НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут прикладного системного аналізу

Кафедра штучного інтелекту

До захисту допущено:

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою
«Системи і методи штучного інтелекту»

спеціальності 122 «Комп'ютерні науки»

на тему: «Методи штучного інтелекту для керування транспортним засобом в симуляції складного середовища»

Виконала:
студентка IV курсу, групи КІ-12
Лоєнко Вікторія Григорівна _____

Керівник:
асистент кафедри ШІ, к.т.н,
Осауленко В’ячеслав Миколайович _____

Консультант з економічного розділу:
доцент кафедри ЕК ФММ, к.е.н., доцент,
Рощина Надія Василівна _____

Консультант з нормоконтролю:
фахівець першої категорії кафедри ШІ, к.т.н., доцент
Комариста Богдана Миколаївна _____

Рецензент:
доцент кафедри ММАД ФТІ, к.т.н., доцент,
Хайдуров Владислав Володимирович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ
завідувачка кафедри
_____ Ірина ДЖИГИРЕЙ
«31» січня 2025р.

ЗАВДАННЯ
на дипломну роботу студенту
Лоєнко Вікторії Григорівні

1. Тема роботи «Методи штучного інтелекту для керування транспортним засобом в симуляції складного середовища», керівник роботи Осауленко В’ячеслав Миколайович, асистент кафедри штучного інтелекту, кандидат технічних наук в області штучного інтелекту, затверджені наказом по НН ІПСА від «26» травня 2025 р. № 1759-с.

2. Термін подання студентом роботи «5» червня 2025 року.

3. Вихідні дані до роботи: програмне забезпечення для вирішення задачі керування транспортним засобом в симуляції складного середовища

4. Зміст роботи: теоретичні відомості, програмна реалізація, розробка додатку, функціонально-вартісний аналіз програмного продукту

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н. В., доцент кафедри економічної кібернетики	05.02.2025	03.06.2025

7. Дата видачі завдання «05» лютого 2025 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Вибір теми і постановка задач дослідження	15.03.2025	Виконано
2	Аналіз актуальності задачі дослідження	20.03.2025	Виконано
3	Огляд літератури за темою ДР	05.04.2025	Виконано
4	Написання першого розділу ДР	20.04.2025	Виконано
5	Написання другого розділу ДР	03.05.2025	Виконано
6	Розробка програмного продукту	15.05.2025	Виконано
7	Написання третього розділу ДР	17.05.2025	Виконано
8	Написання та обрахунок економічної частини ДР	20.05.2025	Виконано
9	Аналіз та впорядкування результатів	30.05.2025	Виконано
10	Оформлення та редагування ДР	02.06.2025	Виконано
11	Підготовка презентації для доповіді	03.06.2025	Виконано

Студент Вікторія ЛОЄНКО

Керівник В’ячеслав ОСАУЛЕНКО

АНОТАЦІЯ

Дипломна робота:154 сторінки, 30 рисунків, 22 таблиці, 2 додатки та 41 посилання.

ШТУЧНИЙ ІНТЕЛЕКТ, НАВЧАННЯ З ПІДКРІПЛЕННЯМ, АВТОНОМНЕ КЕРУВАННЯ, МУЛЬТИАГЕНТНІ СИСТЕМИ, МУЛЬТИСЕНСОРНА ІНТЕГРАЦІЯ, СИМУЛЯЦІЙНЕ СЕРЕДОВИЩЕ, ДИНАМІКА ТРАНСПОРТНИХ ЗАСОБІВ, КРИТИЧНІ СЦЕНАРІЇ.

Метою дослідження є розробка та експериментальне обґрунтування нового підходу до автономного керування транспортним засобом у складних симульованих умовах на основі поєднання методів навчання з підкріпленням, самонавчання та мультисенсорної інтеграції з урахуванням фізично достовірної динаміки й мультиагентної взаємодії.

Об’єкт дослідження – процес автономного керування транспортними засобами в симульованих середовищах із динамічними дорожніми й погодними умовами.

Предмет – методи навчання з підкріпленням для формування стратегій керування на основі мультисенсорних даних, адаптації до змін середовища й взаємодії з іншими агентами.

Застосовано формалізацію задач як марковського процесу прийняття рішень (MDP) і диференціальні ігри для моделювання взаємодії агентів. Для навчання використано алгоритми RL (PPO, HAPPO), meta-learning, curriculum learning, поєднання Behavior Cloning і RL. Самонавчання покращує подання станів, а генеративні моделі (VAE, GAN) створюють складні сценарії. Використано підхід централізованого тренування з децентралізованим виконанням (CTDE). Особливу увагу приділено інтеграції даних (LiDAR, камери, IMU) та масштабованому тренуванню в середовищі Isaac Gym. Розроблено функцію винагороди, що враховує безпеку, стабільність та взаємодію з оточенням.

Результати можуть бути використані для підвищення ефективності автономного керування, розробки симуляційних платформ і впровадження інтелектуальних транспортних систем.

ABSTRACT

Thesis: 154 pages, 30 figures, 22 tables, 2 appendices, and 41 references.

ARTIFICIAL INTELLIGENCE, REINFORCEMENT LEARNING, AUTONOMOUS DRIVING, MULTI-AGENT SYSTEMS, MULTI-SENSOR INTEGRATION, SIMULATION ENVIRONMENT, VEHICLE DYNAMICS, CRITICAL SCENARIOS.

The aim of the research is to develop and experimentally justify a new approach to autonomous vehicle control in complex simulated environments. The approach is based on the integration of reinforcement learning, self-supervised learning, and multisensor data fusion, taking into account physically accurate dynamics and multi-agent interaction.

The object of the research is the process of autonomous vehicle control in simulated environments with dynamic road and weather conditions.

The subject of the study includes reinforcement learning methods for developing control strategies based on multisensor data, adaptation to changing conditions, and interaction with other agents in a multi-agent environment.

The task was formalized as a Markov Decision Process (MDP), and differential game theory was used to model agent interactions. For agent training, various RL algorithms (PPO, HAPPO), meta-learning for real-time adaptation, curriculum learning for gradual task complexity, and a combination of Behavior Cloning and RL were used to form robust policies. Self-supervised learning improved state representations via pretraining, while generative models (VAE, GAN) were applied to create complex and critical scenarios. The centralized training with decentralized execution (CTDE) approach was employed.

Special attention was given to engineering-level multisensor integration (LiDAR, cameras, IMU) and scalable training using the Isaac Gym environment. A combined reward function was designed, taking into account safety, motion stability, and environmental interaction.

The results can be applied to enhance the reliability and efficiency of autonomous control systems under complex conditions, support the development of simulation platforms for training autonomous agents, and advance intelligent transportation and safety technologies.

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ 10
ВСТУП 11
РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ 13
1.1 Актуальність задач автономного керування транспортними засобами 13
1.2 Постановка задачі автономного керування в екстремальних умовах 14
1.2.1 Опис задачі 14
1.2.2 Особливості екстремальних умов 15
1.3 Загальні характеристики середовищ та виклики при екстремальному керуванні 17
1.4 Мультиагентна взаємодія в контексті автономного водіння 18
1.5 Автономне керування як марковський процес прийняття рішень 21
1.5.1 Основні поняття: стани, дії, винагороди 22
1.5.2 Стратегії, цільові функції та оптимізація політик 22
1.6 Методи навчання з підкріпленням для задач автономного керування 24
1.6.1 DQN, PPO, A3C RL-алгоритми 24
1.6.2 Моделювання фізики та використання MPC в поєднанні з RL 27
1.6.3 Методи мета-навчання та адаптації політик 28
1.7 Самокероване навчання агентів у складних умовах 29
1.8 Застосування генеративного штучного інтелекту для генерації складних сценаріїв 31
1.9 Федеративне навчання та розподілене тренування агентів 33
Висновки до розділу 1 36
РОЗДІЛ 2 МАТЕМАТИЧНІ ТА МЕТОДИЧНІ ОСНОВИ РОБОТИ 38
2.1 Архітектура мультиагентної системи 38
2.1.1 Основи мультиагентної взаємодії в системах автономного керування 38
2.1.2 Побудова симуляційного середовища в Isaac Sim 39

