Також перспективним напрямом є збільшення кількості класів, використання складніших моделей YOLOv8 та подальше вдосконалення логіки формування підказок для водія.

Таким чином, розроблена система може бути використана як прототип інтелектуальної системи підтримки водія та як основа для подальшого розвитку програмних засобів аналізу дорожньої ситуації.

Перелік посилань

1. World Health Organization. Global status report on road safety 2023. Geneva : World Health Organization, 2023. URL: https://www.who.int/publications/i/item/9789240086517
2. European Commission. Advanced driver assistance systems. European Commission, Mobility and Transport. URL: https://road-safety.transport.ec.europa.eu/eu-road-safety-policy/priorities/safe-vehicles/advanced-driver-assistance-systems_en
3. Redmon J., Divvala S., Girshick R., Farhadi A. You Only Look Once: Unified, Real-Time Object Detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016. P. 779–788. URL: https://arxiv.org/abs/1506.02640
4. Ren S., He K., Girshick R., Sun J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. Advances in Neural Information Processing Systems. 2015. URL: https://arxiv.org/abs/1506.01497
5. Liu W., Anguelov D., Erhan D., Szegedy C., Reed S., Fu C.-Y., Berg A. C. SSD: Single Shot MultiBox Detector. European Conference on Computer Vision. 2016. URL: https://arxiv.org/abs/1512.02325
6. Ultralytics. YOLOv8 Documentation. URL: https://docs.ultralytics.com/models/yolov8/
7. Ultralytics. Train Custom Data. URL: https://docs.ultralytics.com/modes/train/
8. Ultralytics. Model Validation with Ultralytics YOLO. URL: https://docs.ultralytics.com/modes/val/
9. Ultralytics. Performance Metrics Deep Dive. URL: https://docs.ultralytics.com/guides/yolo-performance-metrics/
10. Ultralytics. Ultralytics YOLO Repository. GitHub. URL: https://github.com/ultralytics/ultralytics
11. Yaseen M. What is YOLOv8: An In-Depth Exploration of the Internal Features of the Next-Generation Object Detector. arXiv preprint. 2024. URL: https://arxiv.org/abs/2408.15857
12. Flores-Calero M. та ін. Traffic Sign Detection and Recognition Using YOLO Object Detection Algorithm: A Systematic Review. Mathematics. 2024. Vol. 12, No. 2. Article 297. URL: https://www.mdpi.com/2227-7390/12/2/297

13. Stallkamp J., Schlipsing M., Salmen J., Igel C. The German Traffic Sign Recognition Benchmark: A Multi-Class Classification Competition. International Joint Conference on Neural Networks. 2011. DOI: 10.1109/IJCNN.2011.6033395.

14. Chu J., Zhang C., Yan M., Zhang H., Ge T. TRD-YOLO: A Real-Time, High-Performance Small Traffic Sign Detection Algorithm. Sensors. 2023. Vol. 23, No. 8. Article 3871. URL: https://www.mdpi.com/1424-8220/23/8/3871

