МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ
Циклова комісія комп’ютерної інженерії та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему

ПРОГРАМНИЙ ПРОТОТИП МОДЕЛЮВАННЯ БЕЗПІЛОТНОГО КЕРУВАННЯ АВТОТРАНСПОРТОМ

Виконала: студентка групи 1П-22
Спеціальності
121 Інженерія програмного забезпечення
Тимур СУЩЕНКО
Керівник:
Майя ЛЮТА

Черкаси 2026

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ

Циклова комісія комп'ютерної інженерії та інформаційних технологій

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувачка циклової комісії КІ та ІТ

_____ Наталя ФАЛЬЧЕНКО

«_____» _____ 2025 р.

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

Сущенку Тимуру Руслановичу

1. Тема кваліфікаційної роботи Програмний прототип моделювання безпілотного керування автотранспортом

Керівник роботи Люта Майя В'ячеславівна, викладач вищої категорії затверджені наказом закладу фахової передвищої освіти від «06» жовтня 2025 року № 84/1-у.

2. Строк подання студентом кваліфікаційної роботи 02.06.2026

3. Вихідні дані до кваліфікаційної роботи: мова програмування Python, бібліотеки OpenCV, NumPy, Pandas, PyTorch, Ultralytics YOLOv8, засоби обробки відеоданих та телеметрії, формати зберігання даних CSV і JSON, середовище розробки Visual Studio Code, система контролю версій Git, засоби моделювання UML у нотації PlantUML (діаграми прецедентів, компонентів, діяльності, послідовності та станів), веббраузери Google Chrome, Mozilla Firefox, Microsoft Edge, наукові та нормативні джерела з автономного керування транспортними засобами (SAE J3016, ISO 26262, ISO 21448 SOTIF, ASAM OpenSCENARIO, OpenDRIVE)..

4. Зміст кваліфікаційної роботи: Вступ. Концептуальні та технологічні засади програмної системи безпілотного керування. Проєктування програмного прототипу. Реалізація та критичне оцінювання. Висновки

5. Дата видачі завдання 15.09.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Терміни виконання етапів	Примітка про виконання
1	Вступ	14.10.2025	
2	Розділ 1 Концептуальні та технологічні засади програмної системи безпілотного керування	09.12.2025	
3	Розділ 2 Проєктування програмного прототипу	10.03.2026	
4	Розділ 3 Реалізація та критичне оцінювання	28.04.2026	
6	Висновки	12.05.2026	
7	Оформлення кваліфікаційної роботи (чистовий варіант)	26.05.2026	
8	Перевірка кваліфікаційної роботи на наявність ознак плагіату (за 10 днів до захисту)	02.06.2026	
9	Подання кваліфікаційної роботи на затвердження завідувачці циклової комісії (за 7 днів до захисту)	10.06.2026	

Студент _____ Тимур СУЩЕНКО

Керівник роботи _____ Майя ЛЮТА

АНОТАЦІЯ

Кваліфікаційну роботу присвячено розробці програмного прототипу моделювання безпілотного керування автотранспортом. У роботі автономне керування розглядається як комплекс програмних компонентів, що здійснюють обробку сенсорних даних, розпізнавання об’єктів, формування керувальних команд, перевірку їх відповідності умовам експлуатації та накопичення результатів для подальшого аналізу.

Проведено аналіз предметної області автономного транспорту, рівнів автоматизації, концепції ODD, особливостей camera-only підходу, використання CAN-телеметрії, сценарного тестування та підходів до забезпечення безпеки. На основі сучасних стандартів і наукових досліджень сформовано вимоги до програмного прототипу та обґрунтовано його архітектуру.

Практичним результатом роботи є програмний прототип, що включає модулі завантаження та обробки відео і телеметрії, детекції об’єктів, прогнозування керувальних дій, контролю обмежень, журналювання та формування звітів. Для опису архітектури системи розроблено UML-діаграми в нотації PlantUML.

Отримані результати можуть використовуватися як навчально-дослідний інструмент для демонстрації принципів автономного керування, побудови ML-компонентів і сценарного тестування. Визначено обмеження прототипу та перспективи його подальшого розвитку.

Ключові слова: АВТОНОМНЕ КЕРУВАННЯ, БЕЗПІЛОТНИЙ ТРАНСПОРТНИЙ ЗАСІБ, ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ODD, SOTIF, CNN, YOLOV8, END-TO-END LEARNING, PLANTUML, СЦЕНАРНЕ ТЕСТУВАННЯ.

ANNOTATION

This qualification thesis is devoted to the development of a software prototype for autonomous vehicle control simulation. The study considers autonomous driving as a set of software components that process sensor data, perform object recognition, generate control commands, verify their compliance with operating conditions, and collect results for further analysis.

The research analyzes the domain of autonomous transportation, levels of driving automation, the Operational Design Domain (ODD) concept, camera-only approaches, CAN telemetry, scenario-based testing, and safety assurance methods. Based on current standards and scientific studies, the requirements for the software prototype were defined and its architecture was justified.

The practical outcome of the work is a software prototype that includes modules for video and telemetry acquisition and processing, object detection, control action prediction, constraint checking, logging, and report generation. UML diagrams developed in PlantUML notation were used to describe the system architecture.

The obtained results can be used as an educational and research tool for demonstrating the principles of autonomous driving, machine learning components, and scenario-based testing. The limitations of the prototype and directions for its further development are also identified.

Keywords: autonomous driving, autonomous vehicle, software engineering, ODD, SOTIF, CNN, YOLOv8, end-to-end learning, PlantUML, scenario-based testing.

Keywords: AUTONOMOUS DRIVING, AUTONOMOUS VEHICLE, SOFTWARE ENGINEERING, ODD, SOTIF, CNN, YOLOV8, END-TO-END LEARNING, PLANTUML, SCENARIO-BASED TESTING.

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ADAS – Advanced Driver Assistance Systems, системи допомоги водію

ADS – Automated Driving System, автоматизована система керування рухом

АТЗ – автономний транспортний засіб

BEV – Bird’s-Eye View, подання сцени з висоти пташиного польоту

CAN – Controller Area Network, автомобільна мережа контролерів

CNN / ЗНМ – Convolutional Neural Network, згорткова нейронна мережа

E2E – end-to-end, наскрізний підхід до навчання моделі

FoV – Field of View, поле зору датчика

GNSS – Global Navigation Satellite System

IMU – Inertial Measurement Unit, інерційний блок вимірювання

ODD – Operational Design Domain, операційна область застосування ADS

SOTIF – Safety of the Intended Functionality, безпека передбачуваної функціональності

UML – Unified Modeling Language, уніфікована мова моделювання

V2X – Vehicle-to-Everything, обмін даними транспортного засобу з іншими об’єктами інфраструктури

ЗМІСТ

ВСТУП

Автоматизоване керування автотранспортом належить до складних кіберфізичних напрямів, у яких програмне забезпечення безпосередньо впливає на поведінку технічної системи, безпеку руху, якість прийняття рішень та відповідальність за наслідки роботи алгоритмів. Для інженерії програмного забезпечення така тематика є показовою, оскільки поєднує аналіз предметної області, формалізацію вимог, архітектурне проєктування, реалізацію, тестування, оцінювання якості й документування результатів відповідно до підходів SWEBOK та інженерії вимог [1; 2].

Актуальність роботи зумовлена тим, що автономне керування вже не може розглядатися лише як задача розпізнавання об’єктів на зображенні. Сучасні програмні системи автономного водіння будуються як багаторівневі конвеєри, у яких поєднуються сприйняття сцени, прогнозування поведінки учасників руху, планування траєкторії, формування керувальних дій, контроль обмежень та збирання доказів якості. Оглядові дослідження підкреслюють, що розвиток галузі відбувається у напрямі зв’язування perception-planning-control, застосування end-to-end підходів, сенсорного злиття та BEV-представлень [3; 4; 5; 6; 7].

Разом із технічним прогресом зростають вимоги до коректного визначення меж застосування системи. Рівень автоматизації не можна встановлювати лише за фактом наявності нейромережі або демонстрації руху у відео. Потрібно враховувати операційну область застосування, типи доріг, погодні умови, допустимі сценарії, очікувану роль людини, механізми деградації та наявність засобів перевірки. Тому в цій роботі програмний результат визначається як навчально-дослідний прототип моделювання окремих ADS-функцій, а не як готовий автопілот реального транспортного засобу.

Метою кваліфікаційної роботи є розроблення та інженерне обґрунтування програмного прототипу моделювання безпілотного керування автотранспортом із визначеною операційною областю, формалізованими вимогами, модульною

архітектурою, нейромережевими компонентами сприйняття й керування, сценарним тестуванням та критичним оцінюванням результатів.

Об’єктом роботи є процес програмного моделювання функцій автономного керування автотранспортом у межах заданої операційної області. Предметом роботи є програмні засоби обробки сенсорних даних, розпізнавання об’єктів дорожньої сцени, формування керувальної дії, перевірки обмежень та оцінювання результатів роботи прототипу.

Для досягнення мети поставлено такі завдання: проаналізувати сучасний стан автономного керування та вимоги до програмних прототипів; визначити рівні автоматизації, ODD і межі відповідальності системи; сформувати функціональні та нефункціональні вимоги; побудувати UML-моделі програмної системи засобами PlantUML; обґрунтувати архітектуру компонентів і конвеєр даних; описати реалізацію модулів розпізнавання та керування; визначити тестові сценарії, метрики, ризики й напрями подальшого розвитку.

Методами роботи є аналіз і систематизація джерел, інженерія вимог, UML-моделювання, порівняльний аналіз методів автономного керування, прототипування програмних компонентів, сценарне тестування, якісне оцінювання ризиків та інтерпретація результатів з урахуванням меж застосування. Для практичної частини розглянуто Python, OpenCV, TensorFlow/Keras, модель YOLOv8, згорткові нейронні мережі, конфігураційні файли, журнали виконання та звітні артефакти.

Очікуваним результатом роботи є не ізольований фрагмент коду, а цілісний комплект інженерних артефактів: опис предметної області, таблиці вимог, UML-моделі, компонентна структура, схема конвеєра даних, тестова стратегія, опис програмних модулів, критерії приймання, ризики та рекомендації щодо розвитку. Саме поєднання цих артефактів дозволяє обґрунтувати програмну систему як результат інженерної діяльності.

Інженерна особливість роботи полягає у поєднанні двох рівнів аналізу. Перший рівень стосується машинного навчання: детекції об’єктів, згорткових нейронних мереж, прогнозування керування та обмежень end-to-end моделей.

Другий рівень стосується інженерії ПЗ: вимог, модульності, трасування, перевірок, відтворюваності запусків і пояснення меж інтерпретації результатів.

Практична цінність роботи полягає у створенні структурованого навчального прототипу, який демонструє не лише роботу окремої нейромережі, а й повний інженерний контекст: вхідні дані, архітектуру, вимоги, обмеження, тестування, метрики та пояснення отриманих результатів. Такий підхід дозволяє використовувати роботу в освітньому процесі для пояснення зв’язку між машинним навчанням та інженерією програмного забезпечення.

Додатково робота орієнтована на демонстрацію зв’язку між теоретичним аналізом і практичним програмним результатом. У ній не достатньо просто описати нейронні мережі або перелічити сенсори автономного автомобіля. Потрібно показати, як ці поняття перетворюються на вимоги, модулі, інтерфейси, стани, сценарії перевірки та звітні артефакти. Саме тому значну увагу приділено PlantUML-моделюванню, трасуванню вимог і сценарному оцінюванню.

У межах роботи використовується обережна термінологія. Поняття «безпілотне керування» в назві теми розглядається як прикладна сфера, тоді як фактичний результат визначається як програмний прототип моделювання. Такий підхід дозволяє уникнути некоректного прирівнювання навчальної реалізації до реальної автомобільної системи та водночас зберегти інженерну змістовність роботи.

Структурно дослідження побудовано так, щоб кожен наступний розділ спирався на попередній. Аналіз автономного керування формує підстави для вимог; вимоги визначають архітектуру; архітектура задає логіку реалізації; реалізація перевіряється через сценарії, метрики й ризики. Така послідовність відповідає життєвому циклу програмного забезпечення та дозволяє подати диплом як цілісну інженерну роботу.

РОЗДІЛ 1 КОНЦЕПТУАЛЬНІ ТА ТЕХНОЛОГІЧНІ ЗАСАДИ ПРОГРАМНОЇ СИСТЕМИ БЕЗПІЛОТНОГО КЕРУВАННЯ

1.1 Предметна область і інженерний фокус роботи

У межах цієї роботи предметна область не зводиться до опису автомобіля як механічного об’єкта. Вона розглядається як сукупність програмних, апаратних і організаційних рішень, які разом утворюють керувальний контур. Програмна частина такого контуру має отримати вхідні дані, перевірити їхню придатність, перетворити на ознаки, виконати розпізнавання або прогнозування, сформувати керувальну дію та зафіксувати результат для подальшого аналізу. Саме така послідовність дозволяє перевести загальну ідею безпілотного керування у формат інженерної задачі, де кожен модуль має призначення, межі відповідальності та перевірюваний результат.

Для програмної інженерії особливо важливо відокремити демонстраційний прототип від реальної safety-critical системи. Реальний автономний автомобіль повинен мати резервування датчиків, відмовостійке виконання, формалізовані сценарії тестування, валідацію на великій кількості дорожніх ситуацій і підтвердження відповідності стандартам. Навчальний прототип, навпаки, має показати логіку побудови системи, принципи роботи ML-компонентів і спосіб фіксації обмежень. Такий підхід не знижує практичну цінність роботи, а робить її коректною: результат оцінюється саме в межах заявленої ролі прототипу.

Інженерний фокус також передбачає наявність простежуваності між проблемою, вимогами, архітектурою, реалізацією та перевіркою. Якщо в предметній області визначено ризик некоректного прогнозу керма, то в проєктній частині має з’явитися компонент контролю допустимості, у тестовій частині - сценарій різкої команди, а у висновках - обмеження щодо використання моделі. Без такого зв’язку робота перетворюється на набір розрізнених описів. Тому подальші розділи побудовано як послідовний перехід від доменного аналізу до програмних артефактів.

Окремою характеристикою предметної області є залежність результатів від якості даних. Кадри дорожньої сцени можуть містити тіні, засвічення, розмиття, часткове перекриття об’єктів, нетипові ракурси або розмітку, яка відсутня в навчальній вибірці. Телеметрія може мати пропуски, різну частоту дискретизації або часовий зсув відносно відео. Саме тому прототип має не лише запускати модель, а й пояснювати, які передумови повинні бути виконані для інтерпретації результатів.