2.1.3 Інтеграція фізично достовірної динаміки транспортного засобу в Isaac Gym 42
2.1.4 Використання сенсорів: камери, LiDAR, IMU – структура та обробка даних 43
2.2 Формалізація задачі як задачі навчання з підкріпленням 45
2.2.1 Математичне формулювання задачі як марковського процесу прийняття рішень (MDP) 46
2.2.2 Визначення простору станів, дій та функції винагороди 47
2.2.2.1 Проєктування станів і дій для базового керування (PPO/DQN) 47
2.2.2.2 Проєктування винагород для стабільного слідування траєкторії 49
2.2.2.3 Розширення простору станів у задачі маневрування (Imitation Learning + RL) 52
2.2.2.4 Формування комбінованої функції винагороди для уникнення перешкод 54
2.2.2.5 Інкорпорація сегментаційної інформації в простір спостережень 55
2.2.2.6 Проєктування станів і винагород для задачі адаптації до зчеплення (лід, сніг) 56
2.2.2.7 Формалізація взаємодії з іншими водіями як диференціальна гра 58
2.2.2.8 Генерація складних сценаріїв за допомогою генеративних моделей 62
2.2.2.9 Meta-RL: адаптація політики до нових умов на ходу 62
2.2.2.10 Self-Supervised Learning: формування позитивних пар для навчання 63
2.2.3 Обмеження та особливості RL у фізично достовірному моделюванні 64
2.3 Реалізація політик та стратегій керування 66
2.3.1 Огляд типів політик: стохастичні vs детерміновані 66

2.3.2 Вибір архітектур: Actor-Critic, PPO, SAC 67
2.3.3 Особливості реалізації політик в мультиагентному середовищі Isaac Gym 68
2.3.4 Координація дій між агентами: централізоване vs децентралізоване управління 69
2.4 Конфігурація середовища тренування 70
2.4.1 Параметризація середовища: екстремальні умови (дощ, ожеледь, пил, аварії) 70
2.4.2 Структура сценаріїв: рельєф, щільність руху, типи загроз 72
2.4.3 Масштабоване паралельне тренування в Isaac Gym 74
2.5 Новизна застосованого підходу 75
2.5.1 Модифікація політик із врахуванням мультисенсорної інформації 76
2.5.2 Генеративні моделі для симуляції рідкісних критичних ситуацій 77
2.5.3 Комбіноване навчання з підкріпленням та самокероване навчання 78
2.5.4 Стійкість навчання за рахунок адаптації до симуляційних похибок 80
Висновки до розділу 2 81
РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ 82
3.1 Архітектура реалізації експериментального середовища 82
3.1.1 Структура середовища на базі Isaac Gym 83
3.1.2 Організація мультиагентної симуляції 87
3.2 Реалізація базової політики PPO 89
3.2.1 Результати використання PPO алгоритму для простого руху з точки А в точку Б 91
3.3 Поведінкове попереднє навчання 93
3.4 Інтеграція HAPPO для мультиагентної координації 94
3.4.1 Формалізація взаємодії як диференціальної гри 96
3.4.2 Centralized Training with Decentralized Execution (CTDE) 97
3.4.3 Результати використання HAPPO алгоритму для руху з точки А в точку Б в складнозмодельованому середовищі 98
3.5 Інкорпорація сегментаційної інформації та використання сегментаційної

моделі для визначення полоси 100
3.6 Адаптація до нових дорожніх умов (Meta-RL) 102
3.7 Генерація складних сценаріїв за допомогою генеративних моделей 103
3.8 Self-Supervised Learning для покращення представлень 104
Висновки до розділу 3 106
РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ 108
4.1 Постановка задачі 109
4.2 Обґрунтування функцій програмного продукту 111
4.3 Вибір параметрів для програмного продукту 113
4.4 Експертний аналіз параметрів 117
4.5 Аналіз якості реалізації варіантів функцій 121
4.6 Економічний аналіз розробки 122
4.7 Вибір оптимального варіанту реалізації ПП 129
Висновки до розділу 4 130
ВИСНОВКИ 131
ПЕРЕЛІК ПОСИЛАНЬ 133
ДОДАТОК А ЛІСТИНГ ПРОГРАМНОГО КОДУ 139
ДОДАТОК Б ЕЛЕКТРОННА ПРЕЗЕНТАЦІЯ 148

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

ШІ – штучний інтелект.

MDP – Markov Decision Process (Марковський процес прийняття рішень).

RL – reinforcement learning (навчання з підкріпленням).

MARL – мультиагентне навчання з підкріпленням.

ТЗ – транспортний засіб.

LiDAR – Light Detection and Ranging.

ВСТУП

Стрімкий розвиток штучного інтелекту та прагнення суспільства до автоматизації всіх процесів, з яких складається буденне життя, та створення ідеальних умов, у яких функціонування людства стає більш організованим, створюють нові виклики для інженерів та розробників з більш складною специфікою, аніж початкова задача[1].

Автоматизація керування автомобілями вже демонструє високий рівень розвитку та ефективні результати згідно з практичними тестами та впровадженням даної технології у реальне життя. компанія Waymo надає понад 200 000 оплачуваних поїздок щотижня в містах США, таких як Сан-Франциско, Лос-Анджелес та Фінікс, і нещодавно розпочала тестування автономних автомобілів у Токіо [1]. У Великій Британії проєкт CAVForth запровадив перший у світі повнорозмірний автономний автобусний сервіс, який працював на маршруті між Ferrytoll Park and Ride та Edinburgh Park [2]. У Японії компанія Nissan проводить випробування безпілотних транспортних засобів на жвавих вулицях Йокогами, використовуючи комбінацію камер, радарів та LiDAR-сенсорів для навігації в умовах інтенсивного трафіку [3].

Автоматизація керування автомобілями стикається з низкою викликів у складних погодних та дорожніх умовах, таких як сніг, туман, ожеледь та погане дорожнє покриття. Ці умови можуть значно знижувати ефективність систем автономного водіння, впливаючи на точність сенсорів, алгоритмів розпізнавання та навігації [4]. Забезпечення надійного та безпечного керування транспортними засобами в екстремальних дорожніх умовах – таких як зниження коефіцієнта тертя, занос, пошкодження дорожнього покриття та

[1]Тут і нижче використано такий інструмент штучного інтелекту як чат-бот з генеративним штучним інтелектом ChatGPT виключно для корегування та редагування тексту, створеного автором цієї дипломної роботи, на основі автоматизованої перевірки граматики, структури та стилю, що відповідає Політиці використання штучного інтелекту для академічної діяльності в КПІ ім. Ігоря Сікорського (протокол №11 Вченої ради КПІ ім. Ігоря Сікорського від 11 грудня 2023 р.).

аварійні ситуації все ж залишається складним та досі нерозв’язаним завданням. Традиційні методи, такі як класичні алгоритми навчання з підкріпленням, PID-контролери та евристичні підходи, часто не забезпечують достатньої адаптивності та стійкості в умовах високої невизначеності [5, 6].

У межах даної роботи було розроблено симуляційне середовища для моделювання екстремальних дорожніх умов з можливістю параметризації характеристик покриття, рівня пошкоджень дороги, зовнішніх перешкод та раптових змін динамічних параметрів руху. Також було створено багатоагентну систему навчання з підкріпленням, яка включає в себе основного агента – керуючого алгоритму транспортного засобу та допоміжного агента – аналітичного модулю прогнозування ризиків, що використовує фізично обґрунтовані моделі для оцінювання динаміки автомобіля. Розроблено генеративну модель штучного інтелекту для синтезу реалістичних сценаріїв дорожніх інцидентів з урахуванням реальних статистичних даних про ДТП та погодні умови. Упровадження методів мета-навчання забезпечує швидку адаптацію агентів до нових середовищ та нестандартних дорожніх умов без необхідності повного перенавчання моделі. Реалізація самокерованого (self-supervised) та федеративного навчання сприятиме децентралізованому накопиченню знань без порушення конфіденційності даних користувачів.

Таким чином, запропонований підхід спрямований на створення адаптивної, стійкої та безпечної системи автономного керування транспортним засобом, здатної ефективно функціонувати в умовах високої невизначеності та екстремальних дорожніх ситуацій.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність задач автономного керування транспортними засобами

Останні роки характеризуються стрімким розвитком автономних транспортних систем, що пов’язано зі зростанням потреби у підвищенні безпеки дорожнього руху, зменшенні впливу людського фактора та оптимізації транспортних потоків. Автономне керування транспортними засобами (ТЗ) дозволяє не лише знизити кількість аварій, викликаних помилками водія, але й покращити енергоефективність, скоротити затори та забезпечити мобільність для людей з обмеженими фізичними можливостями.

Особливу важливість набуває автономне керування в екстремальних умовах – таких як ожеледиця, снігопад, дощ, туман, недостатнє освітлення або складний рельєф місцевості. У таких ситуаціях знижується ефективність традиційних алгоритмів, а середовище стає частково або повністю невизначеним, що вимагає адаптивних та стійких до зовнішніх змін стратегій поведінки.