15. CVAT. CVAT Documentation. URL: https://docs.cvat.ai/docs/

16. CVAT. CVAT Overview. URL: https://docs.cvat.ai/docs/getting_started/overview/

17. Docker. Docker Compose Documentation. URL: https://docs.docker.com/compose/

18. OpenCV. VideoCapture Class Reference. URL: https://docs.opencv.org/4.x/d8/dfe/classcv_1_1VideoCapture.html

19. OpenCV. Drawing Functions in OpenCV. URL: https://docs.opencv.org/4.x/dc/da5/tutorial_py_drawing_functions.html

20. Qt. Qt for Python / PySide6. URL: https://www.qt.io/qt-for-python

Додаток А – Код основного интерфейсу

```
from __future__ import annotations

from concurrent.futures import Future, ThreadPoolExecutor
import sys
import time
from pathlib import Path

import cv2
from ultralytics import YOLO

from PySide6.QtCore import Qt, QTimer
from PySide6.QtGui import QImage, QPixmap
from PySide6.QtWidgets import (
    QApplication,
    QFileDialog,
    QHBoxLayout,
    QLabel,
    QMainWindow,
    QMessageBox,
    QPushButton,
    QSlider,
    QStyle,
    QVBoxLayout,
    QWidget,
)

from sign_advice import AdviceWindow

ROOT = Path(__file__).resolve().parent.parent
PROJECT_ROOT = ROOT.parent
TRAINING_ROOT = PROJECT_ROOT / "model_training"

MODEL_PICK_CANDIDATES = (
    ROOT / "model_run" / "weights" / "best.pt",
    ROOT / "model_run" / "best.pt",
    TRAINING_ROOT / "runs" / "detect" / "model_runs" / "train"
/ "weights" / "best.pt",
    TRAINING_ROOT / "runs" / "detect" / "reports" / "runs" /
"weights" / "best.pt",
    TRAINING_ROOT / "yolov8n.pt",
)

def existing_pick_dir(candidates: tuple[Path, ...], fallback:
Path) -> str:
    for path in candidates:
        if path.exists():
```

```
            return str(path.parent if path.is_file() else path)
        if path.parent.exists():
            return str(path.parent)
    return str(fallback)

def run_yolo_inference(
    model: YOLO,
    frame,
    frame_idx: int,
    generation: int,
    conf: float,
    imgsz: int,
    device,
):
    res = model.predict(
        source=frame,
        conf=conf,
        imgsz=imgsz,
        device=device,
        verbose=False,
    )[0]

    annotated = res.plot()
    detections_lines = []
    detected_names: list[str] = []

    if res.boxes is not None and len(res.boxes) > 0:
        boxes = res.boxes
        confs = boxes.conf.tolist()
        clss = boxes.cls.tolist()
        names = model.names

        items = sorted(zip(confs, clss), reverse=True)
        for c, k in items[:30]:
            name = names.get(int(k), str(int(k))) if
isinstance(names, dict) else str(int(k))
            detected_names.append(name)
            detections_lines.append(f"{name:>14}
conf={c:.2f}")

    return {
        "frame_idx": frame_idx,
        "generation": generation,
        "annotated": annotated,
        "detections_lines": detections_lines,
        "detected_names": detected_names,
    }

def draw_sidebar(frame, lines: list[str], width: int = 420):
```

```
    h, w = frame.shape[:2]
    new = cv2.copyMakeBorder(frame, 0, 0, 0, width,
cv2.BORDER_CONSTANT, value=(20, 20, 20))

    x0 = w + 12
    y = 28
    cv2.putText(new, "Detections (this frame)", (x0, y),
                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (240, 240, 240),
2, cv2.LINE_AA)
    y += 26

    if not lines:
        cv2.putText(new, "none", (x0, y),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.65, (180, 180,
180), 2, cv2.LINE_AA)
        return new

    for t in lines[:22]:
        y += 22
        cv2.putText(new, t, (x0, y),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.58, (220, 220,
220), 1, cv2.LINE_AA)

    return new

class YoloViewer(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("YOLO Qt Viewer")
        self.resize(1500, 900)

        self.model = None
        self.model_path = None
        self.video_path = None

        self.cap = None
        self.current_frame = None
        self.current_view = None

        self.conf = 0.10
        self.imgsz = 1280
        self.device = None
        self.every = 1

        self.fps = 25.0
        self.total_frames = 0
        self.frame_idx = 0
        self.last_infer = -999999
        self.paused = True
```

```
        self.slider_is_pressed = False
        self.was_playing_before_drag = False

        self.infer_executor = ThreadPoolExecutor(max_workers=1)
        self.pending_infer: Future | None = None
        self.pending_frame_idx: int | None = None
        self.inference_generation = 0

        self.t0 = time.time()

        self.timer = QTimer(self)
        self.timer.timeout.connect(self.next_frame)

        self.infer_poll_timer = QTimer(self)
        self.infer_poll_timer.setInterval(15)

self.infer_poll_timer.timeout.connect(self.poll_inference)

        self.build_ui()