З погляду освітнього результату цінність прототипу полягає у відтворюваності. Студент або викладач має мати змогу повторити запуск, змінити параметри, переглянути журнал і порівняти отримані метрики з очікуваними. Для цього архітектура повинна бути не монолітною, а модульною: компоненти завантаження, попередньої обробки, детекції, керування, контролю та звітування мають бути логічно відокремлені.

Додатково предметну область варто розглядати через поняття експериментального циклу. Користувач не просто запускає модель, а проходить послідовність дій: готує дані, обирає сценарій, задає межі ODD, запускає обробку, отримує журнал і оцінює метрики. Такий цикл допомагає побачити, які саме програмні функції потрібні для контрольованого дослідження автономного керування. Він також пояснює, чому в роботі важливі не лише нейромережі, а й інтерфейси між модулями, формат звітності та правила обробки помилок.

У практичному сенсі прототип має виконувати роль стенда, на якому можна порівнювати різні варіанти моделей і параметрів. Якщо надалі замінити детектор, змінити поріг confidence або підключити інший набір кадрів, архітектура повинна дозволити зробити це без повного переписування системи. Такий підхід відповідає принципам повторного використання й поступового вдосконалення програмного забезпечення.

Для дипломної роботи важливо, щоб така постановка задачі залишалася послідовною в усіх розділах. Якщо в першому розділі автономне керування описується як кіберфізична система з обмеженнями, то в другому розділі ці обмеження мають перетворитися на вимоги й архітектурні рішення, а в третьому

- на конкретні модулі, журнали та метрики. Послідовність викладу допомагає читачеві бачити не лише окремі факти, а логіку інженерного розроблення.

Автономне керування автотранспортом поєднує комп'ютерний зір, обробку сигналів, автомобільні мережі, планування руху, машинне навчання, вбудоване програмування та інженерію безпеки. У межах спеціальності «Інженерія програмного забезпечення» ключовим є не лише опис фізичних датчиків або нейромережевих моделей, а й пояснення того, як ці елементи перетворюються на програмну систему з визначеними входами, виходами, вимогами, обмеженнями та критеріями перевірки.

Предметна область характеризується високою невизначеністю умов. Камера може втрачати якість через освітлення, туман або засвітлення; телеметрія може надходити з різною частотою; розмітка даних може містити похибки; нейромережева модель може демонструвати добрий результат на відомих прикладах, але помилятися на рідкісних сценаріях. Тому програмний прототип має бути побудований так, щоб явно фіксувати межі застосування й не створювати хибного враження повної автономності.

У роботі автономне керування розглядається як програмний контур, що приймає кадр дорожньої сцени й телеметричні ознаки, виконує попередню обробку, передає дані в модуль сприйняття, формує прогноз керувальної дії, перевіряє її в межах допустимих умов і записує результати експерименту. Така постановка дозволяє поєднати машинне навчання з типовими артефактами інженерії ПЗ: вимогами, архітектурою, UML-моделями, тестами, метриками й документацією.

Прототип не призначений для керування реальним автомобілем на дорогах загального користування. Його коректна роль - навчально-дослідне моделювання окремих ADS-функцій. Саме таке обмеження є важливим елементом професійної відповідальності, оскільки нейромережева модель без сертифікованого safety case, апаратного резервування, сценарної валідації та контролю деградації не може подаватися як готова система автономного водіння.

Інженерний фокус роботи охоплює чотири групи питань: аналіз технологічного контексту, формалізацію вимог, побудову архітектури й оцінювання результатів. Перша група відповідає на питання, які методи та стандарти важливі для автономного керування. Друга визначає, що саме повинен робити прототип. Третя пояснює, як модулі взаємодіють між собою. Четверта дозволяє встановити, наскільки отриманий результат відповідає заявленим межам.

З позиції життєвого циклу програмного забезпечення робота має горизонтально-поглиблений характер: вона не створює промисловий автомобільний продукт, але детально опрацьовує архітектуру, вимоги, моделювання, тестування й оцінювання ML-компонентів. Такий підхід відповідає навчальній меті фахового молодшого бакалавра, оскільки демонструє здатність поєднувати аналіз, проєктування та практичну реалізацію.

Таблиця 1.1 – Інженерна інтерпретація предметної області

Аспект	Зміст у межах роботи	Інженерне значення
Сенсорні дані	відеокадри, можлива CAN-телеметрія, часові мітки	визначають формат вхідних даних і вимоги до синхронізації
Сприйняття сцени	детекція автомобілів, пішоходів, дорожніх об’єктів	формує ознаки для подальшого прийняття рішення
Керувальна дія	прогноз кута керма або іншої команди	є виходом моделі та потребує перевірки меж
ODD	дорога, погода, освітлення, допустимі сценарії	визначає, де прототип можна коректно інтерпретувати
Тестування	модульні, інтеграційні та сценарні перевірки	підтверджує працездатність не лише успішного шляху
Оцінювання	MAE, FPS, confidence, кількість попереджень	дозволяє робити обережні, верифіковані висновки

1.2 Рівні автоматизації, ODD і межі відповідальності

Рівні автоматизації потрібні не для формального опису, а для правильної інтерпретації відповідальності. Система допомоги водію, яка лише підказує або стабілізує окрему дію, має інший профіль ризику, ніж система, яка самостійно приймає рішення про рух. Для дипломної роботи це означає, що не можна

переносити терміни промислового автономного водіння на навчальний прототип без уточнення. Якщо прототип прогнозує кут керма за кадром, це ще не робить його повноцінною ADS-системою, оскільки відсутні повний контур сприйняття, планування, резервування, fallback і юридично значуща валідація.

ODD у роботі виконує роль фільтра допустимості. Він визначає, для яких умов взагалі можна аналізувати результат моделі: тип дороги, освітлення, швидкісний режим, наявність розмітки, формат відео, припустимий інтервал телеметрії. Якщо вхідний сценарій виходить за ці межі, коректною поведінкою прототипу є не продовження впевненого керування, а попередження або блокування оцінки. Таке трактування наближає роботу до сучасної інженерної практики, де межі застосування є частиною вимог.

Для програмної системи важливо, щоб ODD був представлений не лише текстово, а й у вигляді параметрів, які можна використати під час запуску. Наприклад, у конфігурації можуть задаватися мінімальна якість кадру, допустимий діапазон швидкості, максимальна затримка телеметрії, перелік підтримуваних режимів і пороги confidence. Це дозволяє пов’язати опис предметної області з реальною логікою SafetyGuard та тестовими сценаріями.

Відповідальне формулювання рівня автономності також захищає роботу від завищених висновків. Якщо програмна система перевірена лише на короткому наборі даних, то висновок має стосуватися саме цього набору та заданого експериментального режиму. У тексті недоцільно вживати формулювання, які створюють враження готовності до дорожнього використання. Натомість слід говорити про демонстрацію принципів, структуризацію архітектури та сценарне оцінювання в контрольованому середовищі.

Поєднання понять SAE-рівнів, ODD, SOTIF і кібербезпеки показує, що автономне керування не є суто задачею нейромережі. Навіть точна модель сприйняття може бути небезпечною, якщо вона застосовується за межами призначених умов або не має механізму обробки невпевнених прогнозів. Тому в

роботі вводиться додатковий програмний шар контролю, який відокремлює вихід ML-моделі від остаточної оцінки допустимості команди.

Визначення меж відповідальності також стосується користувача прототипу. Оператор експерименту відповідає за вибір даних, перевірку конфігурації та інтерпретацію результатів, а система відповідає за коректне виконання запрограмованого сценарію. Якщо дані не відповідають ODD, відповідальною поведінкою є повідомлення про це, а не приховане продовження обробки. Така логіка робить модель роботи прозорою.

У дипломній роботі доцільно окремо підкреслювати, що рівень автономності не визначається однією функцією. Наявність детектора об’єктів, прогнозу керма або навіть короткої демонстрації руху не означає досягнення рівня L3 чи L4. Рівень автономності пов’язаний із повнотою динамічної задачі керування, відповідальністю за моніторинг середовища, механізмами fallback та межами застосування.

У контексті захисту це дозволяє аргументовано відповідати на питання про межі роботи. Якщо запитують, чи може прототип керувати реальним автомобілем, відповідь має спиратися на ODD, рівні автоматизації та відмінність між демонстраційною моделлю й сертифікованою системою. Така відповідь показує розуміння професійної відповідальності розробника програмного забезпечення.

Рівні автоматизації дорожніх транспортних засобів доцільно описувати за таксономією SAE J3016. Вона розмежовує рівні від 0 до 5 залежно від того, хто виконує динамічну задачу керування, хто контролює середовище, хто відповідає за fallback та в яких умовах система може діяти [8]. Для кваліфікаційної роботи важливо не підміняти цю класифікацію загальними формулюваннями про «безпілотність».

Рівень 3 передбачає умовну автоматизацію, у якій система виконує динамічну задачу керування в межах визначеного ODD, але може вимагати від людини прийняття керування. Рівень 4 передбачає, що система має власний fallback у межах ODD, а рівень 5 не обмежується конкретними умовами.

Навчальний прототип, який працює з відео, CNN і демонстраційними сценаріями, не має підстав класифікуватися як повноцінний ADS рівня 3 або вище.

ODD задає умови, у яких автоматизована система розроблена для функціонування. ISO 34503 описує ієрархічну таксономію операційної області: дорожнє середовище, об’єкти, умови освітлення, погодні чинники, динамічні учасники, геометрію дороги й інші параметри [9]. Для програмного прототипу ODD потрібний не як формальність, а як межа інтерпретації результатів.

Функціональна безпека і SOTIF доповнюють одне одного. ISO 26262 стосується небезпек, пов’язаних із відмовами електричних та електронних систем [10], тоді як ISO 21448 розглядає небезпеки справної системи, яка поводиться неправильно через обмеження сприйняття або алгоритмів [11]. Для ML-компонентів SOTIF особливо важливий, оскільки помилка може виникати без апаратної відмови.

Кібербезпека також є суттєвою для транспортних систем. ISO/SAE 21434 задає підхід до cybersecurity engineering у дорожніх транспортних засобах [12]. У межах прототипу це означає потребу в перевірці вхідних файлів, обережному поводженні з конфігураціями, журналюванні помилок і недопущенні неконтрольованого виконання зовнішніх даних.

Відповідальне формулювання меж роботи передбачає, що прототип демонструє алгоритмічний принцип і програмну структуру, але не гарантує безпечне керування в реальному середовищі. Така позиція є не недоліком, а ознакою коректного інженерного опису: робота показує, які компоненти потрібні для переходу від демонстрації до більш зрілої системи.

Таблиця 1.2 – Робоче визначення ODD програмного прототипу

Параметр ODD	Прийняте обмеження	Причина обмеження
Тип середовища	демонстраційна дорожня сцена або симуляційний фрагмент	спрощення перевірки й відтворюваності
Швидкісний режим	низькі або середні швидкості в навчальному сценарії	зменшення ризику різких маневрів у моделі

Продовження таблиці 1.2

Сенсори	камера як базове джерело, телеметрія як допоміжне джерело	доступність даних і простота реалізації
Погодні умови	без гарантій роботи в тумані, снігу, сильному дощі	camera-only підхід чутливий до погоди
Динамічні об’єкти	автомобілі, пішоходи, базові перешкоди	ці класи доступні для детекторів і тестових сценаріїв
Fallback	попередження або блокування команди	прототип не має фізичного виконавчого контуру

1.3 Сенсорна архітектура та сучасні методи автономного керування

Сенсорна архітектура автономного транспорту визначає не лише якість сприйняття, а й складність програмної системи. Камери забезпечують багату візуальну інформацію, але залежать від освітлення та погодних умов. Радар краще працює в складних погодних умовах, але має нижчу просторову деталізацію. Лідар формує точну геометрію простору, однак збільшує вартість і обсяг даних. Тому вибір camera-only підходу в навчальному прототипі є прийнятним, але повинен супроводжуватися критичним описом обмежень.

CAN-телеметрія в такій системі виконує роль контексту, який допомагає інтерпретувати відео. Один і той самий кадр може вимагати різних дій залежно від швидкості, попереднього кута керма або режиму руху. У демонстраційному прототипі телеметрія може бути спрощеною, але її наявність важлива для правильної постановки інженерної задачі: автономне керування працює не з ізольованим зображенням, а з часовою послідовністю станів транспортного засобу.

Модульний і наскрізний підходи мають різні переваги. Модульна архітектура краще пояснюється, оскільки окремо видно детекцію, оцінювання сцени, прийняття рішення та контроль. Наскрізна модель простіша за кількістю ручних правил, але важче пояснюється й сильніше залежить від навчальних даних. У роботі доцільно використовувати гібридну інтерпретацію: ML-компоненти формують прогнози, а програмна архітектура забезпечує обмеження, журналювання та контроль відхилень.

Сучасні архітектури автономного водіння дедалі частіше використовують злиття датчиків, трансформерні блоки, BEV-представлення та планувально орієнтовані моделі. Проте перенесення таких підходів у фахову кваліфікаційну роботу має бути обґрунтованим. Якщо обчислювальні ресурси, датасети й час обмежені, важливіше коректно реалізувати невеликий, але пояснюваний прототип, ніж декларативно згадати складні методи без зв'язку з практичною частиною.

Отже, сенсорний і алгоритмічний аналіз у роботі використовується не як самостійний огляд, а як основа для проєктних рішень. Саме з нього випливають вимоги до вхідних даних, формат FramePacket, потреба в попередній обробці, наявність SafetyGuard, перелік метрик і обережне трактування результатів.

Використання лише камери робить прототип простішим і доступнішим для навчального середовища, але водночас збільшує залежність від умов зйомки. Саме тому в роботі важливо не приховувати обмеження camera-only підходу. Вони мають бути винесені в ODD, тестові сценарії та інтерпретацію метрик. Така відвертість є ознакою якісної інженерної роботи.

Алгоритмічний рівень повинен враховувати часову природу керування. Окремий кадр дає лише моментальний зріз сцени, тоді як рух транспортного засобу залежить від попередніх станів. У спрощеному прототипі це можна врахувати через телеметрію, часові мітки та аналіз послідовності кадрів. Повна рекурентна або трансформерна модель може бути наступним етапом розвитку.

Звідси випливає ще один важливий принцип: складність методу має відповідати ресурсам і цілям роботи. У роботі поєднано відносно просту CNN-модель керування з актуальним детектором YOLOv8, оскільки така комбінація дає змогу показати повний інженерний конвеєр без переходу до надмірно складної промислової ADS-архітектури. Важливо, що YOLOv8 розглядається не як ізольована модель, а як компонент із чітким входом, виходом, порогами confidence і місцем у сценарному оцінюванні [26].