Крім того, поява мультиагентних систем у транспортній галузі – наприклад, у вигляді кооперації між автономними автомобілями – відкриває нові можливості для реалізації узгодженого, безпечного та ефективного руху у динамічному середовищі. Для цього необхідно враховувати не лише локальну поведінку кожного агента, але й глобальні аспекти – взаємодію, розподіл інформації, координацію та ухвалення рішень у реальному часі.

Усе це зумовлює зростаючий інтерес наукової спільноти та індустрії до розробки інтелектуальних підходів до автономного керування, зокрема з використанням методів навчання з підкріпленням, генеративного штучного інтелекту та самокерованого навчання, які здатні пристосовуватись до нових сценаріїв без ручного програмування та масштабуватись на великі системи.

Саме тому дослідження в цій галузі є надзвичайно актуальними, практично значущими та перспективними в контексті майбутнього транспорту.

1.2 Постановка задачі автономного керування в екстремальних умовах

Задача автономного керування транспортним засобом в екстремальних умовах є складним багатокомпонентним завданням, яке включає в себе численні підзадачі, такі як виявлення навколишнього середовища, планування траєкторії руху, прийняття рішень в умовах обмеженої інформації, а також адаптація до змінних умов. При цьому враховуються як фізичні аспекти поведінки автомобіля, так і психологічні фактори, пов'язані з інтерпретацією і реагуванням на ситуації, які можуть виникати на дорозі.

У загальному випадку, задачу можна розглядати як задачу Марковського процесу прийняття рішень (MDP) [7], де агент (автономний транспортний засіб) взаємодіє з навколишнім середовищем для досягнення оптимального результату, враховуючи такі фактори, як швидкість, напрямок руху, умови дорожнього покриття та наявність перешкод. Завдання ускладнюється через низьку видимість, змінні погодні умови (ожеледиця, снігопад, дощ), а також необхідність взаємодії з іншими транспортними засобами.

1.2.1 Опис задачі

Модель середовища. Моделювання середовища для автономного водіння в екстремальних умовах повинно включати змінні, що описують траєкторію руху транспортного засобу, умови покриття дороги (лід, сніг, дощ), стан

автомобіля (швидкість, напрямок, зчеплення з дорогою) та можливі перешкоди (інші транспортні засоби, нерівності на дорозі).

Цілі агентів. Основною метою є забезпечення безпеки руху та оптимального керування транспортним засобом при змінних умовах середовища. Агент повинен вчасно реагувати на потенційні небезпеки (захист від зіткнень з іншими транспортними засобами або перешкодами) та мінімізувати пошкодження в екстремальних умовах, таких як ковзання на льоду чи гальмування на мокрій дорозі.

Дії агента. Агент може здійснювати різноманітні дії, які включають маніпулювання газом, гальмами та кермом для регулювання швидкості, напрямку руху та рівня зчеплення з дорогою.

Нагорода. Нагорода, яку отримує агент, має бути сформульована таким чином, щоб мотивувати агентів до оптимальних маневрів та мінімізації ризику. Враховуються як негативні аспекти (штрафи за зіткнення з іншими об'єктами, виїзд за межі траєкторії), так і позитивні (бонуси за збереження стабільної швидкості, плавні маневри, ефективне уникання перешкод).

Обмеження. Особливістю задачі є обмеження, які виникають через реалістичні фізичні властивості автомобіля та обмеження самого середовища (наприклад, обмежена видимість на зимовій дорозі, зміна стану покриття дороги).

1.2.2 Особливості екстремальних умов

Екстремальні умови дорожнього середовища становлять одну з найскладніших задач для систем автономного керування транспортними засобами: як і для самих водіїв, так і для автономного керування в контексті використання методів та підходів штучного інтелекту.

У реальних умовах транспортний засіб може зіткнутися з широким спектром дорожніх покриттів, таких як:

- лід та/або сніг, що знижують коефіцієнт зчеплення з дорогою;
- мокрий або брудний асфальт, що змінює умови гальмування та керованості;
- пісок, гравій, що спричиняє нестабільність та ковзання;
- вибоїни, що є частою проблемою та причиною багатьох поломок та аварійних ситуацій.

У таких ситуаціях виникає потреба в адаптивному моделюванні фізичних характеристик середовища у режимі реального часу. Мультиагентні системи можуть бути організовані так, щоб кожен агент відповідав за моніторинг певного аспекту стану дороги (наприклад, зчеплення, кут дрейфу, температура поверхні) з подальшою агрегацією результатів для прийняття колективного рішення.

Автономна система повинна не лише реагувати на поточні умови, але й прогнозувати зміни в оточенні або ж бути здатною до будь-якого обігу подій. Для цього можливо використовувати передбачувальних агентів для побудови ймовірних карти змін середовища (наприклад, ймовірність обльоденіння певної ділянки шляху) або ж механізми самоадаптації, коли агент коригує свою політику залежно від змін у динаміці навколишнього середовища.

Самокероване навчання дозволяє накопичувати знання про нові ситуації без явного маркування даних, що є особливо цінним в умовах, де неможливо мати достовірно та у якійсь мірі повну інформацію.

Поведінка транспортного засобу в умовах ковзання, екстремального гальмування, або нестійкості під час поворотів потребує реалістичного фізичного моделювання. У нашому випадку агенти, які навчаються за допомогою RL, можуть адаптувати свої дії, враховуючи зміни в фізичних параметрах транспортного засобу та адаптивна політика руху має передбачати стратегії утримання стійкості за умов частих змін навантаження на вісь, центру маси чи інші фізичні характеристики.

1.3 Загальні характеристики середовищ та виклики при екстремальному керуванні

Для того, щоб у повній мірі виконати поставлену задачу, важливо провести повний аналіз та поставити конкретні цілі та вимоги до середовищ. Це дозволяє виокремити специфіку таких середовищ, зрозуміти природу викликів та сформувати вимоги до інтелектуальних систем керування.

Під "екстремальними умовами" у контексті автономного керування маються на увазі такі середовища, у яких:

- фізичні властивості дороги та оточення є нестабільними або швидко змінюваними;
- присутні непередбачувані об'єкти чи учасники дорожнього руху;
- обмежені сенсорні спостереження (через погодні умови, нічний час тощо);
- недостатня або відсутня інформація про глобальну структуру навколишнього середовища.

До таких середовищ належать: засніжені та обмерзлі дороги, що ускладнюють оцінку коефіцієнта зчеплення з дорогою та відносних коефіцієнтів тертя; бездоріжжя або сільські території з непередбачуваними природними перешкодами; міські середовища під час катастроф чи евакуацій, де правила руху можуть бути порушені; низька видимість (туман, дощ, дим), яка обмежує функціональність сенсорів.

У предметній області ми можемо виділити кілька груп викликів, таких як: перцептивні виклики, тобто, забезпечення надійності даних, зібраних із сенсорів, оскільки викривлення зібраних даних можуть спричинити неправильне розпізнавання об’єктів, пропуск перешкод або ж помилок у легалізації; побудова складних моделей динаміки, які можуть адаптуватися до змінних фізичних параметрів у режимі реального часу через нелінійну та неочікувану поведінку системи "транспортний засіб – поверхня дороги";

імовірнісне та сценарне моделювання, врахування різних варіантів розвитку подій за обмеженої вхідної інформації; здатність до швидкого прийняття рішень, де час реакції обмежений, а помилка може мати критичні наслідки, необхідно моделювати ризики та оптимізувати дії в контексті кількох суперечливих цілей зважаючи на 3 пріоритетних критерії: безпека, ефективність, комфорт; самостійна адаптація до нових умов, не передбачених на етапі проєктування (open-ended learning).

1.4 Мультиагентна взаємодія в контексті автономного водіння

У контексті екстремального керування транспортними засобами мультиагентна взаємодія набуває особливої актуальності, оскільки сучасне дорожньо-транспортне середовище є складною динамічною системою, в якій взаємодіють численні суб'єкти – як автономні, так і керовані людьми. Це створює потребу в побудові систем, що здатні не лише адаптувати поведінку до змін середовища, а й узгоджено взаємодіяти з іншими агентами задля досягнення безпечного й ефективного функціонування.