        self.advice_window = AdviceWindow(hold_seconds=3.0)
        self.advice_window.show()

        # Обязательный выбор при старте
        self.initial_pick_required()

    def build_ui(self):
        root = QWidget()
        self.setCentralWidget(root)
        main_layout = QVBoxLayout(root)
        main_layout.setContentsMargins(12, 12, 12, 12)
        main_layout.setSpacing(10)

        top = QHBoxLayout()

        self.model_path_label = QLabel("Модель не выбрана")
        self.model_path_label.setStyleSheet("color: gray;")

self.model_path_label.setTextInteractionFlags(Qt.TextSelectable
ByMouse)

        self.video_path_label = QLabel("Видео не выбрано")
        self.video_path_label.setStyleSheet("color: gray;")

self.video_path_label.setTextInteractionFlags(Qt.TextSelectable
ByMouse)

        self.pick_model_btn = QPushButton("Выбрать модель")
        self.pick_video_btn = QPushButton("Выбрать видео")
        self.reload_btn = QPushButton("Загрузить заново")
```

```
        self.pick_model_btn.clicked.connect(self.pick_model)
        self.pick_video_btn.clicked.connect(self.pick_video)
        self.reload_btn.clicked.connect(self.reload_all)

        top.addWidget(QLabel("Модель:"))
        top.addWidget(self.model_path_label, 1)
        top.addWidget(self.pick_model_btn)

        top.addSpacing(20)

        top.addWidget(QLabel("Видео:"))
        top.addWidget(self.video_path_label, 1)
        top.addWidget(self.pick_video_btn)
        top.addWidget(self.reload_btn)

        self.video_label = QLabel("Нет видео")
        self.video_label.setAlignment(Qt.AlignCenter)
        self.video_label.setStyleSheet("""
            QLabel {
                background-color: #111;
                color: #bbb;
                border: 1px solid #333;
            }
        """)
        self.video_label.setMinimumHeight(650)

        self.timeline = QSlider(Qt.Horizontal)
        self.timeline.setRange(0, 0)

self.timeline.sliderPressed.connect(self.on_slider_pressed)

self.timeline.sliderReleased.connect(self.on_slider_released)
        self.timeline.sliderMoved.connect(self.on_slider_moved)

        time_row = QHBoxLayout()
        self.current_time_label = QLabel("00:00")
        self.duration_label = QLabel("00:00")
        time_row.addWidget(self.current_time_label)
        time_row.addStretch()
        time_row.addWidget(self.duration_label)

        controls = QHBoxLayout()

        self.play_btn = QPushButton()

self.play_btn.setIcon(self.style().standardIcon(QStyle.SP_Media
Play))
        self.play_btn.clicked.connect(self.play)
```

```
        self.pause_btn = QPushButton()

self.pause_btn.setIcon(self.style().standardIcon(QStyle.SP_Medi
aPause))
        self.pause_btn.clicked.connect(self.pause)

        self.stop_btn = QPushButton()

self.stop_btn.setIcon(self.style().standardIcon(QStyle.SP_Media
Stop))
        self.stop_btn.clicked.connect(self.stop)

        self.back_btn = QPushButton("⏪ 2c")
        self.forward_btn = QPushButton("2c ⏩")
        self.back_btn.clicked.connect(lambda:
self.seek_relative(-2000))
        self.forward_btn.clicked.connect(lambda:
self.seek_relative(2000))

        controls.addWidget(self.play_btn)
        controls.addWidget(self.pause_btn)
        controls.addWidget(self.stop_btn)
        controls.addSpacing(10)
        controls.addWidget(self.back_btn)
        controls.addWidget(self.forward_btn)
        controls.addStretch()

        main_layout.addLayout(top)
        main_layout.addWidget(self.video_label, 1)
        main_layout.addWidget(self.timeline)
        main_layout.addLayout(time_row)
        main_layout.addLayout(controls)

    def initial_pick_required(self):
        if not self.pick_model():
            QMessageBox.warning(self, "Закрытие", "Модель не
выбрана.")
            self.close()
            return
        if not self.pick_video():
            QMessageBox.warning(self, "Закрытие", "Видео не
выбрано.")
            self.close()
            return
        self.load_video(self.video_path)

    def reload_all(self):
        self.pause()
```

```
        if not self.pick_model():
            return
        if not self.pick_video():
            return
        self.load_video(self.video_path)

    def pick_model(self):
        path, _ = QFileDialog.getOpenFileName(
            self,
            "Выбрать модель",
            existing_pick_dir(MODEL_PICK_CANDIDATES, ROOT),
            "PyTorch weights (*.pt);;All files (*.*)",
        )
        if not path:
            return False

        try:
            self.invalidate_pending_inference()
            self.model = YOLO(path)
            self.model_path = path
            self.model_path_label.setText(Path(path).name)
            self.model_path_label.setToolTip(path)