Сенсорна архітектура автономного транспортного засобу зазвичай поєднує камери, радар, лідар, GNSS/IMU, одометрію та автомобільні мережі. Кожен

сенсор має власну фізичну природу, частоту оновлення, похибки й обмеження. Камера забезпечує багатий семантичний опис сцени, але залежить від освітлення та погоди. Радар краще працює у складних погодних умовах і вимірює швидкість, але гірше передає семантику. Лідар забезпечує точну геометрію, однак має вартісні й погодні обмеження.

CAN-шина використовується як джерело телеметрії: швидкості, кута керма, стану педалей, технічних параметрів і часових міток. Для навчання моделі важлива синхронізація відеокадрів і телеметричних повідомлень. Якщо синхронізація порушується, модель може навчитися помилкових залежностей між дорожньою сценою та керувальною дією.

Сучасні методи автономного керування умовно поділяють на модульні, наскрізні та гібридні. Модульна архітектура краще пояснюється та тестується, але накопичує помилки між етапами. End-to-end підхід оптимізує модель за цільовою дією, але може страждати від причинної плутанини, низької інтерпретованості й недостатньої стійкості до рідкісних сценаріїв [3; 4].

Класичний підхід end-to-end був популяризований роботами, у яких нейромережа напряму прогнозувала керувальну дію за зображенням з камери [13; 14]. Для навчальної роботи такий підхід зручний, оскільки дозволяє показати зв’язок між зоровими даними і керуванням. Однак у реальній системі він має бути доповнений контролем обмежень, сценарною перевіркою і журналюванням невизначеності.

Сучасніші дослідження демонструють перехід до трансформерних архітектур, мультисенсорного злиття і BEV-представлень. TransFuser інтегрує зображення та LiDAR-ознаки через механізм уваги [5], UniAD формує planning-oriented підхід до повного стека автономного водіння [6], а BEVFusion уніфікує багатомодальні ознаки у просторі з висоти пташиного польоту [7].

Для даної кваліфікаційної роботи сучасні комплексні моделі не є обов’язковими для повної реалізації, але вони потрібні для критичного порівняння. На цьому тлі CNN і YOLOv8 використовуються як доступні й практично відтворювані компоненти навчального прототипу: CNN демонструє

принцип прогнозування керування, а YOLOv8 забезпечує актуальний рівень об’єктної детекції в межах Python-екосистеми та підтримує роботу з попередньо навченими вагами [26].

Таким чином, інженерно коректна позиція полягає в тому, щоб поєднати доступні засоби реалізації з актуальним розумінням галузі: реалізувати навчальний прототип на Python/OpenCV/TensorFlow, але оцінювати його з урахуванням сучасних вимог до ODD, тестування, сенсорного злиття та відповідальності за межі застосування.

Таблиця 1.3 – Критичне порівняння підходів автономного керування

Підхід	Переваги	Обмеження	Роль у роботі
Модульний	пояснюваність, локальне тестування, простіша заміна компонентів	накопичення помилок між модулями	основа архітектури прототипу
End-to-end	простий зв’язок між кадром і керуванням, навчальна наочність	проблеми інтерпретації та доменного зсуву	демонстраційна модель керування
Гібридний	поєднання модульності та оптимізації цілі	складніший дизайн і валідація	орієнтир для розвитку
BEV / sensor fusion	краще просторове уявлення сцени	потребує більше даних і ресурсів	сучасний напрям порівняння
Foundation models	можливість сценарної генерації та багатомодального аналізу	обчислювальна складність і питання надійності	перспектива дослідження

У першому розділі визначено, що програмний прототип безпілотного керування слід розглядати як систему з чіткими межами, а не як готовий автопілот. Основними межами є ODD, типи сенсорів, характер даних, відсутність фізичного виконавчого контуру та необхідність перевірки кожної керувальної дії.

Аналіз стандартів SAE, ISO та ASAM показав, що для коректного опису автономного керування потрібні не лише поняття нейромережі або датчика, а й категорії рівня автоматизації, операційної області, функціональної безпеки, SOTIF, кібербезпеки й сценарного тестування.

Критичний огляд методів показав, що CNN та YOLOv8 можуть бути використані як обґрунтовані компоненти навчально-дослідного прототипу, якщо їхні результати оцінюються в межах заданої ODD і не переносяться безпосередньо на реальний дорожній рух. Сучасний розвиток галузі пов’язаний із мультисенсорним злиттям, BEV-представленнями, planning-oriented архітектурами та сценарною валідацією, тому реалізований прототип слід трактувати як інженерну модель для демонстрації принципів, а не як завершену ADS-систему.

РОЗДІЛ 2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ПРОТОТИПУ

2.1 Формалізація вимог до програмного прототипу

Формалізація вимог у цій роботі виконує роль мосту між предметною областю та реалізацією. Загальне твердження про «моделювання безпілотного керування» є занадто широким, тому воно розкладається на конкретні функції: імпорт даних, попередню обробку, запуск детектора, отримання прогнозу керування, перевірку обмежень, журналювання та формування звіту. Кожна з цих функцій має бути перевірюваною, інакше її неможливо об’єктивно підтвердити під час захисту.

Функціональні вимоги описують, що саме має робити система, а нефункціональні - якими характеристиками має володіти результат. Для прототипу автономного керування нефункціональні вимоги не менш важливі, ніж функціональні. Наприклад, відтворюваність запуску, зрозумілі повідомлення про помилки, модульність і журналювання визначають, чи можна використати систему як навчальний інженерний артефакт, а не просто як одноразовий скрипт.

Під час формулювання вимог необхідно уникати надто загальних критеріїв на кшталт «система повинна працювати швидко» або «модель повинна бути точною». Натомість краще вказувати спостережувані критерії: час обробки кадру фіксується в журналі, помилка керування обчислюється за MAE/RMSE, некоректний файл обробляється без аварійного завершення, а вихід за межі ODD породжує статус WARNING або BLOCKED.

У роботі вимоги групуються відповідно до життєвого циклу експерименту. Перша група стосується підготовки даних, друга - виконання ML-компонентів, третя - контролю допустимості, четверта - збереження результатів. Така класифікація зручна, бо безпосередньо відповідає майбутній архітектурі та тестовим сценаріям.

Діаграма прецедентів у цьому контексті потрібна не для декоративного подання, а для фіксації меж системи. Вона показує, що користувач не

втручається в кожну внутрішню операцію, а запускає сценарій, налаштовує параметри та отримує результат. Внутрішні дії - детекція, прогнозування, контроль і звітування - виконуються системою, тому саме вони мають бути відображені в архітектурі та тестах.

Під час захисту вимоги виконують ще й комунікативну функцію. Вони допомагають пояснити, що саме було зроблено, які функції перевірялися та чому певні можливості не заявляються. Якщо система не має реального керування виконавчими механізмами, це не є недоліком за умови, що така межа прямо зафіксована в вимогах і меті роботи.

Критерії приймання мають бути сформульовані так, щоб їх можна було перевірити без суб'єктивних оцінок. Наприклад, «система формує звіт» означає наявність конкретного файлу з переліком метрик, а «система обробляє помилки» означає, що негативний сценарій завершується керованим статусом і повідомленням у журналі. Це робить результат захищуваним.

Вимоги також створюють основу для оцінювання повноти роботи. Якщо після завершення розроблення можна показати, яка вимога реалізована яким модулем і яким тестом підтверджена, то результат стає значно переконливішим. Для кваліфікаційної роботи це особливо важливо, оскільки комісія оцінює не тільки працездатність програми, а й інженерну зрілість її опису.

Формалізація вимог є ключовою умовою переходу від загального опису автономного керування до інженерної моделі програмної системи. Вимоги визначають, які функції має виконувати прототип, які обмеження слід врахувати, які результати потрібно зберігати та якими критеріями оцінюється успішність реалізації. ISO/IEC/IEEE 29148 визначає інженерію вимог як процес виявлення, аналізу, специфікації, перевірки та супроводу вимог до системи [2].

Основним користувачем прототипу є дослідник або оператор експерименту, який налаштовує параметри запуску, завантажує дані, запускає детекцію й модель керування, переглядає попередження та аналізує результати. Додатковою зацікавленою стороною є керівник або рецензент, який оцінює

повноту артефактів, відтворюваність запуску та відповідність висновків фактичним можливостям прототипу.

Діаграма прецедентів, наведена на рисунку 2.1, показує функціональні межі системи. Вона відокремлює дії користувача від внутрішніх процедур перевірки й звітування. Це важливо, оскільки автономне керування не може демонструватися лише як запуск одного скрипту: система повинна мати сценарій використання, налаштування, перевірку обмежень і результат, який можна проаналізувати.

Рисунок 2.1 – Діаграма прецедентів програмного прототипу, побудована в нотації PlantUML

Таблиця 2.1 – Функціональні вимоги до програмного прототипу

Код	Вимога	Критерій приймання
FR-1	імпорт відеокадрів дорожньої сцени	кадри прочитано, помилки формату зафіксовано
FR-2	приймання або імітація CAN-телеметрії	кут керма, швидкість і часові мітки доступні
FR-3	попередня обробка даних	зображення нормалізовано, часові мітки узгоджено
FR-4	детекція об’єктів сцени	для кадру сформовано класи, рамки та confidence
FR-5	формування керувальної дії	вихід має числовий прогноз кута керма

Продовження таблиці 2.1

FR-6	перевірка допустимості команди	команда дозволена, попереджена або заблокована
FR-7	збереження результатів	збережено журнал, метрики, графіки та конфігурацію
FR-8	повторюваний демонстраційний запуск	користувач може відтворити експеримент за інструкцією

Таблиця 2.2 – Нефункціональні вимоги до програмного прототипу

Код	Характеристика	Критерій приймання
NFR-1	відтворюваність	версії бібліотек, параметри запуску і випадкові зерна зафіксовано
NFR-2	безпечна деградація	помилка кадру або телеметрії не завершує програму аварійно
NFR-3	модульність	детектор і модель керування можна замінити через інтерфейс
NFR-4	продуктивність	час обробки кадру вимірюється і записується
NFR-5	трасованість	вимоги пов’язані з тестами та артефактами звіту
NFR-6	зрозумілість	структура проєкту, конфігурація і README дозволяють відтворити запуск

Вимоги пов’язані з тестами через матрицю трасованості. Така матриця не замінює програмні тести, але показує, які перевірки потрібні для підтвердження кожної заявленої можливості. Для нейромережевого прототипу це особливо важливо: сама наявність моделі не доводить якість системи, якщо не визначено, на яких даних і за якими критеріями вона перевіряється.

Формалізовані вимоги також задають межі оцінювання. Якщо в роботі реалізовано демонстраційний прототип, то висновки мають стосуватися його відтворюваності, структури, точності в заданих умовах і готовності до розширення, а не безпечної роботи на дорогах загального користування.

2.2 Архітектурна модель та компонентна структура

Архітектура програмного прототипу повинна запобігати змішуванню різних відповідальностей в одному модулі. Якщо завантаження даних, обробка кадру, нейромережевий прогноз, контроль обмежень і побудова звіту реалізовані в одному файлі, такий проєкт важко перевіряти, змінювати й пояснювати. Тому в роботі прийнято компонентний підхід, за якого кожен модуль має власний інтерфейс і передає наступному модулю структурований результат.

DataLoader відповідає за первинну достовірність входів. Він не повинен виконувати детекцію або оцінювання, але має перевірити, чи існують файли, чи підтримується формат, чи можна отримати кадри та чи є телеметрія. Такий розподіл дозволяє відокремити помилки даних від помилок моделі. У журналі це має відображатися різними типами повідомлень.

Preprocessor перетворює дані до форми, зручної для моделей. Для відео це може бути зміна розміру, нормалізація значень пікселів, обрізання області інтересу, перетворення кольорового простору або синхронізація часових міток. Важливо, щоб ці дії були параметризованими, оскільки різні моделі можуть вимагати різного розміру входу та іншого порядку каналів.

ObjectDetector і DrivingModel є незалежними ML-компонентами. Детектор повертає класи, рамки та confidence, а модель керування повертає числову команду або вектор керування. SafetyGuard не повинен знати внутрішню архітектуру нейромереж, він працює лише з їхнім виходом і параметрами ODD. Такий принцип спрощує подальше оновлення YOLOv8 до іншої конфігурації, новішої моделі або спеціалізованого детектора без зміни логіки керування, журналювання й оцінювання.

Компонентна модель також допомагає пояснити тестування. Для кожного модуля можна визначити окремий набір перевірок: DataLoader перевіряється на некоректних файлах, Preprocessor - на стабільності формату виходу, ObjectDetector - на наявності очікуваних полів, SafetyGuard - на переходах між станами, ReportBuilder - на формуванні файлів. У результаті тестування стає похідним від архітектури.

Архітектурна модель також визначає напрям майбутнього рефакторингу. Якщо виявиться, що детектор працює повільно, його можна оптимізувати або замінити, не змінюючи формат звіту. Якщо потрібно додати нову метрику, це має стосуватися Evaluator або MetricsCollector, а не DataLoader. Такий поділ зменшує зв’язність і підвищує супроводжуваність системи.

Компонент SafetyGuard варто розглядати як приклад інженерного компромісу. Він не робить модель безпечною в промисловому сенсі, але вводить явний механізм контролю припустимості. Для навчального прототипу цього достатньо, щоб показати різницю між необмеженим виходом нейромережі та програмно контрольованим результатом експерименту.

Компонентна архітектура зменшує ризик неконтрольованого ускладнення проєкту. У прототипах машинного навчання часто трапляється ситуація, коли експериментальний код швидко росте, але втрачає структуру. Розділення на модулі дозволяє зберігати зрозумілість навіть тоді, коли додаються нові моделі, метрики або сценарії.

Архітектура прототипу будується за принципом розподілу відповідальності між компонентами. Окремо виділяються джерела даних, модулі попередньої обробки, розпізнавання, керування, перевірки обмежень і оцінювання. Такий поділ відповідає модульному стилю розроблення і дозволяє замінювати окремі ML-компоненти без переписування всієї системи.

Компонентна UML-діаграма на рисунку 2.2 уточнює взаємодію між модулями. Вона показує, що VideoSource і TelemetrySource не повинні напряму звертатися до нейромережевих моделей. Спочатку дані проходять через DataLoader і Preprocessor, після чого розподіляються між ObjectDetector і DrivingModel. SafetyGuard є окремим компонентом, який не навчає модель, але контролює допустимість її виходу.