Мультиагентна система (MAC) – сукупність з декількох агентів штучного інтелекту, які працюють колективно для виконання завдань від імені користувача або іншої системи. Кожен агент в MAC має індивідуальні властивості, але всі агенти діють спільно, щоб досягти бажаних глобальних властивостей. Мультиагентні системи є цінними для виконання великомасштабних, складних завдань, які можуть охоплювати сотні, якщо не тисячі, агентів [12]. Тобто, кожен з агентів має здатність до таких незалежних властивостей, як локальне прийняття рішень, які є під контролем даного агента; комунікація або обмін інформацією (повна або неповна); координація дій у процесі досягнення індивідуальних або колективних цілей.

Для сфери автономного водіння можуть бути використані такі агенти, як окремі роботизовані транспортні засоби, дорожня інфраструктура така як світлофори, сенсори або ж інформаційні системи, пішоходи, тварини або інші учасники дорожнього руху, змодельовані як агенти для подальшого прогнозування.

У контексті автономного водіння можна виокремити кілька рівнів взаємодії між агентами.

1. Невзаємодіючі агенти. Кожен агент діє автономно без врахування присутності інших. Такий підхід обмежений і небезпечний у реальних умовах, можливий тільки для якихось ізольованих задач, де є впевненість у відсутності небезпеки та навколишніх подразників, тобто, автономне водіння у лабіринті за умов наявності тільки одного транспортного засоба.
2. Індивідуальне передбачення (intent recognition). Агенти намагаються оцінити ймовірні наміри інших учасників руху, використовуючи моделі прогнозування. Часто зустрічається реалізація через нейромережі або байєсівські підходи [13]. Наприклад, прогнозування агента-автомобіля, що ймовірнісний агент-дитина буде перебігати через дорогу.
3. Опосередкована взаємодія через середовище. Взаємодія відбувається не напряму через комунікацію, а через спостереження за змінами в оточенні, спричиненими діями інших агентів. Наприклад, маневр іншого авто як сигнал небезпеки або ж початок переходу дороги у непризначеному для цього місці пішоходом.
4. Пряма координація. У разі використання V2V (vehicle-to-vehicle) або V2X (vehicle-to-everything) комунікації можлива явна координація дій, як-от колективне уникнення аварії або узгоджене гальмування [14].

Мультиагентні системи мають досить непоганий перформенс в екстремальних умовах, тому можемо визначити низку переваг даної

конфігурації, таких як масштабованість – додавання нових агентів не порушує функціонування системи; гнучкість – кожен агент може адаптуватися до змін незалежно один від одного; колективне навчання – можливість обміну досвідом між агентами (використовуючи генеративні моделі або самонавчання); стійкість – розподілена система менш вразлива до виходу з ладу окремих елементів.

Попри вище перелічені переваги, мультиагентний підхід має і свої виклики, серед яких можемо виокремити проблему узгодження політик дій – кожен з агентів у мультиагентній системі з більшою імовірністю має власні цілі та стратегії дій, що не завжди узгоджуються з загальною метою системи, такою як забезпечення безпеки всіх учасників дорожнього руху, що створює виклик у координації дій агентів для досягнення спільної мети без конфліктів; відсутність централізованого контролю – відсутній єдиний центр управління, що може призвести до непередбачуваних результатів, оскільки агенти приймають рішення на основі локальної інформації без врахування глобального стану системи, стає більш проблематичним при недостатній комунікації агентів; гетерогенність агентів, або ж нестача спільного контексту – агенти використовують різні сенсори, алгоритми та уявлення про середовище, що ускладнює їх взаємодію та координацію, що в свою чергу призводить до неправильного розуміння ситуацій та невідповідних чи неузгоджених дій; конфлікти інтересів – оптимальний варіант дій одного агента створює перешкоди для оптимальної поведінки іншого агента.

Для більш швидкого та ефективного вирішення проблем у мультиагентній системі варто використати мультиагентне навчання з підкріпленням (англ. Multi-agent reinforcement learning, MARL) - це підрозділ навчання з підкріплення, що зосереджений на вивченні поведінки декількох агентів, які співіснують у спільному середовищі: кожен агент мотивується власною винагородою і виконує дії для просування власних інтересів; в деяких середовищах ці інтереси протилежні інтересам інших агентів, що призводить до складної групової динаміки [15], також допустимим варіантом покращення ефективності тренувань є використання генеративного ШІ для більшої

сценарної різноманітності та можливості передбачити більшість ситуацій, що можуть трапитися під час поїздки.

1.5 Автономне керування як марковський процес прийняття рішень

Автономне керування транспортним засобом доцільно розглядати як задачу прийняття рішень у складному динамічному середовищі, однією з найбільш придатних математичних моделей для формалізації даної задачі є Марковський процес прийняття рішень (Markov Decision Process, далі - MDP). MDP визначається множиною станів ($s \in S$), множиною всіх можливих дій ($a \in A$), функцією переходу ($T(s, a, s')$), функцією винагороди ($R(s)$) та коефіцієнтом дисконтування (γ) [8]. Щоб зробити модель математично керованою, коефіцієнт дисконтування обмежується діапазоном $0<\gamma<1$. У даному дослідженні кортеж $\{S, A, T, R, \gamma\}$ представляє MDP.

Агент (наприклад, особа, яка приймає рішення, водій тощо) виконує дію, ґрунтуючись на поточному стані та інтуїтивно визначеній стратегії (політиці). Наслідки цієї дії є стохастичними і описуються функцією ймовірності переходу. Після переходу до нового стану агент отримує певну винагороду, пов’язану з цим станом. Процес прийняття рішень триває нескінченно [8].

Загалом, термін "Марковський" означає, що майбутній та попередній стани незалежні за умови знання поточного стану. Зокрема, для MDP значущо, що майбутні результати залежать лише від поточного стану та виконаної дії [8]. MDP дозволяє формалізувати послідовні рішення в умовах невизначеності та зміни середовища – властивостей, характерних для екстремального водіння.

1.5.1 Основні поняття: стани, дії, винагороди

В контексті автономного керування MDP описується як кортеж ⟨S, A, P, R, γ⟩ з 5 елементів, де:

- S (States) – стан може включати позицію транспортного засобу, швидкість, прискорення, стан дорожнього покриття, наявність перешкод, погодні умови тощо;
- A (Actions) – у випадку автомобіля це може бути поворот керма, гальмування, прискорення, зміни смуги;
- P (Transition Probability) – функція ймовірностей переходу, яка описує, з якою ймовірністю відбудеться перехід до нового стану s′ при виконанні дії a в стані s: P(s′ | s, a);
- R (Reward Function) – може охоплювати:
  - винагорода, тобто, підкріплення за безпечну поведінку (наприклад, уникнення зіткнень);
  - штрафи за перевищення швидкості, різке гальмування чи виїзд за межі смуги;
  - заохочення за ефективну траєкторію (мінімум витрат енергії або часу);
- γ (Gamma) – у реальному керуванні допомагає балансувати між короткостроковими та довгостроковими стратегіями.

1.5.2 Стратегії, цільові функції та оптимізація політик

У MDP агент формує стратегію (політику) – правило, яке визначає, яку дію обрати в кожному стані.

Політика (policy) π(s) – функція, яка зіставляє кожному стану s певну дію a або розподіл імовірностей дій [8].

Мета агента – максимізувати очікувану кумулятивну винагороду:

$$V^{\pi}(s) = \Xi\left[\sum_{t=0}^{\infty} \gamma^{t} R(s_t, a_t) | s_0 = s, \pi\right]$$

де V^{π}(s) – цільова функція вартості для політики π; γ – коефіцієнт дисконтування; $R(s_t, a_t)$ – винагорода на k-му кроці.

Існує кілька підходів до оптимізації політик.

Методи динамічного програмування. Value Iteration – ітеративно оновлює функцію вартості станів до досягнення збіжності [9];
Policy Iteration – чергує етапи оцінки політики та її покращення для знаходження оптимальної політики [9].

Моделі навчання з підкріпленням (Reinforcement Learning). Не потребують повного знання функцій P або R. Агент навчається на основі взаємодії з середовищем, наприклад Q-learning, де використовується модель без політики, яка навчається шляхом оновлення значень Q-функції для пар стан-дія [10]; чи Actor-Critic, де алгоритм поєднує політико-орієнтований (актор) та ціннісний (критик) підходи для ефективного навчання [11].

Методи оптимізації політик (Policy Gradient, PPO, TRPO). Напряму оптимізують параметризовану політику, ефективні при неперервних задачах, наприклад, таких, як автономне керування.

У разі мультиагентної постановки, кожен агент оптимізує власну політику з урахуванням впливу інших агентів, що може розглядатися в термінах ігрової теорії або мультиагентного навчання з підкріпленням (MARL).