            self.model_path_label.setStyleSheet("color:
white;")
            return True
        except Exception as e:
            QMessageBox.critical(self, "Ошибка", f"Не удалось
загрузить модель:\n{e}")
            return False

    def pick_video(self):
        path, _ = QFileDialog.getOpenFileName(
            self,
            "Выбрать видео",
            str(ROOT),
            "Video files (*.mp4 *.mkv *.avi *.mov *.webm);;All
files (*.*)",
        )
        if not path:
            return False

        self.video_path = path
        self.video_path_label.setText(Path(path).name)
        self.video_path_label.setToolTip(path)
        self.video_path_label.setStyleSheet("color: white;")
        return True

    def load_video(self, path: str):
        self.pause()
```

```
        self.invalidate_pending_inference()
        self.advice_window.clear_all()

        if self.cap is not None:
            self.cap.release()
            self.cap = None

        self.cap = cv2.VideoCapture(path)
        if not self.cap.isOpened():
            QMessageBox.critical(self, "Ошибка", "Не удалось
открыть видео.")
            return

        self.fps = self.cap.get(cv2.CAP_PROP_FPS) or 25.0
        self.total_frames =
int(self.cap.get(cv2.CAP_PROP_FRAME_COUNT) or 0)
        self.frame_idx = 0
        self.last_infer = -999999
        self.t0 = time.time()

        duration_ms = int((self.total_frames / self.fps) *
1000) if self.fps > 0 else 0
        self.timeline.setRange(0, max(self.total_frames - 1,
0))
        self.timeline.setValue(0)
        self.current_time_label.setText("00:00")

self.duration_label.setText(self.ms_to_time(duration_ms))

        ok, frame = self.cap.read()
        if ok:
            self.current_frame = frame.copy()
            self.render_current_frame()
            self.cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
            self.frame_idx = 0

    def play(self):
        if self.cap is None or self.model is None:
            return
        interval = max(1, int(1000 / self.fps))
        self.timer.start(interval)
        self.paused = False

    def pause(self):
        self.timer.stop()
        self.paused = True

    def stop(self):
        self.pause()
```

```
        self.invalidate_pending_inference()
        self.advice_window.clear_all()
        if self.cap is not None:
            self.cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
            self.frame_idx = 0
            self.timeline.setValue(0)
            self.current_time_label.setText("00:00")

            ok, frame = self.cap.read()
            if ok:
                self.current_frame = frame.copy()
                self.render_current_frame()
                self.cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
                self.frame_idx = 0

    def next_frame(self):
        if self.cap is None:
            return

        if self.pending_infer is not None:
            self.consume_finished_inference()
            if self.pending_infer is not None:
                return

        ok, frame = self.cap.read()
        if not ok:
            self.pause()
            return

        self.frame_idx =
int(self.cap.get(cv2.CAP_PROP_POS_FRAMES)) - 1
        self.current_frame = frame.copy()
        self.render_current_frame()

        if not self.slider_is_pressed:
            self.timeline.setValue(self.frame_idx)

        current_ms = int((self.frame_idx / self.fps) * 1000) if
self.fps > 0 else 0

self.current_time_label.setText(self.ms_to_time(current_ms))

    def render_current_frame(self):
        if self.current_frame is None or self.model is None:
            return

        frame = self.current_frame.copy()

        if (self.frame_idx - self.last_infer) >= self.every:
```

```
            self.start_inference(frame, self.frame_idx)
            self.show_processed_frame(frame, [],
self.frame_idx)
            return

        self.advice_window.update_detected_classes([])
        self.show_processed_frame(frame, [], self.frame_idx)

    def start_inference(self, frame, frame_idx: int):
        if self.pending_infer is not None:
            return

        self.pending_frame_idx = frame_idx
        self.pending_infer = self.infer_executor.submit(
            run_yolo_inference,
            self.model,
            frame.copy(),
            frame_idx,
            self.inference_generation,
            self.conf,
            self.imgsz,
            self.device,
        )
        self.infer_poll_timer.start()

    def poll_inference(self):
        self.consume_finished_inference()
        if self.pending_infer is None:
            self.infer_poll_timer.stop()

    def consume_finished_inference(self) -> bool:
        if self.pending_infer is None or not