Розміщення SafetyGuard між моделлю та виконанням експерименту є принциповим. Навіть у навчальному прототипі вихід нейромережі не слід автоматично трактувати як безпечну команду. Команда має пройти перевірку діапазону, статусу вхідних даних, confidence детекції та відповідності ODD.

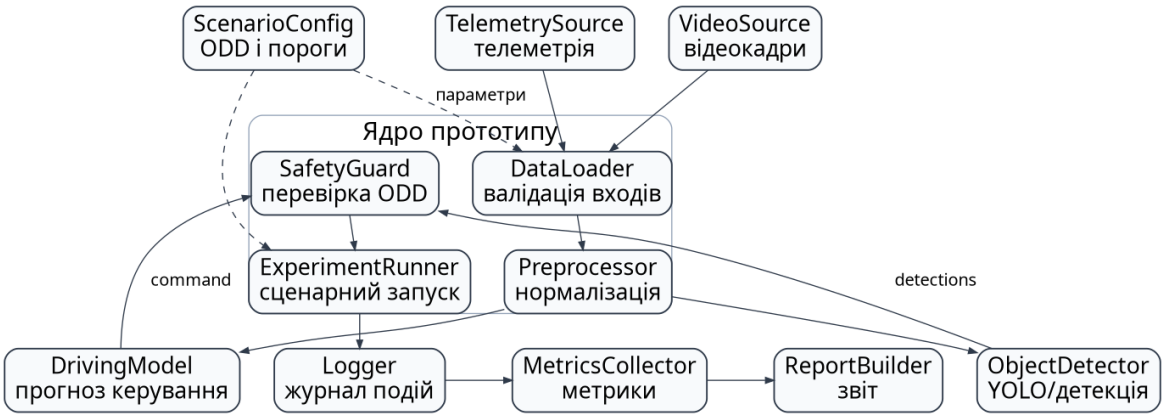


Рисунок 2.2 – Компонентна діаграма програмного прототипу, побудована в нотації PlantUML

Таблиця 2.3 – Основні компоненти архітектури

Компонент	Відповідальність	Вхід	Вихід
VideoSource	отримання відеокадрів	відеофайл або камера	послідовність кадрів
TelemetrySource	отримання телеметричних ознак	CSV/CAN-лог/імітація	швидкість, кут керма, timestamp
DataLoader	завантаження та первинна валідація	файли, конфігурація	FramePacket
Preprocessor	нормалізація і синхронізація	FramePacket	тензори й ознаки
ObjectDetector	розпізнавання об’єктів	кадр	класи, рамки, confidence
DrivingModel	прогноз керувальної дії	кадр, телеметрія	ControlCommand
SafetyGuard	перевірка меж і ODD	команда, detections, стан	ALLOWED/WARNING/BLOCKED
Evaluator	збір метрик і звітування	журнал, мітки, статуси	CSV, графіки, звіт

Таке розділення знижує зв’язаність і робить проєкт придатним для поетапного оновлення ML-компонентів. Якщо згодом виникне потреба перейти від YOLOv8 до новішої або спеціалізованої моделі тієї самої екосистеми, зміни мають стосуватися насамперед реалізації ObjectDetector, а не всього конвеєра, журналювання чи звітності.

Для реалізації компонентів доцільно використовувати конфігураційний файл. У ньому фіксуються шляхи до даних, пороги confidence, розміри зображень, допустимий діапазон керма, параметри запуску й режим оцінювання. Це підвищує відтворюваність, оскільки результати можна пов’язати не лише з кодом, а й з конкретними налаштуваннями експерименту.

2.3 Конвеєр даних і сценарій виконання експерименту

Конвеєр даних у прототипі автономного керування має бути детермінованим настільки, наскільки це можливо в межах ML-середовища. Одна й та сама конфігурація, той самий набір кадрів і ті самі ваги моделі повинні приводити до порівнюваних результатів. Якщо бібліотека або апаратне середовище вносить випадковість, це потрібно зафіксувати в описі експерименту, інакше повторний запуск може дати інші значення метрик без зрозумілої причини.

FramePacket доцільно розглядати як центральний об’єкт обміну даними між етапами. Він пов’язує кадр, часову мітку, телеметрію, ознаки, результат детекції, прогноз керування та статус контролю. Така структура зменшує ризик втрати контексту: кожен результат можна повернути до конкретного кадру й перевірити, які умови вплинули на рішення.

Важливим елементом конвеєра є обробка помилок. Якщо кадр не прочитано або телеметрія відсутня, система не повинна створювати фіктивний коректний результат. Краще сформувати неповний FramePacket, записати попередження та пропустити сценарій або перевести SafetyGuard у проблемний стан. Це демонструє інженерно відповідальну поведінку навіть у навчальному прототипі.

Діаграма активності показує логіку проходження даних через основні етапи, але її потрібно читати разом із діаграмою послідовності. Активність описує алгоритм, а послідовність - взаємодію об’єктів. Разом вони пояснюють, коли завантажуються параметри, коли виконується синхронізація, де з’являється прогноз і в який момент SafetyGuard може заблокувати результат.

Звітування є завершальною частиною конвеєра, а не другорядним додатком. Якщо результати не збережені, їх неможливо перевірити, порівняти або відтворити. Тому прототип має формувати щонайменше журнал подій, таблицю прогнозів, таблицю детекцій і підсумкові метрики. Ці файли виконують роль доказової бази під час аналізу.

У конвеєрі важливо розрізняти сирі дані, проміжні дані та результати. Сирі дані бажано не змінювати, щоб можна було повторити обробку. Проміжні дані можуть зберігатися вибірково для налагодження. Результати повинні зберігатися завжди, оскільки саме вони підтверджують виконання експерименту. Такий поділ допомагає організувати каталоги data, temp і reports.

Для кожного запуску доцільно формувати унікальний ідентифікатор. Він може включати дату, час, назву конфігурації або короткий номер сценарію. Це дозволяє не перезаписувати попередні результати й порівнювати експерименти між собою. Навіть у невеликій дипломній роботі така практика демонструє зрілість підходу до експериментування.

Конвеєр даних повинен бути описаний так, щоб його можна було реалізувати різними способами. Наприклад, відео може читатися з файлу, з камери або з симулятора, але для наступних компонентів результат має виглядати однаково. Це досягається через уніфікований формат пакета даних і чіткі правила обробки помилок.

Конвеєр даних визначає послідовність перетворень від вхідних файлів до звіту експерименту. Для ML-компонентів важливо, щоб обробка train/validation/test не змішувала послідовні кадри з одного сценарію, а аугментація застосовувалася лише до навчальної підмножини. Інакше оцінювання буде завищеним через витік даних.

На рисунку 2.3 наведено діаграму активності, що описує типовий запуск експерименту. Вона показує розгалуження для коректних і некоректних даних, паралельне виконання детекції та формування ознак для моделі керування, а також перевірку команди в межах ODD. Така діаграма корисна для розробника, оскільки перетворює текстовий опис у контрольований алгоритм.

Окремою вимогою є журналювання. Якщо кадр пошкоджено, телеметрія відсутня або confidence критичного об’єкта нижчий за поріг, система повинна не приховувати ситуацію, а явно позначати її в журналі. Це дозволяє відрізняти помилки алгоритму від некоректних вхідних даних і робити обережні висновки.

Рисунок 2.3 – Діаграма активності конвеєра даних, побудована в нотації PlantUML

Таблиця 2.4 – Структура пакета даних FramePacket

Поле	Тип	Призначення	Перевірка
frame_id	int/str	ідентифікатор кадру	унікальність у межах запуску
timestamp	float/datetime	часова мітка кадру	наявність і монотонність
image	array	матриця зображення	непорожній розмір і коректні канали
speed	float	швидкість або її імітація	допустимий діапазон
steering_target	float	еталонний кут керма для навчання	наявність для train/test
status	enum	стан пакета даних	OK / INCOMPLETE / ERROR
metadata	dict	джерело, сценарій, ODD-теги	наявність ключових полів

Сценарій виконання експерименту доцільно описувати не лише активністю, а й послідовністю повідомлень між модулями. Рисунок 2.4 показує, як ExperimentRunner координує завантаження пакета даних, детекцію об’єктів, прогноз керування, перевірку SafetyGuard і запис метрик. Такий сценарій підтверджує, що система має не випадковий набір функцій, а керований порядок взаємодії компонентів.

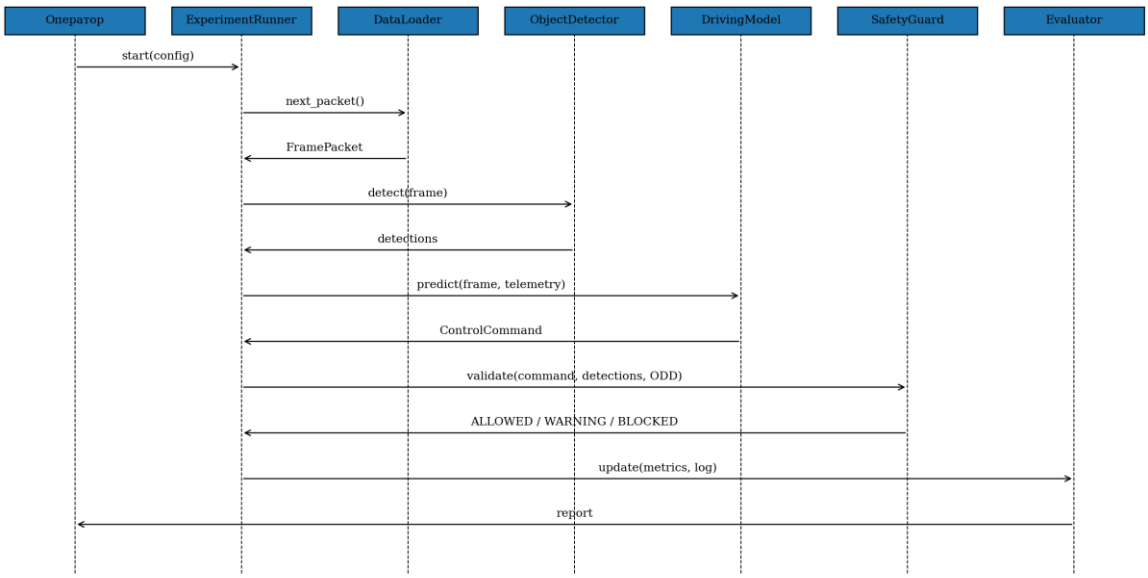


Рисунок 2.4 – Діаграма послідовності сценарного запуску експерименту, побудована в нотації PlantUML

Таблиця 2.5 – Трасування вимог до компонентів і артефактів

Вимога	Компоненти	Перевірка	Артефакт
FR-1, FR-2	VideoSource, TelemetrySource, DataLoader	завантаження коректних і некоректних файлів	лог імпорту
FR-3	Preprocessor	нормалізація кадру та синхронізація timestamp	збережений FramePacket
FR-4	ObjectDetector	детекція об’єктів із порогом confidence	кадр із рамками
FR-5	DrivingModel	прогноз кута керма	CSV із prediction/target
FR-6	SafetyGuard	перевірка меж і ODD	статус ALLOWED/WARNING/BLOCKED
FR-7, NFR-1	Evaluator, ReportBuilder	повторний запуск із конфігурацією	метрики, графіки, звіт

2.4 Тестування, сценарне оцінювання, контроль обмежень

Тестування в роботі групується за рівнями, щоб уникнути хаотичного переліку перевірок. На нижньому рівні розташовані модульні тести окремих функцій і класів. Далі йдуть інтеграційні перевірки передавання FramePacket між компонентами. Вище розташовані сценарні тести, у яких перевіряється поведінка системи в умовах нормального запуску, помилкових даних, низького confidence або виходу за межі ODD.

Для ML-систем особливо важливі негативні сценарії. Звичайний тест «модель запустилася» майже нічого не говорить про надійність. Набагато ціннішими є перевірки, у яких частина даних відсутня, кадр має погану якість, команда виходить за допустимий діапазон, а система має сформувати зрозуміле попередження. Такі сценарії демонструють, що прототип не лише обчислює результат, а й контролює межі його використання.

Метрики потрібно інтерпретувати разом, а не окремо. Висока швидкість обробки кадрів не компенсує погану якість детекції, а низька середня помилка керма не гарантує відсутності рідкісних великих відхилень. Саме тому в роботі

запропоновано поєднувати метрики детекції, керування, продуктивності, кількості попереджень і відтворюваності запуску.

Станова модель SafetyGuard потрібна для формалізації реакції на проблеми. Стани READY, RUNNING, WARNING, BLOCKED і FINISHED відображають не фізичний стан автомобіля, а логіку програмного контролю. Якщо система переходить у WARNING, результат може бути зафіксований як сумнівний; якщо переходить у BLOCKED, команда не повинна вважатися допустимою. Це робить оцінювання більш прозорим.

Проміжним підсумком розділу є те, що архітектура, конвеєр і тестування утворюють єдину систему. Вимога породжує компонент, компонент породжує сценарій перевірки, сценарій породжує метрику, а метрика впливає на висновок. Саме такий зв’язок дозволяє вважати роботу інженерною, а не лише оглядовою.

Трасованість тестів до вимог дозволяє швидко виявити прогалини. Якщо певна вимога не має жодного тестового сценарію, її виконання фактично не підтверджене. Якщо тест не пов’язаний із жодною вимогою, він може бути зайвим або погано сформульованим. Тому матриця трасування є корисним інструментом контролю повноти роботи.

Сценарне оцінювання повинно містити як типові, так і проблемні випадки. Типові випадки демонструють основний шлях роботи системи, а проблемні - її поведінку в умовах невизначеності. Для автономного керування саме проблемні випадки є найбільш важливими, бо вони виявляють слабкі місця моделей, конфігурації та контролю допустимості.

Важливо також розмежувати дефект програми й обмеження моделі. Якщо система аварійно завершується через відсутній файл, це дефект програмної реалізації. Якщо модель не розпізнає об’єкт у складному кадрі, це може бути обмеженням навчальних даних або архітектури. У звіті ці випадки мають аналізуватися по-різному.

Підсумковий висновок до розділу повинен показувати, що проєктна частина завершилася не лише набором діаграм, а цілісною моделлю майбутньої реалізації. Вимоги, компоненти, конвеєр, стани та тестові сценарії утворюють

структуру, за якою можна перевірити практичний результат і пояснити його обмеження.

Таким чином, тестування в роботі виконує не допоміжну, а системоутворювальну роль. Воно з'єднує вимоги, архітектуру та реалізацію, а також показує, що автор розуміє різницю між успішним запуском програми й перевіреним інженерним результатом. Для автономного керування така різниця є принциповою.