1.6 Методи навчання з підкріпленням для задач автономного керування

Застосування методів навчання з підкріпленням є особливо актуальними для автономного керування ТЗ, оскільки дозволяють агенту навчатися прийняттю оптимальних рішень на основі взаємодії із середовищем. Дана методика тренувань є критично значущою в умовах високої динамічності, невизначеності чи наявності багатофакторних впливів, характерних для екстремальних дорожніх умов (наприклад, слизьке покриття, обмежена видимість, агресивна поведінка інших учасників руху). Ще більш ефективним з боку результатів та цікавим з боку тренування напрямом є мультиагентні RL-системи, в яких ми спрямовані на одночасне навчання декількох агентів, які, в свою чергу координують або ж адаптують свою поведінку відносно один одного в реальному часі.

1.6.1 DQN, PPO, A3C RL-алгоритми

Серед найбільш поширених алгоритмів RL, які застосовуються для задач автономного керування, слід виокремити представлені нижче.

- **Deep Q-Network (DQN)** – глибока нейромережева апроксимація функції Q, яка використовується для вибору оптимальної дії в кожному стані середовища. DQN підходить для середовищ з дискретними діями, наприклад, вибір між різними маневрами (обгін, гальмування, поворот). Однак у контексті екстремальних умов DQN може бути недостатньо гнучким, з огляду на обмежену здатність до узагальнення в безперервних просторах [16] (рис. 1.1).

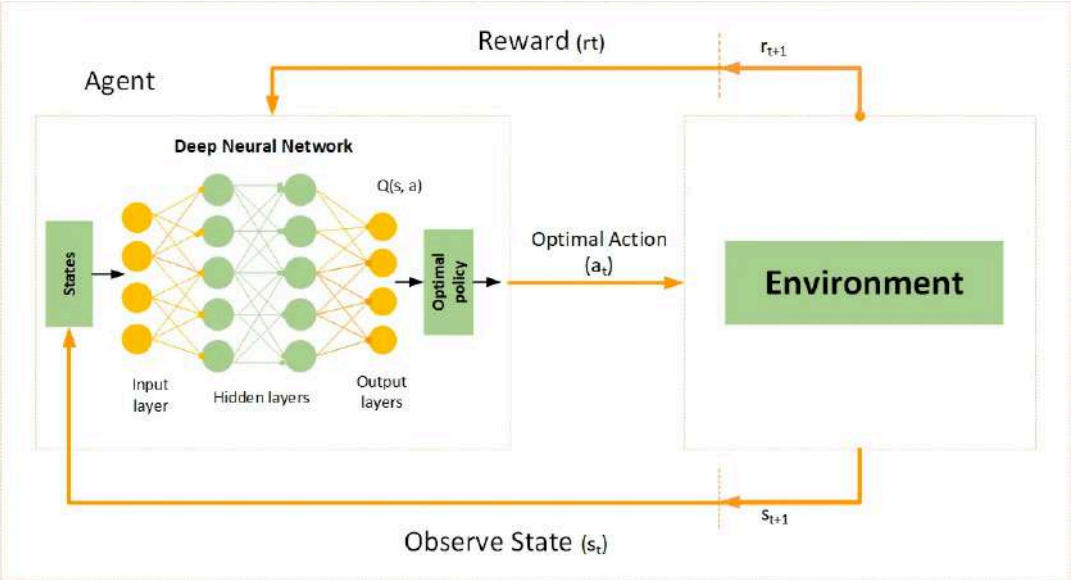


Рисунок 1.1 – Принцип роботи Deep Q-Network (взято з [16])

- **Proximal Policy Optimization (PPO)** – алгоритм політик, який забезпечує стабільне оновлення параметрів політики за рахунок обмеження амплітуди змін. PPO є одним з найпопулярніших методів у сучасному RL завдяки своїй простоті та ефективності в задачах з безперервними діями – що робить його ефективним для керування швидкістю, кутом повороту керма та гальмуванням [17] (рис. 1.2).

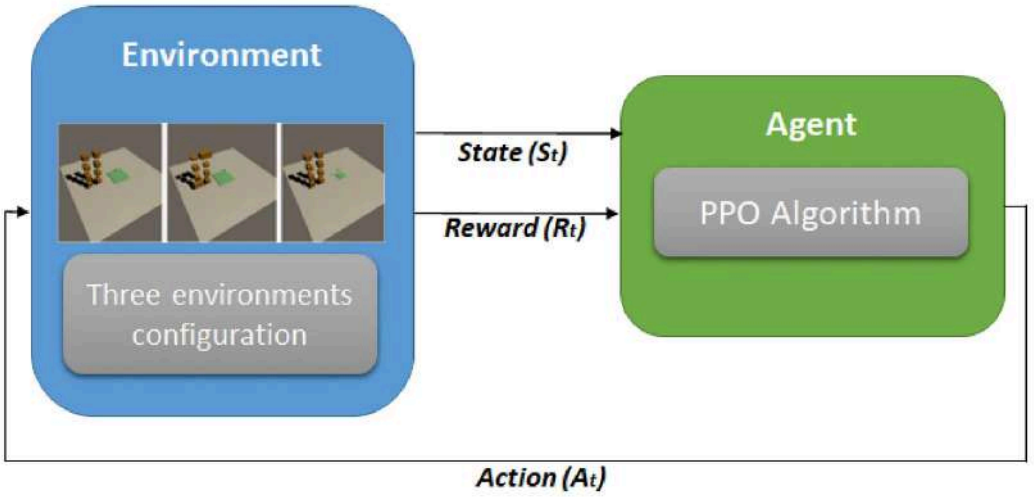


Рисунок 1.2 – Принцип роботи Proximal Policy Optimization [17]

- **Asynchronous Advantage Actor-Critic (A3C)** – алгоритм на основі актор-критик архітектури, де декілька агентів (потоків) паралельно досліджують середовище та оновлюють глобальну політику. Це дозволяє пришвидшити навчання та досягти більшої стійкості до нестабільностей середовища, особливо в мультиагентних сценаріях [18].

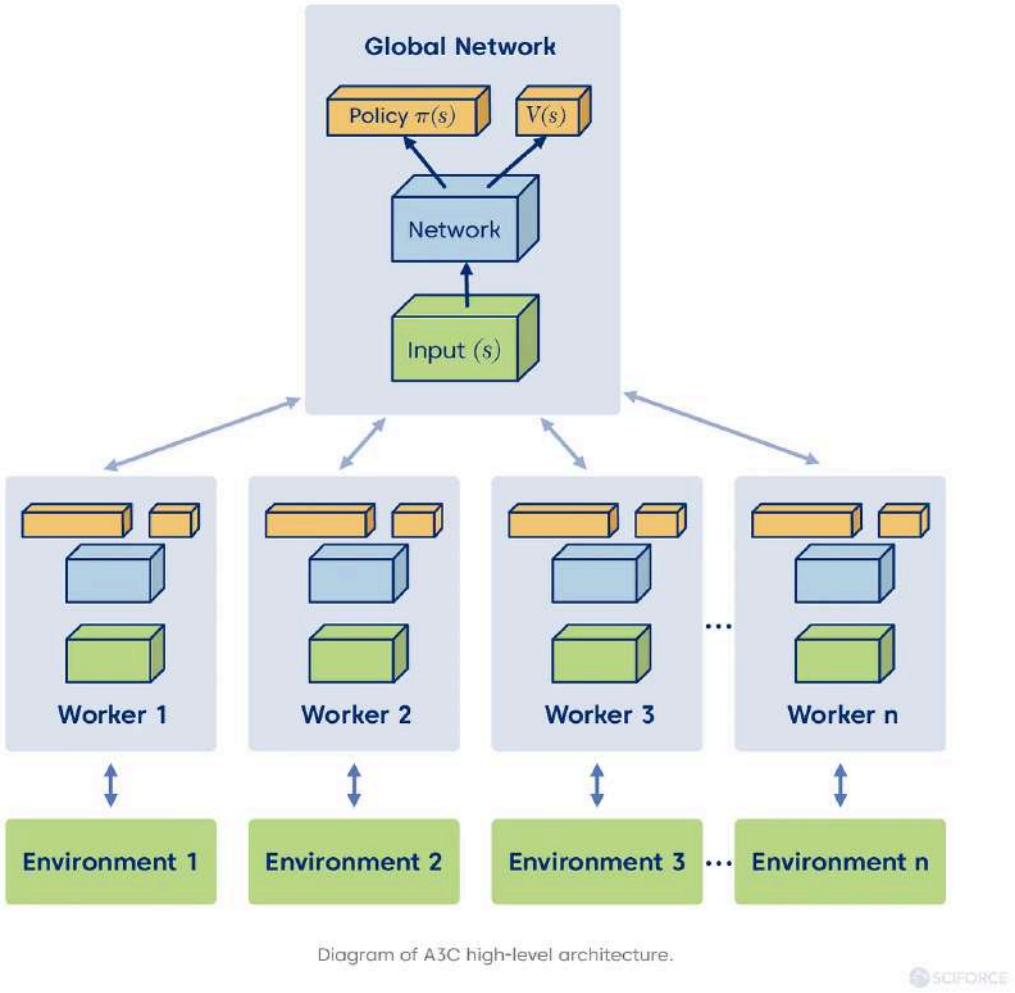


Рисунок 1.3 – Принцип роботи Asynchronous Advantage Actor-Critic [18]

У контексті мультиагентного керування важливо враховувати не лише ефективність окремого агента, але і його здатність до координації та адаптації у

взаємодії з іншими – що вимагає розширень базових алгоритмів, таких як Multi-Agent PPO (MAPPO) або QMIX.