self.pending_infer.done():
            return False

        future = self.pending_infer
        self.pending_infer = None
        self.pending_frame_idx = None

        try:
            result = future.result()
        except Exception as e:
            self.pause()
            QMessageBox.critical(self, "Ошибка", f"Ошибка
анализа кадра:\n{e}")
            return True

        if result["generation"] != self.inference_generation:
            return True
```

```
        self.last_infer = result["frame_idx"]

self.advice_window.update_detected_classes(result["detected_nam
es"])
        self.show_processed_frame(
            result["annotated"],
            result["detections_lines"],
            result["frame_idx"],
        )
        return True

    def invalidate_pending_inference(self, reset_executor: bool
= True):
        self.inference_generation += 1
        if self.pending_infer is not None:
            self.pending_infer.cancel()
            self.pending_infer = None
            self.pending_frame_idx = None
        self.infer_poll_timer.stop()
        if reset_executor:
            self.infer_executor.shutdown(wait=False,
cancel_futures=True)
            self.infer_executor =
ThreadPoolExecutor(max_workers=1)

    def show_processed_frame(self, annotated, detections_lines:
list[str], frame_idx: int):
        dt = time.time() - self.t0
        eff_fps = frame_idx / dt if dt > 0 else 0.0
        status = f"frame={frame_idx}/{self.total_frames}
effFPS={eff_fps:.1f}  conf={self.conf:.2f}  imgsz={self.imgsz}"
        cv2.putText(annotated, status, (12, 28),
cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 0, 0), 4, cv2.LINE_AA)
        cv2.putText(annotated, status, (12, 28),
cv2.FONT_HERSHEY_SIMPLEX, 0.75, (255, 255, 255), 2,
cv2.LINE_AA)

        view = draw_sidebar(annotated, detections_lines,
width=420)
        self.current_view = view
        self.show_frame(view)

    def show_frame(self, frame_bgr):
        frame_rgb = cv2.cvtColor(frame_bgr, cv2.COLOR_BGR2RGB)
        h, w, ch = frame_rgb.shape
        bytes_per_line = ch * w
```

```
        qimg = QImage(frame_rgb.data, w, h, bytes_per_line,
QImage.Format_RGB888)
        pixmap = QPixmap.fromImage(qimg)

        scaled = pixmap.scaled(
            self.video_label.size(),
            Qt.KeepAspectRatio,
            Qt.SmoothTransformation,
        )
        self.video_label.setPixmap(scaled)

    def resizeEvent(self, event):
        super().resizeEvent(event)
        if self.current_view is not None:
            self.show_frame(self.current_view)

    def on_slider_pressed(self):
        self.slider_is_pressed = True
        self.was_playing_before_drag = not self.paused
        self.pause()

    def on_slider_released(self):
        self.slider_is_pressed = False
        self.seek_to_frame(self.timeline.value())
        if self.was_playing_before_drag:
            self.play()

    def on_slider_moved(self, value):
        current_ms = int((value / self.fps) * 1000) if self.fps
> 0 else 0

self.current_time_label.setText(self.ms_to_time(current_ms))

    def seek_to_frame(self, frame_idx: int):
        if self.cap is None:
            return

        self.invalidate_pending_inference()
        frame_idx = max(0, min(frame_idx, self.total_frames -
1))
        self.cap.set(cv2.CAP_PROP_POS_FRAMES, frame_idx)

        ok, frame = self.cap.read()
        if ok:
            self.frame_idx = frame_idx
            self.current_frame = frame.copy()
            self.render_current_frame()
            self.timeline.setValue(frame_idx)
```

```
            current_ms = int((frame_idx / self.fps) * 1000) if
self.fps > 0 else 0

self.current_time_label.setText(self.ms_to_time(current_ms))

    def seek_relative(self, delta_ms: int):
        if self.cap is None or self.fps <= 0:
            return
        delta_frames = int((delta_ms / 1000.0) * self.fps)
        target = max(0, min(self.total_frames - 1,
self.frame_idx + delta_frames))
        self.seek_to_frame(target)

    def closeEvent(self, event):
        self.invalidate_pending_inference(reset_executor=False)
        self.infer_executor.shutdown(wait=False,
cancel_futures=True)
        self.advice_window.close()
        super().closeEvent(event)

    @staticmethod
    def ms_to_time(ms: int) -> str:
        total_sec = max(0, ms // 1000)
        h = total_sec // 3600
        m = (total_sec % 3600) // 60
        s = total_sec % 60
        if h > 0:
            return f"{h:02}:{m:02}:{s:02}"
        return f"{m:02}:{s:02}"

def main():
    app = QApplication(sys.argv)
    window = YoloViewer()
    window.show()
    sys.exit(app.exec())

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

Додаток Б – Код системи підказок