Тестування автономного керування не може зводитися до перевірки запуску коду. Потрібно перевіряти обробку даних, якість детекції, адекватність виходу моделі, повторюваність експериментів і поведінку системи за межами ODD. Тому тестова стратегія поєднує модульні, інтеграційні, сценарні та метаморфні перевірки.

Для середовищ моделювання важливими є CARLA, ASAM OpenDRIVE та ASAM OpenSCENARIO. CARLA надає відкриту платформу для симуляції міського руху [15], OpenDRIVE описує дорожні мережі [16], а OpenSCENARIO описує динамічну поведінку учасників і сценарії тестування [17]. Такі стандарти дозволяють переходити від окремого відеофрагмента до відтворюваних сценаріїв.

Дослідження DeepTest і DeepRoad показують, що для нейромережевих систем автономного водіння доцільно використовувати не лише звичайні unit-тести, а й трансформації вхідних даних, метаморфне тестування, моделювання погодних змін і перевірку стійкості до змінених умов [18; 19]. Сучасні огляди foundation models також підкреслюють роль генерації та аналізу складних сценаріїв для валідації автономних систем [20; 21].

Станова діаграма на рисунку 2.5 моделює поведінку SafetyGuard. Вона показує, що система може перебувати в режимах READY, RUNNING, WARNING, BLOCKED і FINISHED. Така модель корисна для реалізації тестів: кожен перехід можна перевірити окремим сценарієм, наприклад низькою впевненістю детектора, відсутньою телеметрією або командою за межами допустимого діапазону.

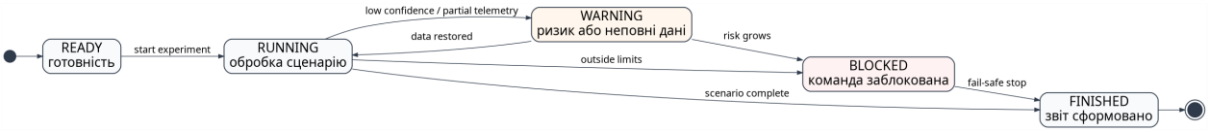


Рисунок 2.5 – Станова діаграма контролю допустимості керувальної дії, побудована в нотації PlantUML

Таблиця 2.6 – Приклади тестових сценаріїв

Код	Сценарій	Що перевіряється	Очікуваний результат
TC-1	коректний відеофайл	завантаження кадрів	усі кадри прочитано
TC-2	пошкоджений кадр	стійкість до помилки	кадр пропущено, журнал створено
TC-3	відсутня телеметрія	робота з неповними даними	команда заблокована або позначена
TC-4	різке засвітлення	робастність camera-only моделі	зростання помилки зафіксовано
TC-5	поява пішохода	детекція критичного об’єкта	подія записана
TC-6	низький confidence	перехід у WARNING	команда не приймається без застереження
TC-7	вихід за ODD	SafetyGuard	команда має статус BLOCKED
TC-8	повторний запуск	відтворюваність	метрики збігаються в межах допустимого відхилення

Таблиця 2.7 – Метрики оцінювання прототипу

Група	Метрика	Що показує	Обмеження інтерпретації
Детекція	confidence, кількість виявлень, пропуски критичних об’єктів	якість модуля сприйняття	confidence не дорівнює істинній імовірності без калібрування
Керування	MAE/RMSE кута керма	похибку прогнозу відносно мітки	open-loop метрика не враховує накопичення помилок
Продуктивність	час обробки кадру, FPS	ресурсну придатність	залежить від апаратного середовища
Надійність	кількість WARNING/BLOCKED	частоту проблемних ситуацій	потрібна репрезентативність сценаріїв
Відтворюваність	збіг результатів повторних запусків	стабільність експерименту	може порушуватися через випадковість ML-бібліотек

Приймальне тестування має фіксувати не лише очікуваний успішний результат, а й поведінку системи за відхилень. Для автономного керування це принципово: більшість ризиків виникає не тоді, коли дані ідеальні, а тоді, коли сенсор частково втрачає якість, кадр не відповідає очікуваному формату або телеметрія запізнюється.

Сценарії бажано групувати за джерелом ризику: вхідні дані, сприйняття, керувальна дія, ODD, продуктивність і відтворюваність. Така класифікація допомагає уникнути ситуації, коли всі тести перевіряють лише запуск програми, але не перевіряють її поведінку в проблемних умовах.

У навчальному прототипі не обов’язково реалізовувати повний промисловий набір сценаріїв, однак потрібно показати принцип їхньої побудови. Для кожного сценарію доцільно задавати вхідні дані, параметри ODD, очікуваний статус SafetyGuard, очікуваний артефакт звіту та критерій приймання.

Матриця нижче показує, як вимоги, сценарії та артефакти можуть бути пов’язані між собою. Вона корисна для захисту, оскільки демонструє, що тестування не є додатковим описом після реалізації, а прямо пов’язане з архітектурою та вимогами.

Таблиця 2.8 – Матриця сценарного оцінювання прототипу

Група сценаріїв	Приклад ситуації	Задіяні компоненти	Очікуваний артефакт
Вхідні дані	відео має некоректний кодек або пошкоджений кадр	DataLoader, Logger	помилка в журналі без аварійного завершення
Синхронізація	timestamp кадру не має відповідної телеметрії	DataLoader, Preprocessor, SafetyGuard	FramePacket зі статусом INCOMPLETE
Якість зображення	кадр затемнений або засвічений	Preprocessor, ObjectDetector, Evaluator	зниження confidence та запис попередження
Детекція	пішохід частково перекритий автомобілем	ObjectDetector, SafetyGuard	WARNING або збережений кадр для аналізу
Керування	модель прогнозує різкий кут керма	DrivingModel, SafetyGuard	BLOCKED через перевищення меж

Продовження таблиці 2.8

ODD	сценарій позначено як нічний або погана погода	ScenarioConfig, SafetyGuard	попередження про вихід за ODD
Продуктивність	довга послідовність кадрів	ExperimentRunner, Evaluator	таблиця часу обробки кадрів
Відтворюваність	повторний запуск із тією самою конфігурацією	Config, Evaluator	збіг метрик або пояснене відхилення
Звітність	завершення експерименту	ReportBuilder	CSV, графіки, короткий текстовий звіт
Негативний шлях	відсутній файл ваг моделі	ObjectDetector/DrivingModel, Logger	зрозуміле повідомлення про помилку

Окремо потрібно враховувати open-loop та closed-loop оцінювання. У open-loop режимі система прогнозує дію для вже записаного кадру, а результат порівнюється з міткою. У closed-loop режимі прогнозована дія впливає на стан середовища, тому помилки можуть накопичуватися. Для кваліфікаційної роботи достатньо описати open-loop реалізацію та обґрунтувати, чому closed-loop є наступним етапом.

Ще одним важливим питанням є калібрування порогів. Значення confidence 0,5 або 0,8 не є універсальним критерієм безпеки. Воно залежить від моделі, набору даних, класів об’єктів і вартості помилок. Тому в прототипі поріг має бути параметром конфігурації, а не «зашитою» константою без пояснення.

Для оцінювання моделі керування доцільно використовувати не одну метрику, а групу показників. MAE кута керма показує середню похибку, RMSE сильніше штрафує великі відхилення, кількість різких команд показує потенційні проблеми стабільності, а кількість блокувань SafetyGuard показує, як часто модель виходить за допустимі межі.

Після виконання сценаріїв результати слід інтерпретувати обережно. Якщо модель працює на демонстраційному наборі, це означає лише підтвердження принципу, а не промислову придатність. Коректний висновок має містити умови

експерименту, тип даних, обмеження ODD і перелік ситуацій, які не перевірялися.

У тексті пояснювальної записки подаються готові UML-діаграми, а не програмний код їх побудови. Такий підхід робить розділ читабельнішим для рецензента й комісії: увага зосереджується на вимогах, компонентах, потоках даних і станах контролю, а не на синтаксисі PlantUML. За потреби вихідні файли діаграм можуть зберігатися серед матеріалів проєкту, однак у дипломі достатньо навести самі графічні моделі та їх пояснення.

Діаграми виконують роль інженерних доказів: вони показують межі системи, розподіл відповідальності між компонентами, сценарій запуску експерименту й реакцію SafetyGuard на проблемні стани. Саме графічне подання дозволяє швидко зіставити вимоги, архітектуру та подальшу реалізацію.

Таблиця 2.9 – Призначення UML-діаграм у розділі 2

Модель	Що підтверджує	Зв’язок із вимогами
Діаграма прецедентів	межі системи та ролі користувачів	FR-1–FR-8
Компонентна діаграма	розподіл відповідальності між модулями	NFR-3, NFR-5
Діаграма активності	послідовність обробки даних і гілки помилок	FR-3, FR-6, FR-7
Діаграма послідовності	взаємодію компонентів під час запуску сценарію	FR-4–FR-7
Станова діаграма	логіку READY/RUNNING/WARNING/BLOCKED/FINISHED	FR-6, TC-6, TC-7

У другому розділі сформовано функціональні та нефункціональні вимоги до програмного прототипу безпілотного керування. Вимоги пов’язано з компонентами, тестами та артефактами звітування, що забезпечує трасованість і контрольованість практичної частини.

Побудовано UML-діаграми в нотації PlantUML: діаграму прецедентів, компонентну діаграму, діаграму активності, діаграму послідовності та станову діаграму SafetyGuard. Вони підтверджують інженерну складову роботи, оскільки

описують не лише нейромережеву модель, а й структуру, сценарії взаємодії та поведінку системи за проблемних умов.

Запропонована архітектура дозволяє розглядати YOLOv8, CNN-модель керування та інші ML-елементи як замінні компоненти з визначеними входами й виходами. Тестова стратегія охоплює успішні та негативні сценарії, що є необхідною умовою коректного оцінювання автономного керування.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ТА КРИТИЧНЕ ОЦІНЮВАННЯ

3.1 Засоби реалізації та структура програмного проєкту

Реалізаційна частина має відповідати архітектурі, а не суперечити їй. Якщо в розділі 2 виділено окремі компоненти, то в програмному проєкті повинні існувати відповідні модулі або класи. Навіть якщо їхня реалізація демонстраційна, важливо зберегти логічний поділ. Це полегшує пояснення роботи, дозволяє додавати тести та створює основу для подальшого розвитку прототипу.

Python-екосистема зручна для такої роботи завдяки великій кількості бібліотек і низькому порогу інтеграції. Проте вона має й обмеження: продуктивність може залежати від апаратного середовища, версій бібліотек і доступності GPU. Тому в роботі доцільно не лише назвати бібліотеки, а й описати спосіб фіксації середовища через requirements, README та конфігураційні файли.

Структура проєкту повинна дозволяти запускати короткий демонстраційний сценарій без ручного редагування коду. Параметри мають задаватися конфігурацією: шлях до відео, шлях до телеметрії, поріг confidence, допустимий діапазон команди, режим збереження результатів. Таке рішення робить прототип гнучкішим і ближчим до реальної інженерної практики.

Журналювання є важливим елементом реалізації. У журналі варто зберігати старт і завершення запуску, кількість оброблених кадрів, кількість пропусків, помилки читання, попередження SafetyGuard і підсумкові метрики. Це допомагає не лише під час відлагодження, а й під час захисту, коли потрібно показати, що результат отримано контрольовано.

Окремо слід враховувати відокремлення демонстраційних даних від програмного коду. Дані можуть мати інший ліцензійний статус, велику вагу або обмеження поширення. Тому в репозиторії доцільно зберігати короткий sample-набір або інструкцію, як підготувати дані, а великі датасети згадувати як зовнішні ресурси.

Для зручності супроводу бажано використовувати єдині правила іменування. Назви класів, файлів і компонентів на діаграмах повинні відповідати одне одному. Якщо в UML зазначено SafetyGuard, то в коді не варто називати його випадковим іншим терміном. Узгодженість назв зменшує когнітивне навантаження й полегшує перевірку роботи.

Реалізація має враховувати, що різні користувачі можуть запускати прототип у різних середовищах. Тому абсолютні шляхи, прив’язані до конкретного комп’ютера, слід замінювати відносними шляхами або параметрами конфігурації. Це просте правило значно підвищує відтворюваність і зменшує кількість технічних проблем під час демонстрації.

Організація файлів проєкту має відображати очікуваний спосіб роботи користувача. Людина, яка відкриває проєкт уперше, повинна швидко знайти інструкцію, конфігурацію, основний запуск, тести та результати. Якщо структура зрозуміла без додаткових пояснень, це свідчить про якісну підготовку практичного артефакту.

Практична реалізація прототипу орієнтована на Python-екосистему, оскільки вона забезпечує доступ до бібліотек комп’ютерного зору, машинного навчання, обробки даних і побудови демонстраційних експериментів. Для нейромережевої частини доцільно використовувати TensorFlow/Keras [22; 23], для обробки зображень - OpenCV [24], а для організації запуску, журналювання та обробки файлів - стандартні засоби Python [25].

Структура проєкту повинна бути зрозумілою для повторного запуску. Мінімально вона має містити каталоги src, tests, configs, data, reports і docs. У src розміщуються програмні модулі, у tests - перевірки, у configs - параметри експерименту, у data - демонстраційні набори або посилання на них, у reports - результати запусків, у docs - інструкції та додаткові схеми.

Для забезпечення відтворюваності важливо зафіксувати версії бібліотек, параметри запуску, розміри вхідних зображень, пороги confidence, шлях до ваг моделі та формат звітних файлів. Якщо модель використовує попередньо навчені

ваги, їх походження має бути описане в README або окремому файлі license_notes.

Реалізація має будуватися так, щоб демонстраційний запуск не вимагав великого набору даних. Для захисту достатньо короткого сценарію, який показує проходження кадру через DataLoader, Preprocessor, ObjectDetector, DrivingModel, SafetyGuard і Evaluator. Повний експеримент може бути поданий як перспективний режим, якщо він потребує значних обчислювальних ресурсів.

Таблиця 3.1 – Рекомендована структура програмного проєкту

Каталог / файл	Призначення	Приклад вмісту
src/data_loader.py	завантаження відео й телеметрії	клас DataLoader, валідація файлів
src/preprocessor.py	нормалізація кадрів	resize, normalization, timestamp alignment
src/detector.py	детекція об’єктів	інтерфейс ObjectDetector, YOLOv8-реалізація
src/driving_model.py	прогноз керування	CNN-модель або завантаження ваг
src/safety_guard.py	перевірка меж і ODD	ALLOWED/WARNING/BLOCKED
src/evaluator.py	збір метрик	MAE, FPS, warnings, CSV
configs/experiment.yaml	параметри запуску	шляхи, пороги, ODD-теги
tests/	перевірки	unit/integration/scenario tests
reports/	результати	графіки, журнали, звіти

У межах пояснювальної записки код доцільно подавати не великими фрагментами, а через інтерфейси, псевдокод і зв’язок із вимогами. Такий підхід краще відповідає інженерній логіці роботи: читач бачить не випадкові рядки програми, а призначення модуля, його вхідні дані, вихідні дані та роль у загальному конвеєрі.