1.6.2 Моделювання фізики та використання MPC в поєднанні з RL

Фізично ґрунтоване моделювання поведінки ТЗ є критично важливим компонентом для забезпечення реалістичності навчального середовища. Зокрема, model predictive control (MPC) – це класичний підхід до оптимального керування, що дозволяє передбачати траєкторію на горизонті планування і оптимізувати дії з урахуванням обмежень системи.

Поєднання RL з MPC дозволяє поєднати переваги моделювання на основі знань про динаміку (physics-informed control) із гнучкістю, яку забезпечує навчання з підкріпленням. Один з підходів – використовувати RL для модифікації або калібрування MPC в реальному часі. Такий гібридний підхід дозволяє домогтися високої адаптивності та безпечної поведінки, особливо в непередбачуваних ситуаціях, де модель MPC може бути неточною або неповною [19].

Перспективним та цікавим для дослідження є напрям ***Learning-based MPC***, в якому динаміка середовища апроксимується через нейронні мережі або Gaussian Processes, а управління виконується через MPC над цією апроксимацією. Це дає змогу розв’язувати задачі адаптивного планування в умовах невизначеності [20].

1.6.3 Методи мета-навчання та адаптації політик

Конфігурація сценаріїв з екстремальними або швидкозмінними умовами середовища малодієво поєднуються з класичним навчанням політик з нуля, оскільки даний метод виявляється повільним та неефективним. Тому значної уваги заслуговують методи мета-навчання (англ. meta-learning), які дають змогу агенту навчитися "навчатися" – тобто, адаптуватися до нових ситуацій з мінімальною кількістю взаємодій.

Одним із таких підходів є Model-Agnostic Meta-Learning (MAML), що дозволяє агенту оптимізувати початкову політику так, з мінімальною кількістю даних. Основна ідея полягає в тому, щоб навчити модель таким чином, щоб вона могла ефективно налаштовуватися на нові задачі за допомогою кількох кроків градієнтного спуску [21]. Основними принципами MAML є

- мета-навчання через оптимізацію початкових параметрів: замість навчання моделі для конкретної задачі, MAML навчає початкові параметри, які можуть бути швидко адаптовані до нових задач [21];
- двоетапний процес навчання:
  - Внутрішній цикл: для кожної задачі модель адаптується за допомогою кількох кроків градієнтного спуску на невеликому наборі даних [21];
  - зовнішній цикл: оновлюються початкові параметри моделі таким чином, щоб після адаптації у внутрішньому циклі модель показувала хороші результати на нових задачах в подальшому [21];
- агностичність до архітектури моделі: MAML не залежить від конкретної архітектури моделі і може бути застосований до різних типів моделей, таких як нейронні мережі, дерева рішень тощо [21];
- застосування в умовах обмежених даних: MAML особливо ефективний у сценаріях з обмеженою кількістю даних (few-shot

learning), де традиційні методи машинного навчання можуть бути неефективними [21].

Методи адаптації політик також включають такі методології, як RL with Online Adaptation – алгоритми, які оновлюють політику або її частину під час експлуатації в новому середовищі або ж Contextual Policy Embedding – підходи, де політика залежить від виведеного контексту середовища, що дозволяє універсальним агентам демонструвати адаптивну поведінку без необхідності повного перенавчання.

Дані методи дозволяють створювати агентів, здатних до швидкої переадаптації, що є ключовим у мультиагентних системах в умовах екстремальної невизначеності, обмеженого зв'язку або непередбачуваних взаємодій.

1.7 Самокероване навчання агентів у складних умовах

Self-supervised RL (SSRL) - підхід, який дозволяє агентам самостійно генерувати сигнали для навчання, поєднуючи переваги класичного навчання з підкріпленням (англ. RL) і самокерованого навчання (англ. self-supervised learning). Такий підхід є особливо актуальним у випадках, коли зовнішня винагорода є рідкісною, шумною або недоступною [22].

Основною ідеєю SSRL є те, що, агенти створюють власні допоміжні завдання (англ. auxiliary tasks) для навчання репрезентацій середовища, ці завдання не вимагають маркованих даних і можуть включати в себе такі як: передбачення наступного стану (англ. forward dynamics prediction), реконструкцію сенсорних спостережень (англ. autoencoding), контрастивне навчання, що полягає в навчанні агента розрізняти схожі та відмінні стани та цільове посилення інформаційної насиченості представлень (maximize information gain), тобто кількість нової інформації, яку агент отримує про

динаміку середовища в результаті певної дії, вона вимірюється як зменшення невизначеності (ентропії) щодо стану середовища після виконання дії, цей підхід дозволяє агенту ефективніше досліджувати середовище, спрямовуючи увагу на ті дії, які найбільше зменшують невизначеність у його змодельованому середовищі [23].

Основними методами в SSRL є:

– Contrastive Learning (CURL, CPC):
  - CURL (Contrastive Unsupervised Representations for RL) – агент навчається розрізняти подібні й різні стани (або пари: стан/дія/наступний стан), щоб сформувати більш інформативну латентну репрезентацію [24]; CURL дає змогумоделювати зміни в навколишньому середовищі, наприклад, розпізнавати ситуації гальмування, зміни покриття дороги чи появу об'єктів, без явного маркування;
  - CPC (Contrastive Predictive Coding) – передбачення майбутніх спостережень у латентному просторі з використанням контрастивної втрати [25];
– Forward/Inverse Dynamics Models; такі моделі дають змогу агенту навчатися з досвіду, навіть коли винагорода недоступна; наприклад, автономне авто може «вивчити», які дії ведуть до стабілізації на слизькій дорозі:
  - Forward model: передбачення наступного стану на основі поточного стану та дії;
  - Inverse model: передбачення дії, яка призвела до переходу між станами;
– Data Augmentation for RL:
  - Техніки на зразок RAD (Reinforcement Learning with Augmented Data) або DrQ (Data-Regularized Q) дають змогу використовувати трансформовані зображення для покращення

узагальнення політики; метод є ефективним у симуляціях, що передбачають перенесення агентів у реальні умови.

У задачах керування транспортом в екстремальних умовах SSRL є ефективним та корисним через такі фактори, як недоступність винагороди у складних дорожніх умовах (туман, дощ, обрив сигналу GPS), агент може використовувати внутрішні цілі для продовження навчання та адаптації; імітація небезпечних сценаріїв: SSRL дозволяє навчатись на змодельованих аварійних ситуаціях без реального ризику; перенесення навчених політик: завдяки покращеним репрезентаціям SSRL-підходи краще узагальнюють політику на нові умови; швидка адаптація до змін середовища: наприклад, самостійна адаптація до нового дорожнього покриття, зміни ваги ТЗ (пасажири/вантаж), або збій датчиків.

1.8 Застосування генеративного штучного інтелекту для генерації складних сценаріїв

Для створення реалістичних та варіативних сценаріїв для навчання та тестування автономних транспортних засобів в екстремальних умовах ключову роль відіграє генеративний ШІ, який надає змогу моделювати рідкісні та критичні ситуації, які складно або небезпечно відтворити в реальному світі, забезпечуючи таким чином більш ефективне навчання агентів.

Основними підходами до генерації сценаріїв є генерація сценаріїв на основі даних (англ. Data-driven Scenario Generation): моделі, такі як TrafficGen [26], використовують реальні дані про дорожній рух для навчання генеративних моделей, які можуть створювати нові сценарії, що відображають реальні умови дорожнього руху (рис. 1.4). Це дозволяє створювати сценарії, які є як реалістичними, так і різноманітними, що сприяє кращому узагальненню навчених політик; генерація сценаріїв на основі моделювання поведінки (англ.