```
from __future__ import annotations

from dataclasses import dataclass, field
import time

from PySide6.QtCore import Qt
```

```
from PySide6.QtWidgets import (
    QFrame,
    QLabel,
    QScrollArea,
    QVBoxLayout,
    QWidget,
)

@dataclass(frozen=True)
class AdviceDefinition:
    title: str
    summary: str
    tips: tuple[str, ...] = field(default_factory=tuple)

DEFAULT_ADVICE: dict[str, AdviceDefinition] = {
    "crosswalk": AdviceDefinition(
        title="Пішохідний перехід",
        summary="Попереду зона, де пішоходи можуть вийти на
дорогу.",
        tips=(
            "Завчасно зменш швидкість і будь готовий
зупинитися.",
            "Перевір узбіччя, тротуари та острівці безпеки.",
            "Не обганяй і не прискорюйся перед переходом.",
        ),
    ),
    "children_on_road": AdviceDefinition(
        title="Діти на дорозі",
        summary="Поруч може бути школа, майданчик або ділянка з
непередбачуваною появою дітей.",
        tips=(
            "Суттєво знизь швидкість і збільш дистанцію.",
            "Очікуй раптовий вихід дитини з-за авто, зупинки
або перешкоди.",
            "Тримай ногу ближче до гальма і не відволікайся.",
        ),
    ),
    "speed_limit": AdviceDefinition(
        title="Обмеження швидкості",
        summary="Знак задає максимально дозволену швидкість на
ділянці.",
        tips=(
            "Плавно приведи швидкість до дозволеної.",
            "Врахуй покриття, трафік, погоду та видимість.",
            "Не відкладай гальмування до самого знака або
камери контролю.",
        ),
    ),
    "speed bump": AdviceDefinition(
```

```
        title="Штучна нерівність",
        summary="Попереду лежачий поліцейський або ділянка, де
треба зменшити швидкість.",
        tips=(
            "Завчасно сповільнися без різкого гальмування в
останній момент.",
            "Проїжджай нерівність прямо, без різких маневрів.",
            "Пильнуй пішоходів: такі знаки часто поруч із
переходами або школами.",
        ),
    ),
    "speed_bump": AdviceDefinition(
        title="Штучна нерівність",
        summary="Попереду лежачий поліцейський або ділянка, де
треба зменшити швидкість.",
        tips=(
            "Завчасно сповільнися без різкого гальмування в
останній момент.",
            "Проїжджай нерівність прямо, без різких маневрів.",
            "Пильнуй пішоходів: такі знаки часто поруч із
переходами або школами.",
        ),
    ),
    "turn_sign": AdviceDefinition(
        title="Напрямок руху або поворот",
        summary="Знак підказує дозволений або рекомендований
напрямок руху.",
        tips=(
            "Заздалегідь займи потрібну смугу.",
            "Увімкни покажчик повороту та перевір дзеркала.",
            "Не змінюй траєкторію різко, якщо пропустив
потрібний напрямок.",
        ),
    ),
    "bus_stop": AdviceDefinition(
        title="Зупинка громадського транспорту",
        summary="Поруч можуть бути автобус, пасажири та
пішоходи біля проїзної частини.",
        tips=(
            "Зменш швидкість і контролюй правий край дороги.",
            "Очікуй пішоходів, які можуть виходити перед або за
автобусом.",
            "Не перекривай зупинку та не виконуй небезпечні
маневри поруч.",
        ),
    ),
    "patrol": AdviceDefinition(
        title="Патруль або контроль",
```