Нижче подано узагальнений псевдокод основних модулів. Він не замінює первинний код, але пояснює, як архітектурна модель другого розділу переходить у програмну реалізацію.

Лістинг 3.1 – Узагальнений псевдокод запуску експерименту

```
load config
initialize DataLoader, ObjectDetector, DrivingModel, SafetyGuard, Evaluator
while DataLoader.has_next():
```

```
    packet = DataLoader.next_packet()
    if packet.status != OK: log warning and continue
    frame = Preprocessor.normalize(packet.image)
    detections = ObjectDetector.detect(frame)
    command = DrivingModel.predict(frame, packet.telemetry)
    decision = SafetyGuard.validate(command, detections, packet.odd_state)
    Evaluator.update(packet.frame_id, detections, command, decision)
Evaluator.finalize_report()
```

3.2 Модулі розпізнавання об’єктів і керування

Модуль розпізнавання об’єктів має повертати результат у стандартизованому форматі. Для кожного об’єкта потрібно зберігати клас, координати рамки, confidence і номер кадру. Якщо модель не виявила об’єктів, це також є результатом, який треба зафіксувати. Відсутність детекцій не повинна руйнувати конвеєр, оскільки в реальних даних такі ситуації є звичайними.

YOLOv8 у роботі використовується як актуальний інструмент об’єктної детекції в екосистемі Ultralytics. На відміну від попередніх поколінь YOLO, його доцільно застосовувати через уніфікований Python API та попередньо навчені ваги, що спрощує інтеграцію з OpenCV, сценарним запуском і формуванням демонстраційних результатів [26]. Для дипломної роботи це важливо, оскільки увага зосереджується не лише на факті розпізнавання, а й на інженерному включенні детектора в контрольований конвеєр.

Модель керування, побудована на CNN, має іншу природу, ніж детектор. Вона не повертає клас об’єкта, а прогнозує числову дію або параметр руху. Тому для неї потрібні інші метрики та інша інтерпретація. Помилка в кілька градусів може бути незначною в одному контексті й небезпечною в іншому, якщо сценарій містить перешкоду або вузьку смугу.

Важливо, щоб модулі розпізнавання та керування не приймали остаточне рішення про безпечність. Їхнє завдання - сформувати прогноз. Остаточне оцінювання допустимості має виконувати SafetyGuard, який враховує межі

команди, confidence, ODD і наявність критичних попереджень. Такий поділ дозволяє зберегти контроль навіть тоді, коли ML-модель повертає впевнений, але потенційно проблемний результат.

У реалізації бажано передбачити можливість режиму mock або stub для окремих моделей. Якщо ваги недоступні або запуск моделі потребує надто багато ресурсів, система може використовувати спрощений прогноз для перевірки конвеєра. Це не замінює реальну модель, але дозволяє тестувати архітектуру, журналювання, SafetyGuard і звітність.

Під час роботи з відео важливо не лише отримати результат моделі, а й зберегти зв’язок із конкретним кадром. Номер кадру, timestamp і шлях до збереженого зображення дозволяють перевірити, чому модель повернула саме такий прогноз. Без цього аналіз перетворюється на перегляд абстрактних чисел, відірваних від дорожньої сцени.

Для модуля керування корисно зберігати не тільки прогноз, а й цільове значення, якщо воно доступне. Це дозволяє обчислити помилку, побудувати графік prediction/target і виявити ділянки, де модель поводиться нестабільно. Таке подання краще демонструє результат, ніж одна усереднена метрика.

У подальшому такий формат результатів дозволить оновлювати демонстраційні моделі без зміни всього конвеєра. Наприклад, замість поточної конфігурації YOLOv8 можна використати легшу модель для слабкого обладнання, точнішу модель для офлайн-аналізу або спеціалізований детектор дорожніх об’єктів. Якщо інтерфейси збережено, інші модулі продовжать працювати з тим самим типом даних: клас, рамка, confidence і час обробки.

Модуль розпізнавання об’єктів приймає кадр дорожньої сцени та повертає список виявлень: клас об’єкта, координати рамки, confidence і час обробки. У практичній частині використовується YOLOv8 як актуальна модель об’єктної детекції з підтримкою попередньо навчених ваг і зручного Python API [26]. У контексті роботи це дає змогу не навчати детектор з нуля, а зосередитися на інтеграції моделі в програмний конвеєр, перевірці результатів і демонстрації зв’язку між сприйняттям та керуванням.

Критичне використання YOLOv8 полягає в тому, що модель не подається як достатня умова безпечного автономного водіння. Вона виконує роль компонента сприйняття, результати якого мають перевірятися порогами confidence, обмеженнями ODD і модулем SafetyGuard. Такий підхід дозволяє використати актуальну модель, але не завищувати рівень готовності прототипу до реального дорожнього застосування.

Модель керування реалізує end-to-end принцип: на основі кадру або підготовлених ознак прогнозується керувальна дія, наприклад кут керма. Перевага такого підходу полягає в простоті демонстрації зв’язку між зображенням і дією. Недолік полягає в тому, що модель може навчитися непричинних залежностей, бути чутливою до доменного зсуву та погано пояснювати власні рішення [4; 13; 14].

Для зменшення ризику некоректної інтерпретації результатів вихід моделі керування не передається безпосередньо у виконавчий механізм. Він проходить через SafetyGuard, який перевіряє діапазон команди, статус вхідних даних, confidence критичних об’єктів і відповідність ODD. У навчальному прототипі SafetyGuard не гарантує функціональну безпеку, але дисциплінує архітектуру й демонструє правильний напрям розвитку.

Дані для навчання й оцінювання можуть походити з відкритих наборів або локальних демонстраційних фрагментів. KITTI, nuScenes і Waymo Open Dataset є важливими прикладами наборів даних для автономного водіння, які демонструють різні масштаби, сенсорні набори та задачі сприйняття [27; 28; 29]. У межах кваліфікаційної роботи вони використовуються як джерела порівняння й орієнтири для майбутньої експериментальної бази.

Таблиця 3.2 – Роль ML-компонентів у прототипі

Компонент	Призначення	Переваги	Обмеження
YOLOv8	детекція об’єктів на кадрі	доступність, зрозуміла структура, приклади реалізації	не є сучасним state-of-the-art, потребує калібрування порогів
CNN-керування	прогноз кута керма	наочний end-to-end принцип	низька пояснюваність, залежність від даних

Продовження таблиці 3.2

Preprocessor	підготовка кадрів і ознак	повторюваність формату	помилка на цьому етапі впливає на всі моделі
SafetyGuard	обмеження команд	зменшує ризик неконтрольованого виходу	не замінює промисловий safety case
Evaluator	аналіз результатів	формує метрики й докази запуску	потребує якісних еталонних даних

Лістинг 3.2 – Псевдокод модуля SafetyGuard

```
def validate(command, detections, packet, odd_state):
    if packet.status == "INCOMPLETE": return "BLOCKED", "incomplete input"
    if abs(command.steering_angle) > MAX_STEERING: return "BLOCKED", "out of range"
    if odd_state not in ALLOWED_ODD: return "WARNING", "outside ODD"
    if critical_object_confidence(detections) < MIN_CONFIDENCE:
        return "WARNING", "low perception confidence"
    return "ALLOWED", "valid command"
```

З погляду інженерії програмного забезпечення важливо, що SafetyGuard формує не лише бінарну відповідь, а й причину рішення. Це дозволяє аналізувати проблемні сценарії: чи команда заблокована через неповні дані, вихід за межі керма, низьку впевненість детектора або порушення ODD.

У звіті експерименту для кожного кадру бажано зберігати frame_id, статус пакета, кількість виявлених об’єктів, максимальний confidence, прогноз кута керма, еталонну мітку за наявності, статус SafetyGuard і час обробки. Такий журнал є основою для подальшого аналізу помилок.

3.3 Оцінювання результатів, ризики та напрями подальшого розвитку

Оцінювання результатів має бути критичним і прив’язаним до умов експерименту. Необхідно вказувати, які дані використано, скільки кадрів оброблено, які параметри конфігурації задано, які метрики обчислено та які

обмеження залишаються. Такий опис дозволяє уникнути завищених висновків і показує, що результат має конкретний контекст застосування.

Ризики прототипу можна поділити на технічні, алгоритмічні та організаційні. Технічні ризики стосуються помилок середовища, версій бібліотек, відсутності GPU або несумісних форматів файлів. Алгоритмічні ризики пов’язані з якістю детекції, перенавчанням, зміщенням даних і нестабільністю прогнозів. Організаційні ризики виникають тоді, коли демонстраційні результати помилково трактують як промислову готовність.

Для кожного ризику бажано вказати спосіб пом’якшення. Помилки середовища зменшуються через requirements і README, ризики даних - через валідацію входів, ризики моделі - через метрики та сценарії, ризики інтерпретації - через чітке формулювання ODD і меж роботи. Такий підхід перетворює перелік ризиків на інженерний інструмент, а не на формальну таблицю.

Дорожня карта розвитку має бути реалістичною. Першочерговими є не найскладніші нейромережі, а стабільність конвеєра, автоматизовані тести, покращене журналювання й відтворюваність. Лише після цього доцільно переходити до симуляторів, OpenSCENARIO, OpenDRIVE, BEV-представлень або sensor fusion. Така послідовність відповідає поступовому нарощуванню інженерної зрілості.

Окремим критерієм зрілості є здатність пояснити помилку. Якщо система видала невдалий прогноз, потрібно мати змогу переглянути кадр, телеметрію, confidence, вхідні параметри й статус SafetyGuard. Без такої інформації складно зрозуміти, чи проблема виникла через дані, модель, конфігурацію або інтерпретацію результату.

Результати експерименту бажано подавати не лише чисельно, а й пояснювально. Якщо метрика погіршилася, потрібно вказати можливі причини: складніші кадри, низьке освітлення, відсутність телеметрії, невідповідність ODD або обмеження навчальної вибірки. Такий аналіз показує здатність не тільки запускати код, а й інтерпретувати його поведінку.

Критичне оцінювання має містити баланс між досягненнями й обмеженнями. Перевагою роботи є структурований прототип, модульність, UML-моделі, сценарне тестування та звітність. Обмеженнями залишаються навчальний масштаб, спрощений набір даних, відсутність сертифікації, повного closed-loop середовища та промислового safety case. Саме така збалансованість робить висновки переконливими.

Оцінювання ризиків варто пов’язувати з конкретними рішеннями роботи. Якщо ризиком є недостатня якість кадрів, то відповіддю має бути перевірка якості входу. Якщо ризиком є різка команда керування, відповіддю є SafetyGuard. Якщо ризиком є невідтворюваність, відповіддю є конфігурація, журнал і збереження run_id.

Оцінювання результатів має бути обережним і прив’язаним до меж ODD. Якщо прототип перевірено на коротких відеофрагментах або демонстраційному наборі даних, висновки мають стосуватися саме цих умов. Неприпустимо переносити отримані показники на всі дорожні ситуації, погодні умови або швидкісні режими.

Open-loop оцінювання порівнює прогноз моделі з записаною міткою, наприклад із фактичним кутом керма. Така перевірка корисна для навчального прототипу, але не показує накопичення помилок у замкненому контурі. Closed-loop оцінювання в CARLA або іншому симуляторі краще відображає поведінку системи, оскільки вихід моделі впливає на подальший стан середовища [15].

Основні ризики прототипу пов’язані з camera-only сприйняттям, обмеженістю даних, недостатньою різноманітністю сценаріїв, відсутністю фізичного виконавчого контуру, складністю інтерпретації CNN та залежністю результатів від параметрів запуску. Кожен із цих ризиків повинен бути явно зафіксований у висновках і не приховуватися загальними формулюваннями про успішність моделі.

Сильними сторонами прототипу є модульна структура, наявність вимог, UML-моделей, тестових сценаріїв, журналювання і критичного подання

нейромережевих компонентів. Навіть якщо окремі модулі є демонстраційними, їх поєднання в архітектурний конвеєр показує інженерне розуміння задачі.

Подальший розвиток доцільно спрямувати на інтеграцію з CARLA, опис сценаріїв у форматах ASAM OpenSCENARIO та OpenDRIVE, розширення сценарних наборів, додавання BEV-представлення, автоматизований запуск тестів і формування звіту в CI-середовищі. Для детекції доцільно зберегти інтерфейс ObjectDetector, щоб у майбутньому порівнювати різні конфігурації YOLOv8 або новіші детектори без перебудови всієї архітектури.

Окремим перспективним напрямом є використання foundation models для генерації складних сценаріїв, пояснення дорожніх ситуацій і синтезу рідкісних випадків. Проте такі підходи потребують обережності: вони можуть збільшити різноманітність тестів, але водночас додають питання достовірності, відтворюваності та контролю галюцинацій.

Таблиця 3.3 – Ризики прототипу та способи їх пом’якшення

Ризик	Прояв	Наслідок	Пом’якшення
Camera-only підхід	залежність від освітлення і погоди	помилки сприйняття	додати радар/лідар або BEV fusion
Малий набір даних	перенавчання на знайомих сценах	завищені метрики	розширити сценарії та розділення даних
Низька інтерпретованість	важко пояснити прогноз керма	складність довіри до рішення	логувати ознаки, помилки, статус SafetyGuard
Відсутність closed-loop	прогноз не впливає на середовище	не видно накопичення помилок	інтегрувати CARLA
Некоректна телеметрія	відсутні або несинхронні мітки	помилкове навчання	валідація timestamp і статус INCOMPLETE
Застарілий детектор	можлива нестабільність детекції у складних кадрах	пропуски об’єктів	інтерфейс для заміни детектора

Таблиця 3.4 – Дорожня карта подальшого розвитку

Етап	Зміст робіт	Очікуваний результат
Короткостроковий	оформити README, requirements, конфігурації й демонстраційний сценарій	відтворюваний запуск прототипу
Короткостроковий	додати unit/integration тести для DataLoader, Preprocessor, SafetyGuard	перевірка базових компонентів
Середньостроковий	порівняти різні конфігурації YOLOv8 або новіші детектори через той самий інтерфейс	порівняння якості сприйняття
Середньостроковий	підключити CARLA для closed-loop сценаріїв	оцінка накопичення помилок
Середньостроковий	описати сценарії в OpenSCENARIO/OpenDRIVE	стандартизоване сценарне тестування
Перспективний	додати BEV або sensor fusion	краще просторове представлення сцени
Перспективний	дослідити foundation models для генерації сценаріїв	розширення рідкісних тестових випадків

Для захисту доцільно підготувати коротку демонстрацію: показ структури проєкту, відкриття конфігурації, запуск обробки кількох кадрів, перегляд виявлених об’єктів, прогнозу керма, статусу SafetyGuard, журналу та таблиці метрик. Такий сценарій чітко показує результат роботи й не потребує довгого навчання моделі під час виступу.