Behavior-based Scenario Generation): підходи, такі як HAD-Gen [27] (рис. 1.5), моделюють різні стилі водіння, включаючи агресивну, обережну та нормальну поведінку, для створення сценаріїв, що охоплюють широкий спектр можливих ситуацій на дорозі, що дозволяє агентам навчатися реагувати на різні типи поведінки інших учасників дорожнього руху; генерація критичних сценаріїв за допомогою навчання з підкріпленням (англ. RL-based Critical Scenario Generation): методи, що поєднують генеративні моделі з навчанням з підкріпленням, дозволяють створювати сценарії, які є як реалістичними, так і викликають помилки в системах автономного керування. Це особливо корисно для виявлення слабких місць у політиках агентів [28].

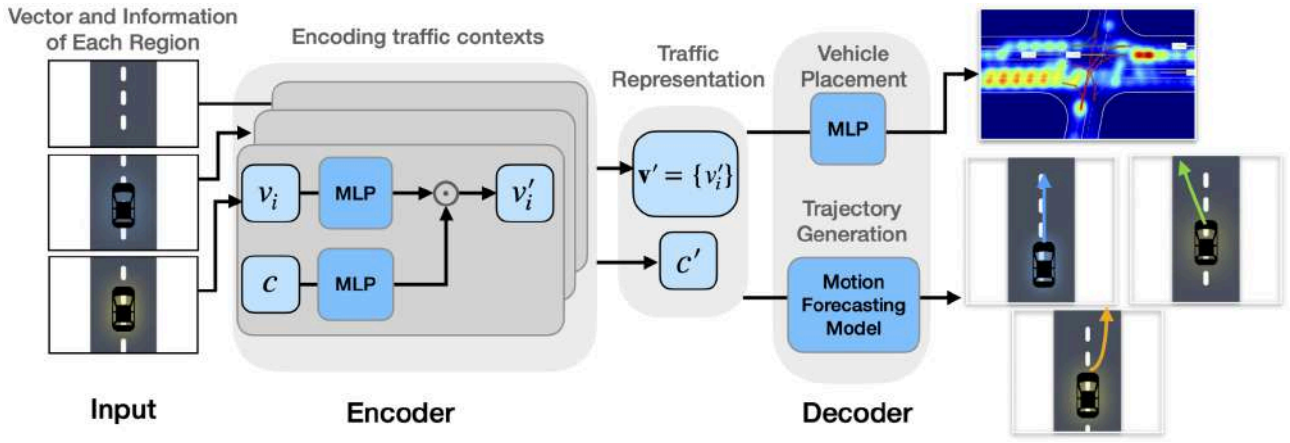


Рисунок 1.4 – TrafficGen для генерації складних сценаріїв у контексті середовища для дорожнього руху [26]

Використання описаних методологій покриває малоймовірні та небезпечні ситуації, урізноманітнює дані та авжеж стабілізує систему і підвищує її надійність.

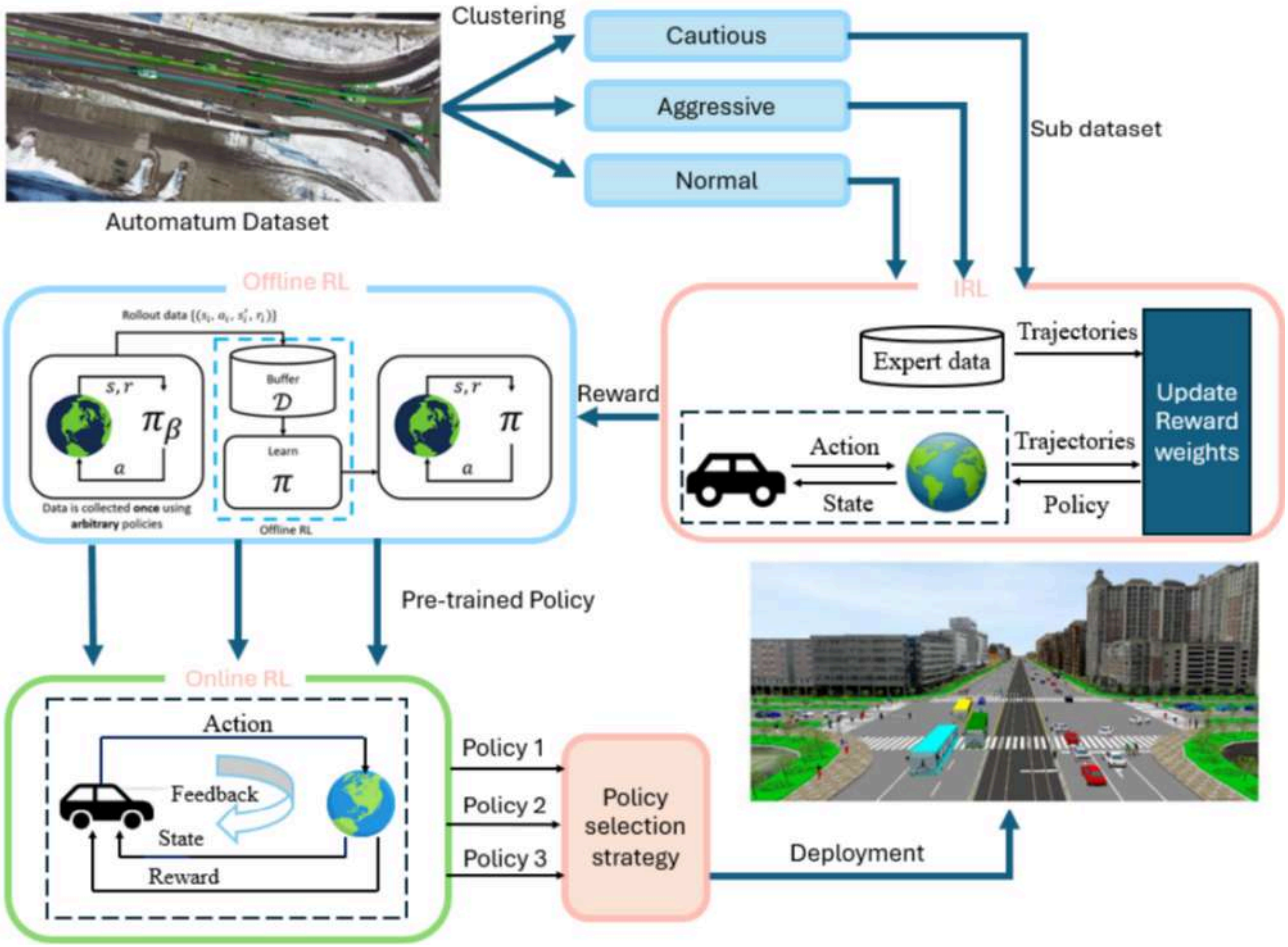


Рисунок 1.5 – HAD-Gen для моделювання поведінки інших учасників дорожнього руху [27]

1.9 Федеративне навчання та розподілене тренування агентів

Федеративне навчання – це передова парадигма машинного навчання, яка полегшує навчання моделей ШІ через мережу децентралізованих пристроїв або серверів. Цей підхід дозволяє кожному вузлу робити свій внесок у процес навчання моделі, використовуючи свої локальні дані, без необхідності обмінюватися або централізувати ці зразки даних. Цей метод уможливлює спільне навчання, зберігаючи при цьому вихідні дані в їхньому джерелі,

вирішуючи проблеми конфіденційності та безпеки. Його реалізація передбачає три ключові етапи: ініціалізацію, локальне навчання та агрегування оновлень [29].

Можемо розглянути сценарій, де автономні ТЗ, розташовані в різних регіонах або умовах (ожеледиця, піщана буря, туман і т.д.), навчаються локально на власному досвіді руху в цих специфічних середовищах. Кожен ТЗ тренує власну локальну модель на основі своїх сенсорних даних, таких як інформація з камер, лідарів або IMU (англ. inertial measurement unit), не передаючи ці дані за межі пристрою, що важливо для забезпечення безпеки та захисту приватності.

Після завершення локального навчання кожен агент надсилає на центральний сервер лише оновлені параметри моделі. Наприклад, ТЗ, що рухається в умовах сильного дощу, адаптує свою модель до зниження видимості та змін у зчепленні з дорогою, а ТЗ на бездоріжжі – до частих коливань і нестабільності та всі інші – за аналогією. Сервер, отримавши оновлення від усіх агентів, агрегує їх – зазвичай шляхом середнього зважування – формуючи нову глобальну модель, яка враховує досвід усіх учасників. Потім оновлена модель повертається на всі ТЗ, покращуючи здатність кожного з них діяти в ширшому спектрі екстремальних умов.