```
        summary="Можлива присутність поліції, контролю
швидкості або регулювання руху.",
        tips=(
            "Перевір швидкість, дистанцію та дотримання
правил.",
            "Будь готовий до сигналів регулювальника або
зупинки транспорту попереду.",
            "Не роби різких маневрів і зберігай передбачувану
траєкторію.",
        ),
    ),
    "parking": AdviceDefinition(
        title="Паркування дозволено",
        summary="Знак позначає місце або зону, де паркування
може бути дозволене.",
        tips=(
            "Перевір додаткові таблички: час, спосіб
постановки, платність.",
            "Паркуйся так, щоб не заважати руху й огляду іншим
водіям.",
            "Перед маневром переконайся, що позаду немає
велосипедистів чи пішоходів.",
        ),
    ),
    "parking ban": AdviceDefinition(
        title="Паркування заборонено",
        summary="На цій ділянці зупинка або стоянка можуть бути
обмежені.",
        tips=(
            "Не плануй паркування в зоні дії знака.",
            "Перевір межі дії знака та додаткові таблички.",
            "Якщо треба висадити пасажира, переконайся, що
коротка зупинка дозволена.",
        ),
    ),
    "parking_ban": AdviceDefinition(
        title="Паркування заборонено",
        summary="На цій ділянці зупинка або стоянка можуть бути
обмежені.",
        tips=(
            "Не плануй паркування в зоні дії знака.",
            "Перевір межі дії знака та додаткові таблички.",
            "Якщо треба висадити пасажира, переконайся, що
коротка зупинка дозволена.",
        ),
    ),
}

class AdviceWindow(QWidget):
```

```
    def __init__(
        self,
        advice_map: dict[str, AdviceDefinition] | None = None,
        hold_seconds: float = 3.0,
    ):
        super().__init__()
        self.advice_map = advice_map or DEFAULT_ADVICE
        self.hold_seconds = hold_seconds
        self._active_signs: dict[str, float] = {}

        self.setWindowTitle("Поради щодо дорожніх знаків")
        self.resize(460, 760)
        self.setMinimumWidth(380)
        self.setStyleSheet(
            """
            QWidget {
                background: #0f1720;
                color: #ecf3ff;
            }
            QLabel#Heading {
                font-size: 20px;
                font-weight: 700;
                color: #f8fbff;
            }
            QLabel#Hint {
                color: #93a4ba;
                font-size: 13px;
            }
            QFrame#AdviceCard {
                background: #172331;
                border: 1px solid #2b435d;
                border-radius: 16px;
            }
            QLabel#CardTitle {
                font-size: 17px;
                font-weight: 700;
                color: #f8fbff;
            }
            QLabel#CardSummary {
                color: #d5e1f0;
                font-size: 14px;
            }
            QLabel#CardMeta {
                color: #8fb8ff;
                font-size: 12px;
                font-weight: 600;
            }
            QLabel#CardTip {
                color: #d9e5f3;
```

```
                font-size: 13px;
            }
            """
        )

        root = QVBoxLayout(self)
        root.setContentsMargins(16, 16, 16, 16)
        root.setSpacing(12)

        heading = QLabel("Підказки за знайденими знаками")
        heading.setObjectName("Heading")
        root.addWidget(heading)

        hint = QLabel("Поради залишаються на екрані ще кілька
секунд після зникнення знака.")
        hint.setWordWrap(True)
        hint.setObjectName("Hint")
        root.addWidget(hint)

        self.empty_label = QLabel("Поки немає активних
підказок. Запусти відео або дочекайся детекцій.")
        self.empty_label.setWordWrap(True)
        self.empty_label.setObjectName("Hint")
        root.addWidget(self.empty_label)

        self.scroll = QScrollArea()
        self.scroll.setWidgetResizable(True)
        self.scroll.setFrameShape(QFrame.NoFrame)