3.4 Демонстраційні артефакти, відтворюваність

Демонстраційний пакет має бути організований так, щоб сторонній перевіряльник міг відтворити основний сценарій без неочевидних дій. Для цього потрібно надати інструкцію встановлення, приклад команди запуску, опис вхідних файлів, очікувані вихідні артефакти та приклад результату. Якщо певний компонент не запускається без зовнішніх ваг або великого датасету, це треба пояснити окремо.

Передзахисна перевірка повинна включати не лише читання тексту, а й запуск демонстраційного сценарію. Варто перевірити, чи створюються всі файли reports, чи не містять шляхи локальних абсолютних адрес, чи зрозумілі

повідомлення про помилки, чи відповідають назви модулів тим, що описані в дипломі. Це зменшує ризик технічних збоїв під час захисту.

Відтворюваність також залежить від оформлення результатів. Якщо в роботі наведено графік або таблицю метрик, має бути зрозуміло, з якого запуску вони отримані. Доброю практикою є збереження run_id, дати запуску, імені конфігурації та версії моделі. Навіть у навчальному прототипі це дисциплінує експеримент і робить результати перевірюваними.

Фінальні висновки розділу мають бути узгоджені з фактичною реалізацією. Якщо реалізовано демонстраційний режим, не слід стверджувати про промислову придатність. Якщо описано лише open-loop оцінювання, потрібно вказати, що closed-loop перевірка є наступним етапом. Така точність формулювань підвищує наукову й інженерну якість роботи.

Підсумково практичний результат роботи слід представляти як структурований програмний прототип з архітектурою, сценаріями, журналюванням, контролем допустимості та звітністю. Саме цей комплекс артефактів підтверджує виконання роботи краще, ніж ізольований запуск окремої нейромережевої моделі.

Пакет демонстраційних артефактів має бути узгоджений із текстом пояснювальної записки. Якщо в розділі описано певний модуль або метрику, у файлах проєкту повинен бути відповідний приклад. Це дозволяє комісії швидко зіставити теоретичний опис із практичною реалізацією та зменшує ризик формального сприйняття роботи.

Відтворюваність фінального результату також залежить від простоти демонстраційного сценарію. Краще показати короткий, але стабільний запуск із підготовленими даними, ніж намагатися під час захисту виконати довге навчання моделі. Основна мета демонстрації - підтвердити архітектуру, конвеєр, контроль обмежень і формування звітних файлів.

Фінальний пакет матеріалів має підтверджувати, що робота може бути перевірена не лише автором. Саме тому важливі README, приклади запуску,

збережені результати, таблиці метрик і UML-діаграми. Вони роблять диплом не разовим текстом, а відтворюваним навчальним інженерним проєктом.

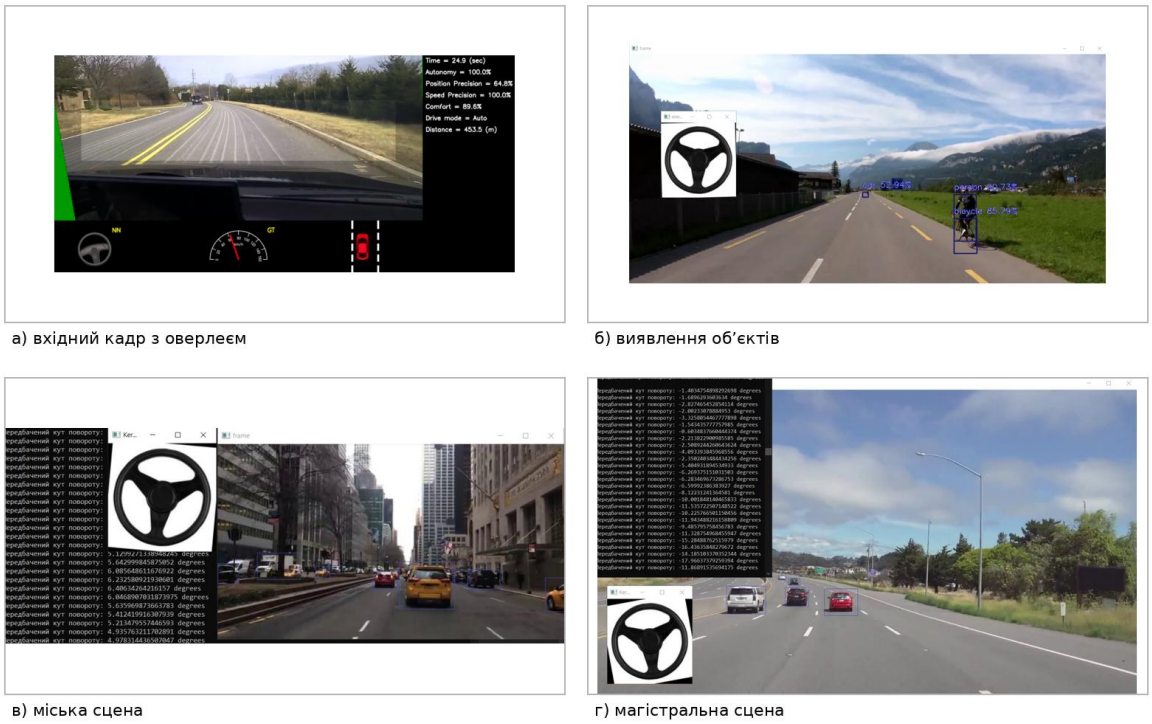
Пояснювальна записка повинна супроводжуватися пакетом артефактів, який дозволяє перевірити практичний результат. Для програмного прототипу автономного керування такими артефактами є вихідний код, інструкція запуску, конфігурації, короткий демонстраційний набір даних, журнали виконання, збережені кадри з детекцією, таблиці метрик і графіки.

Наявність артефактів важлива з двох причин. По-перше, вона підтверджує, що робота не є лише теоретичним оглядом. По-друге, вона полегшує захист: студент може показати, які модулі реалізовано, які параметри використовувалися, які дані подавалися на вхід і які результати було отримано.

Демонстраційний сценарій має бути коротким і контрольованим. Доцільно показати структуру проєкту, конфігураційний файл, запуск обробки кількох кадрів, результат детекції, прогноз керма, статус SafetyGuard, журнал і збережений звіт. Такий сценарій краще сприймається комісією, ніж довге навчання моделі під час захисту.

Якщо повний запуск потребує GPU або великого набору даних, можна підготувати демонстраційний режим із малим набором кадрів і попередньо збереженими вагами. При цьому потрібно прямо зазначити, що це демонстраційний режим, а не промислове оцінювання автономної системи.

Практична демонстрація прототипу повинна містити не лише опис очікуваних артефактів, а й приклади фактичного виведення результатів. У межах роботи демонстраційний запуск охоплює вхідний кадр дорожньої сцени, результат виявлення об’єктів, виведення службового оверлею керування та контроль навчальної динаміки моделі. Такі матеріали дозволяють перевірити, що програмний результат має видимий і відтворюваний вихід, а не обмежується теоретичним описом архітектури.



а) вхідний кадр з оверлеєм

б) виявлення об’єктів

в) міська сцена

г) магістральна сцена

Рисунок 3.1 – Демонстраційні результати роботи програмного прототипу на дорожніх сценах

На рисунку 3.1 показано кілька типових станів демонстраційного запуску: вхідний кадр із параметрами руху, результат детекції об’єктів, міську сцену та магістральну сцену з виведенням службової інформації. Наявність такого візуального результату важлива для захисту, оскільки дозволяє показати зв’язок між даними, моделлю сприйняття, прогнозом керування та журналом виконання.

а) журнал навчання у терміналі

б) графік втрат моделі

Рисунок 3.2 – Демонстраційні артефакти навчання та оцінювання моделі керування

Рисунок 3.2 ілюструє службові результати, які доповнюють візуальну демонстрацію: журнал навчального запуску та графік втрат. Вони потрібні для пояснення того, що робота моделі оцінюється не лише за одиничним кадром, а й за процесом навчання, стабільністю запуску та можливістю повторного аналізу результатів.

Наведені демонстраційні результати не слід трактувати як підтвердження готовності до реального руху. Їх призначення полягає в іншому: показати, що прототип має відтворюваний сценарій запуску, формує візуальний результат обробки кадру, дозволяє аналізувати поведінку моделі та створює матеріали для подальшого оцінювання. У поєднанні з таблицями артефактів і сценаріями тестування ці рисунки підтверджують практичну складову роботи.

Перед фінальним поданням роботи варто перевірити узгодженість між текстом, діаграмами, таблицями та фактичними файлами проєкту. Якщо в тексті заявлено журналювання, має бути приклад журналу. Якщо заявлено конфігурацію, має бути файл конфігурації. Якщо заявлено тестові сценарії, має бути таблиця або каталог сценаріїв.

Таблиця 3.5 – Рекомендований пакет артефактів для демонстрації

Артефакт	Зміст	Формат	Призначення
README	опис встановлення, запуску та параметрів	md/txt	відтворюваність
requirements	версії Python-бібліотек	txt	стабільність середовища
config	шляхи, пороги, режим запуску	yaml/json	керування експериментом
src	модулі DataLoader, Detector, Model, SafetyGuard	py	практична реалізація
tests	модульні та інтеграційні перевірки	py	підтвердження якості
models	ваги або посилання на ваги	h5/pt/onnx/link	повторний запуск
sample_data	короткий відеофрагмент і телеметрія	mp4/csv	демонстрація
reports	звіти запусків	csv/png/pdf	оцінювання

Продовження таблиці 3.5

logs	журнал помилок і подій	log/txt	аналіз поведінки
figures	скріншоти та графіки	png/jpg	ілюстрації для записки
scenarios	опис тестових сценаріїв	json/yaml	відтворюваність тестів
license_notes	відомості про сторонні компоненти	txt	академічна доброчесність

Таблиця 3.6 – Короткий сценарій демонстрації прототипу

Крок	Дія студента	Що бачить комісія	Очікуваний результат
1	показ структури проєкту	каталоги src, tests, reports	зрозуміла організація
2	відкриття конфігурації	шляхи до даних, пороги, ODD-теги	відтворювані параметри
3	запуск демонстраційного режиму	консоль або вікно програми	старт без помилок
4	обробка кадру	рамки об’єктів і confidence	видно роботу детектора
5	показ прогнозу керма	числове значення або індикатор	видно вихід моделі
6	імітація проблемного кадру	попередження в журналі	перевірка негативного сценарію
7	перегляд звіту	CSV/графік/лог	результати збережено
8	пояснення обмежень	ODD і ризики	коректна інтерпретація

Таблиця 3.7 – Передзахисна перевірка узгодженості тексту та артефактів

Пункт перевірки	Питання для самоконтролю	Ознака готовності
Тема	чи відповідає формулювання теми програмному результату?	тема не зводиться до огляду
Мета	чи досягається мета розділами роботи?	кожне завдання має результат
ODD	чи описано межі застосування?	є таблиця ODD і обмеження
Вимоги	чи є FR/NFR?	є таблиці вимог і критерії приймання
Архітектура	чи зрозумілі модулі?	є UML-діаграми та таблиця компонентів
Дані	чи описано вхідні поля?	є структура FramePacket

Продовження таблиці 3.7

Модель	чи пояснено вибір CNN/YOLOv8 і межі їх застосування?	є критичний аналіз меж застосування
Тести	чи є сценарії?	є таблиця тестів і матриця оцінювання
Метрики	чи не завищено висновки?	метрики трактуються обережно
Джерела	чи всі посилання потрібні?	список відповідає тексту
Формат	чи виконані базові вимоги?	Times New Roman, поля, інтервал
Демонстрація	чи є короткий сценарій показу?	є послідовність дій на 3–5 хвилин

Редакційно важливо використовувати однакові терміни. У роботі термін «безпілотне керування» використовується як назва теми, а технічний аналіз спирається на поняття програмного прототипу ADS-функції, ODD, сценарного тестування та SafetyGuard. Це зменшує ризик завищених тверджень про рівень автономності.

Також потрібно уникати категоричних висновків на кшталт «модель безпечна» або «система готова до використання». Коректними є формулювання: «прототип демонструє принцип», «модель перевірено в заданих умовах», «результати не поширюються за межі ODD», «система потребує closed-loop валідації». Такий стиль відповідає сучасній інженерній відповідальності.

Окремо слід перевірити, чи всі рисунки та таблиці використовуються в тексті. Якщо діаграма наведена, перед нею має бути пояснення, після неї - інтерпретація. Це особливо важливо для UML-діаграм: вони не повинні виглядати як декоративні вставки, а мають підтверджувати вимоги, компоненти, сценарії та логіку контролю.

Під час усного захисту бажано розділити демонстрацію на дві частини: інженерну та експериментальну. Інженерна частина пояснює вимоги, архітектуру, PlantUML-діаграми та обмеження ODD. Експериментальна частина показує запуск прототипу, результат детекції, прогноз керма, статус SafetyGuard і сформовані метрики.

Якщо комісія ставить питання про реальне використання системи в автомобілі, відповідь має бути обережною: робота демонструє програмну модель і навчальний конвеєр, але для реального транспорту потрібні сертифікація, апаратне резервування, safety case, кібербезпека, розширені сценарії та багаторівневе тестування.