        self.content = QWidget()
        self.cards_layout = QVBoxLayout(self.content)
        self.cards_layout.setContentsMargins(0, 0, 0, 0)
        self.cards_layout.setSpacing(12)
        self.cards_layout.setAlignment(Qt.AlignTop)

        self.scroll.setWidget(self.content)
        root.addWidget(self.scroll, 1)

        self.render_cards()

    def update_detected_classes(self, class_names: list[str]) -
> None:
        now = time.monotonic()
        for name in class_names:
            normalized = name.strip().lower()
            if normalized:
                self._active_signs[normalized] = now

        expired = [
```

```
            name
            for name, seen_at in self._active_signs.items()
            if now - seen_at > self.hold_seconds
        ]
        for name in expired:
            del self._active_signs[name]

        self.render_cards()

    def clear_all(self) -> None:
        self._active_signs.clear()
        self.render_cards()

    def render_cards(self) -> None:
        while self.cards_layout.count():
            item = self.cards_layout.takeAt(0)
            widget = item.widget()
            if widget is not None:
                widget.deleteLater()

        active_names = sorted(
            self._active_signs,
            key=lambda name: self._active_signs[name],
            reverse=True,
        )

        self.empty_label.setVisible(not active_names)
        self.scroll.setVisible(bool(active_names))

        for name in active_names:
            self.cards_layout.addWidget(self.build_card(name))

        self.cards_layout.addStretch(1)

    def build_card(self, class_name: str) -> QFrame:
        advice = self.advice_map.get(
            class_name,
            AdviceDefinition(
                title=class_name.replace("_", " ").title(),
                summary="Для цього класу ще немає окремого
сценарію. Дій обережно й оцінюй дорожню обстановку.",
                tips=(
                    "Тримай безпечну швидкість і дистанцію.",
                    "Стеж за розміткою, знаками та діями інших
учасників руху.",
                ),
            ),
        )
```

```
        seconds_left = max(0.0, self.hold_seconds -
(time.monotonic() - self._active_signs[class_name]))

        card = QFrame()
        card.setObjectName("AdviceCard")

        layout = QVBoxLayout(card)
        layout.setContentsMargins(16, 16, 16, 16)
        layout.setSpacing(8)

        title = QLabel(advice.title)
        title.setObjectName("CardTitle")
        title.setWordWrap(True)
        layout.addWidget(title)

        meta = QLabel(f"Клас моделі: {class_name}  |  зникне
через ~{seconds_left:.1f} с")
        meta.setObjectName("CardMeta")
        meta.setWordWrap(True)
        layout.addWidget(meta)

        summary = QLabel(advice.summary)
        summary.setObjectName("CardSummary")
        summary.setWordWrap(True)
        layout.addWidget(summary)

        for tip in advice.tips:
            tip_label = QLabel(f"• {tip}")
            tip_label.setObjectName("CardTip")
            tip_label.setWordWrap(True)
            layout.addWidget(tip_label)

        return card
```

Додаток В – Код навчання моделі

```
from datetime import datetime
from pathlib import Path
import shutil

from ultralytics import YOLO

ROOT = Path(__file__).resolve().parent.parent
RUNS_ROOT = ROOT / "runs" / "models"

def build_run_name() -> str:
    return datetime.now().strftime("train_%Y%m%d_%H%M%S")
```

```
def organize_run_outputs(save_dir: Path) -> None:
    artifacts_dir = save_dir / "training_artifacts"
    artifacts_dir.mkdir(exist_ok=True)

    for path in save_dir.iterdir():
        if path.name in {"weights", "training_artifacts"}:
            continue
        shutil.move(str(path), str(artifacts_dir / path.name))

def main():
    run_name = build_run_name()
    model = YOLO(ROOT / "yolov8n.pt")
    results = model.train(
        data=ROOT / "training" / "data.yaml",
        epochs=50,
        imgsz=640,
        project=RUNS_ROOT,
        name=run_name,
        workers=0,
    )
    save_dir = Path(getattr(results, "save_dir", RUNS_ROOT /
run_name))
    organize_run_outputs(save_dir)
    print(f"Training run saved to: {save_dir}")
    print(f"Weights: {save_dir / 'weights'}")
    print(f"Graphs and reports: {save_dir /
'training_artifacts'}")

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