Таблиця 3.8 – Узгодження демонстрації з інженерними артефактами

Питання на захисті	На що посилатися в роботі	Очікувана відповідь
Який практичний результат отримано?	структура проєкту, модулі, сценарій демонстрації	навчальний програмний прототип із модульною архітектурою
Чому це не готовий автопілот?	ODD, SafetyGuard, ризики, висновки	відсутні сертифікація, fallback і closed-loop докази
Де інженерія ПЗ?	FR/NFR, PlantUML, трасування, тести	робота описує вимоги, архітектуру, компоненти та перевірки
Чому використано YOLOv8?	YOLOv8 є актуальною моделлю об’єктної детекції в екосистемі Ultralytics, має зручний Python API, попередньо навчені ваги та дозволяє швидко отримати відтворювану демонстрацію результатів.	базова демонстраційна модель зі зрозумілими обмеженнями
Як перевіряється результат?	таблиці тестових сценаріїв і метрик	через модульні, інтеграційні та сценарні перевірки
Що буде далі?	дорожня карта розвитку	CARLA, OpenSCENARIO, OpenDRIVE, BEV, нові детектори

Таблиця 3.9 – Мінімальний набір результатів одного експериментального запуску

Файл або показник	Зміст	Навіщо потрібний
run_config.yaml	параметри запуску, пороги, ODD-теги	відтворення експерименту
frames_out/	кадри з нанесеними рамками детекції	візуальна перевірка сприйняття
predictions.csv	frame_id, prediction, target, error	оцінювання моделі керування
detections.csv	класи, bbox, confidence, час обробки	аналіз детектора
safety_log.csv	статус ALLOWED/WARNING/BLOCKED і причина	контроль допустимості команди

Продовження таблиці 3.9

metrics.json	MAE, RMSE, FPS, кількість попереджень	узагальнення результатів
experiment.log	помилки, попередження, службові повідомлення	діагностика запуску
report.pdf або report.md	короткий підсумок запуску	матеріал для захисту

Після формування пакета результатів необхідно перевірити, чи немає розриву між заявленими можливостями та фактичними файлами. Наприклад, якщо в тексті зазначено сценарне тестування, то має бути хоча б один опис сценарію або таблиця сценаріїв; якщо зазначено метрики, то має бути файл або таблиця з їхнім значенням.

Корисною є також коротка карта відповідності між артефактами й розділами пояснювальної записки. Вона допомагає швидко знайти, де саме в тексті пояснено той чи інший файл проєкту, і запобігає ситуації, коли практичні результати існують окремо від документа.

Для подальшого супроводу проєкту бажано зберігати всі графічні UML-схеми в каталозі docs/uml. Тоді будь-яка зміна архітектури або вимог може бути синхронізована з діаграмами без ручного перемальовування рисунків у пояснювальній записці.

Таблиця 3.10 – Відповідність артефактів розділам пояснювальної записки

Артефакт	Де описано	Що підтверджує
ODD-таблиця	розділ 1.2	межі застосування прототипу
FR/NFR	розділ 2.1	формалізовані вимоги
PlantUML use case	розділ 2.1	користувацькі сценарії
PlantUML component	розділ 2.2	архітектурну структуру
PlantUML activity/sequence/state	розділи 2.3–2.4	логіку виконання і контролю
псевдокод модулів	розділи 3.1–3.2	перехід від архітектури до реалізації
метрики та ризики	розділ 3.3	критичне оцінювання
пакет демонстрації	розділ 3.4	готовність до захисту

У третьому розділі описано реалізаційний рівень програмного прототипу: структуру проєкту, роль Python-екосистеми, модулі завантаження даних, розпізнавання, керування, контролю обмежень і оцінювання результатів.

YOLOv8 і CNN-модель керування подано як навчально-дослідні компоненти з різними ролями: YOLOv8 забезпечує актуальну об’єктну детекцію, а CNN-модель демонструє принцип end-to-end прогнозування керувальної дії. Такий підхід дозволяє зберегти практичну відтворюваність і водночас не змішувати демонстраційний результат із промисловою готовністю.

Сформовано ризики, метрики й дорожню карту подальшого розвитку. Основними напрямами вдосконалення є closed-loop оцінювання, інтеграція з CARLA, стандартизовані сценарії ASAM, заміна детектора, BEV-представлення, автоматизоване тестування та розширення звітних артефактів.

ВИСНОВКИ

У кваліфікаційній роботі розроблено й інженерно обґрунтовано програмний прототип моделювання безпілотного керування автотранспортом. Роботу побудовано відповідно до логіки інженерії програмного забезпечення: від аналізу предметної області та сучасної джерельної бази до формалізації вимог, архітектурного моделювання, опису реалізації, тестової стратегії, оцінювання ризиків і визначення напрямів подальшого розвитку.

У першому розділі проаналізовано автономне керування як кіберфізичну програмну систему. Розглянуто рівні автоматизації SAE, поняття ODD, функціональну безпеку, SOTIF, кібербезпеку, сенсорну архітектуру, camera-only підхід, CAN-телеметрію, модульні та end-to-end методи. Показано, що навчальний прототип не слід класифікувати як готову систему автономності рівня 3 або вище, оскільки він не має промислового fallback, сертифікованого safety case і достатньої сценарної валідації.

У другому розділі сформовано функціональні й нефункціональні вимоги до прототипу, визначено користувацькі сценарії, побудовано архітектуру компонентів, конвеєр даних, послідовність виконання експерименту та модель станів SafetyGuard. Для підтвердження інженерної складової створено PlantUML-діаграми: прецедентів, компонентів, активності, послідовності та станів. Вони показують, що практична частина розглядається не як окремий нейромережевий скрипт, а як програмна система з визначеними взаємодіями, входами, виходами й перевірками.

У третьому розділі описано реалізаційний рівень прототипу. Визначено структуру проєкту, роль Python, TensorFlow/Keras, OpenCV, Ultralytics YOLOv8, CNN-моделі керування, SafetyGuard, Evaluator та звітних артефактів. Показано, що YOLOv8 придатний для актуальної демонстрації об’єктної детекції, а end-to-end CNN - для пояснення принципу прогнозування керування в межах заданого сценарію.

Мету роботи досягнуто: програмний прототип подано як структуроване програмне рішення з визначеною операційною областю, вимогами,

архітектурою, UML-моделями, програмними компонентами, тестовими сценаріями та критичним оцінюванням. Завдання роботи виконано: проаналізовано сучасні підходи, визначено межі відповідальності, сформовано FR/NFR, побудовано UML-діаграми, описано реалізацію модулів, розроблено тестову стратегію та сформовано дорожню карту розвитку.

Практична цінність роботи полягає в тому, що отриманий матеріал може бути використаний як навчально-дослідний приклад для курсів з інженерії програмного забезпечення, машинного навчання, комп'ютерного зору й тестування програмних систем. Робота демонструє, як поєднати ML-компоненти з вимогами, архітектурою, журналюванням, метриками та сценарним оцінюванням.

Окремим результатом є уточнення критеріїв інтерпретації роботи прототипу. Якщо модель розпізнає об'єкти або прогнозує керування на тестовому фрагменті, це ще не означає готовність до реального дорожнього використання. Тому в роботі запропоновано розглядати кожен запуск через ODD, якість вхідних даних, confidence, обмеження SafetyGuard, наявність журналу та зіставлення prediction/target.

Побудовані UML-діаграми виконують не декоративну, а інженерну функцію. Діаграма прецедентів задає межі користувацьких сценаріїв, компонентна діаграма фіксує відповідальність модулів, діаграма активності описує алгоритм обробки даних, діаграма послідовності показує порядок взаємодії компонентів, а діаграма станів формалізує логіку попередження та блокування небезпечних команд.

Важливим підсумком є те, що розроблений підхід дозволяє відокремити три рівні результату: алгоритмічний, програмно-архітектурний та експериментальний. Алгоритмічний рівень пов'язаний з CNN, YOLOv8, детекцією та прогнозуванням. Програмно-архітектурний рівень представлений компонентами, інтерфейсами, журналюванням і SafetyGuard. Експериментальний рівень представлений сценаріями, метриками, демонстраційними кадрами та звітними файлами.

Робота також показала, що для ML-прототипів недостатньо оцінювати лише точність моделі. Потрібно контролювати якість входів, часову узгодженість даних, стабільність конвеєра, поведінку за помилок, допустимість вихідних команд і відтворюваність запусків. Такий підхід робить оцінювання ближчим до практики інженерії програмного забезпечення, де важлива не тільки модель, а й система навколо неї.

Сформовані в роботі матеріали можуть бути використані як основа для подальших навчальних експериментів. На їх базі можна змінювати моделі, додавати нові сценарії, порівнювати метрики, розширювати SafetyGuard і підключати симуляційне середовище. Завдяки модульності й текстовим UML-описам така модернізація може виконуватися поступово без втрати зрозумілості структури.

Подальший розвиток доцільно спрямувати на підключення CARLA, опис сценаріїв засобами OpenSCENARIO/OpenDRIVE, порівняння різних конфігурацій YOLOv8, реалізацію BEV або sensor fusion, розширення тестових наборів, автоматизацію запуску експериментів і формування звітів. Це дозволить поступово перейти від навчального прототипу до більш зрілої експериментальної платформи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Guide to the Software Engineering Body of Knowledge (SWEBOK Guide), Version 4.0 / ed. H. Washizaki. IEEE Computer Society, 2024. URL: https://www.computer.org/education/bodies-of-knowledge/software-engineering (дата звернення: 17.06.2026).

2. ISO/IEC/IEEE 29148:2018. Systems and software engineering – Life cycle processes – Requirements engineering. Geneva : ISO, 2018. URL: https://www.iso.org/standard/72089.html (дата звернення: 17.06.2026).

3. Badue C., Guidolini R., Carneiro R. V. та ін. Self-driving cars: A survey. Expert Systems with Applications. 2021. Vol. 165. Article 113816. DOI: 10.1016/j.eswa.2020.113816.

4. Chen L., Wu P., Chitta K., Jaeger B., Geiger A., Li H. End-to-end Autonomous Driving: Challenges and Frontiers. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2024. URL: https://arxiv.org/abs/2306.16927 (дата звернення: 17.06.2026).

5. Chitta K., Prakash A., Jaeger B., Yu Z., Renz K., Geiger A. TransFuser: Imitation with Transformer-Based Sensor Fusion for Autonomous Driving. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2023. DOI: 10.1109/TPAMI.2022.3200245.

6. Hu Y., Yang J., Chen L. та ін. Planning-oriented Autonomous Driving. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2023. P. 17853–17862. URL: https://openaccess.thecvf.com/content/CVPR2023/html/Hu_Planning-Oriented_Autonomous_Driving_CVPR_2023_paper.html (дата звернення: 17.06.2026).

7. Liu Z., Tang H., Amini A., Yang X., Mao H., Rus D., Han S. BEVFusion: Multi-Task Multi-Sensor Fusion with Unified Bird’s-Eye View Representation. arXiv:2205.13542. 2022. URL: https://arxiv.org/abs/2205.13542 (дата звернення: 17.06.2026).

8. SAE J3016. Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles. SAE International. URL: https://www.sae.org/standards/j3016-taxonomy-definitions-terms-related-driving-automation-systems-road-motor-vehicles (дата звернення: 17.06.2026).

9. ISO 34503:2023. Road vehicles – Test scenarios for automated driving systems – Specification for operational design domain. Geneva : ISO, 2023. URL: https://www.iso.org/standard/78952.html (дата звернення: 17.06.2026).

10. ISO 26262-1:2018. Road vehicles – Functional safety – Part 1: Vocabulary. Geneva : ISO, 2018. URL: https://www.iso.org/standard/68383.html (дата звернення: 17.06.2026).

11. ISO 21448:2022. Road vehicles – Safety of the intended functionality. Geneva : ISO, 2022. URL: https://www.iso.org/standard/77490.html (дата звернення: 17.06.2026).

12. ISO/SAE 21434:2021. Road vehicles – Cybersecurity engineering. Geneva : ISO, 2021. URL: https://www.iso.org/standard/70918.html (дата звернення: 17.06.2026).

13. Bojarski M., Del Testa D., Dworakowski D. та ін. End to End Learning for Self-Driving Cars. arXiv:1604.07316. 2016. URL: https://arxiv.org/abs/1604.07316 (дата звернення: 17.06.2026).

14. Xu H., Gao Y., Yu F., Darrell T. End-to-end Learning of Driving Models from Large-scale Video Datasets. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2017. P. 3530–3538. DOI: 10.1109/CVPR.2017.376.

15. Dosovitskiy A., Ros G., Codevilla F., Lopez A., Koltun V. CARLA: An Open Urban Driving Simulator. Proceedings of the 1st Annual Conference on Robot Learning (CoRL). 2017. P. 1–16. URL: https://proceedings.mlr.press/v78/dosovitskiy17a.html (дата звернення: 17.06.2026).

16. ASAM OpenDRIVE®. ASAM e.V. URL: https://www.asam.net/standards/detail/opendrive/ (дата звернення: 17.06.2026).

17. ASAM OpenSCENARIO®. ASAM e.V. URL: https://www.asam.net/standards/detail/openscenario/ (дата звернення: 17.06.2026).

18. Tian Y., Pei K., Jana S., Ray B. DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars. Proceedings of the 40th International Conference on Software Engineering (ICSE). 2018. P. 303–314. DOI: 10.1145/3180155.3180220.

19. Zhang M., Zhang Y., Zhang L., Liu C., Khurshid S. DeepRoad: GAN-based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems. Proceedings of the 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE). 2018. P. 132–142. DOI: 10.1145/3238147.3238187.

20. Gao H., Wang Z., Li Y., Long K., Yang M., Shen Y. A Survey for Foundation Models in Autonomous Driving. arXiv:2402.01105. 2024. URL: https://arxiv.org/abs/2402.01105 (дата звернення: 17.06.2026).

21. Gao Y., Piccinini M., Zhang Y. та ін. Foundation Models in Autonomous Driving: A Survey on Scenario Generation and Scenario Analysis. arXiv:2506.11526. 2025. URL: https://arxiv.org/abs/2506.11526 (дата звернення: 17.06.2026).

22. TensorFlow Documentation. URL: https://www.tensorflow.org/api_docs (дата звернення: 17.06.2026).

23. Keras Documentation. URL: https://keras.io/api/ (дата звернення: 17.06.2026).

24. OpenCV Documentation. URL: https://docs.opencv.org/ (дата звернення: 17.06.2026).

25. Python 3 Documentation. URL: https://docs.python.org/3/ (дата звернення: 17.06.2026).

26. Ultralytics YOLOv8. Ultralytics Documentation. URL: https://docs.ultralytics.com/models/yolov8/ (дата звернення: 17.06.2026).

27. Geiger A., Lenz P., Stiller C., Urtasun R. Vision meets robotics: The KITTI dataset. The International Journal of Robotics Research. 2013. Vol. 32, No. 11. P. 1231–1237. DOI: 10.1177/0278364913491297.

28. Caesar H., Bankiti V., Lang A. H. та ін. nuScenes: A multimodal dataset for autonomous driving. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2020. P. 11621–11631.

29. Sun P., Kretzschmar H., Dotiwalla X. та ін. Scalability in Perception for Autonomous Driving: Waymo Open Dataset. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2020. P. 2446–2454.