Таким чином, кожен агент отримує вигоду від колективного досвіду інших, не розкриваючи при цьому власних сирих даних (рис. 1.6). Це дозволяє ефективно тренувати адаптивні політики поведінки, здатні забезпечити безпечне та ефективне керування в умовах часткової невизначеності, обмеженого зв’язку або різноманітних фізичних загроз.

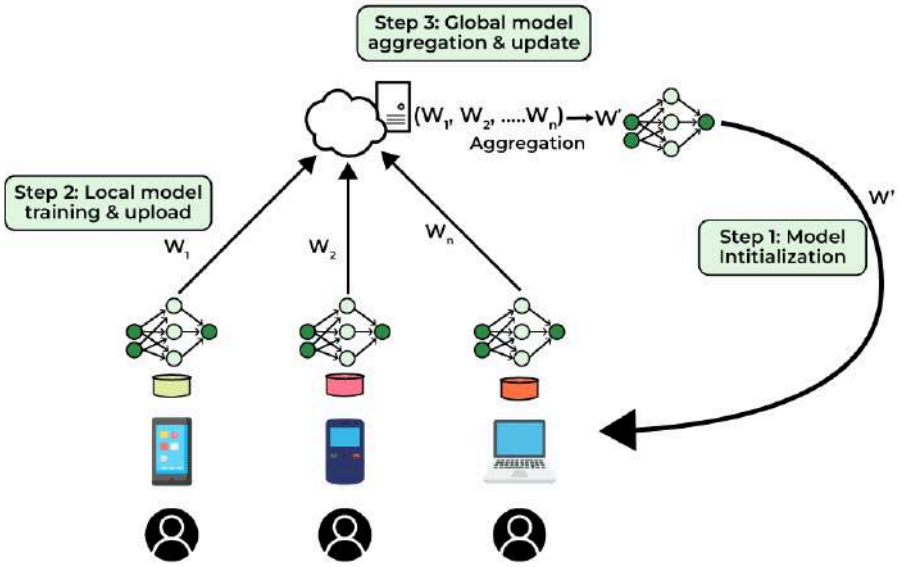


Рисунок 1.6 – Розподілене тренування агентів [29]

Розподілене тренування агентів – це методологія навчання, за якої кілька агентів паралельно досліджують середовище, кожен у своїх умовах, та спільно вдосконалюють загальну політику керування ТЗ, цей підхід є особливо цінним для мультиагентних систем у транспортній сфері, де різні ТЗ можуть працювати в кардинально відмінних умовах таких, як уночі та вдень, у місті та за його межами, на сухому асфальті чи в умовах ожеледиці.

Кожен агент в такій системі виконує локальне навчання, що базується на навчанні з підкріпленням, із врахуванням лише тієї інформації, яку він отримує зі свого локального середовища. Наприклад, один транспортний засіб може тренувати модель ухилення від перешкод у середовищі з високою динамікою (міський трафік), тоді як інший адаптує модель до умов низької видимості (туман, ніч). Такі агенти збирають цінні дані, оновлюють свої локальні моделі і надсилають їх або централізованому вузлу (як у федеративному навчанні), або обмінюються оновленнями безпосередньо між собою (peer-to-peer), формуючи більш універсальну й адаптивну модель керування.

Перевага розподіленого підходу в тому, що тренування охоплює ширший спектр сценаріїв, ніж це можливо в рамках одного агента або симуляції. Завдяки

обміну досвідом, система швидше узагальнює політики, що працюють в багатьох ситуаціях. З технічної точки зору, такий підхід часто базується на асинхронному оновленні параметрів або системах колективного навчання, де агенти використовують спільну пам'ять (shared replay buffer), яка накопичує досвід з різних джерел.

Висновки до розділу 1

У першому розділі було всебічно досліджено сучасні підходи та технології, що лежать в основі автономного керування транспортними засобами в екстремальних умовах. Продемонстровано, що задачі автономного водіння є актуальними в умовах швидкої урбанізації, зростання кількості транспортних засобів і потреби в підвищенні безпеки дорожнього руху. Особлива увага була приділена для висвітлення та дослідження проблеми складності середовищ, у яких функціонують такі системи, включно з погодними аномаліями, низькою видимістю, непередбачуваними діями інших учасників руху та обмеженим зв’язком.

Проаналізовано мультиагентні системи як природну парадигму для реалізації колективної поведінки транспортних засобів, особливо в умовах, коли централізоване керування є неможливим або ризикованим. Сформульовано задачу керування як марковський процес прийняття рішень, що дозволяє формально описувати динаміку взаємодії агента із середовищем, а також розглянуто RL-алгоритми як базові інструменти для навчання стратегій прийняття рішень.

Окрему увагу приділено сучасним методам підвищення ефективності навчання: моделюванню фізики для попередньої симуляції, мета-навчанню для адаптації до нових умов, а також самокерованому навчанню для створення внутрішніх цільових функцій у відсутності явної винагороди. У розрізі цього

підходу розглянуто генеративний ШІ як засіб побудови критичних сценаріїв, які складно або небезпечно отримати в реальному середовищі. Також проаналізовано роль федеративного навчання та розподіленого тренування, що дозволяють агентам навчатись колективно без обміну сирими даними, забезпечуючи як масштабованість, так і захист приватності.

Таким чином, сформовано цілісну картину сучасного стану досліджень і технологій, що дозволяє розробити та обґрунтувати ефективний мультиагентний підхід до автономного керування транспортом в екстремальних умовах з використанням навчання з підкріпленням, генеративного ШІ та самокерованого навчання.

РОЗДІЛ 2 МАТЕМАТИЧНІ ТА МЕТОДИЧНІ ОСНОВИ РОБОТИ

2.1 Архітектура мультиагентної системи

2.1.1 Основи мультиагентної взаємодії в системах автономного керування

Системи автономного керування або ж складні динамічні середовище мають достатньо незвичні зв’язки взаємодій, основну роль у їхній роботі відіграє здатність агентів до координації та взаємодії. Мультиагентні системи передбачають наявність декількох агентів, що існують в одному середовищі, але мають різні цілі, можливості, інформаційне забезпечення (сенсори, тощо), в контексті дорожнього руху ми можемо виокремити як ТЗ, так пішоходів, керуючі дрони або ж штурмани: к віртуальні, так цілком реальні. Основними викликами в роботі з мультиагентими системами є не досягнення цілей сепаровано агентами, а у більшій мірі врахування взаємодій та окремих поведінкових особливостей інших учасників руху задля уникнення конфліктів та підвищення ефективності виконання роботи.

Ми можемо використовувати різні підходи для координації процесів у середовищі, до прикладу: *централізоване керування* – один контролер координує дії всіх агентів (найкраще підходить для задач моделювання трафіку або якоїсь колективної поведінки) [30]; *децентралізоване керування* – кожен агент вирішує, ґрунтуючись лише на локальній інформації (близький до реальних умов автономного транспорту) [31]; *кооперативне навчання з підкріпленням (cooperative MARL)* – агенти навчаються узгодженим діям через обмін інформацією [32]; *конкурентне навчання (competitive MARL)* – агенти діють з протилежними цілями, загальна нагорода поділяється мій ними (взаємодія з агресивними або непередбачуваними учасниками дорожнього руху) [32].

У межах даного дослідження мультиагентна взаємодія розглядається у двох ключових сценаріях.

1. Агент + середовище з іншими динамічними об'єктами (транспорт/перешкоди) – використання RL з розширеною функцією винагороди, що враховує маневрування.
2. Агент + віртуальний “штурман” – приклад кооперативного MARL, де другий агент діє як помічник, що передбачає динаміку або ризики, доповнюючи обмежене сприйняття основного агента.

У середовищі симулятора Isaac Sim мультиагентна система дозволяє моделювати як одночасне існування кількох агентів із різною поведінкою, так і ієрархічні структури взаємодії. Це забезпечує гнучкість у тестуванні як ізольованих стратегій, так і складної взаємодії в умовах схожого на реалістичний трафіку, змінного середовища та непередбачуваних поведінкових та зовнішніх сценаріїв.

2.1.2 Побудова симуляційного середовища в Isaac Sim

Для моделювання середовища та виконання симуляції ми будемо використовувати симулятор від NVIDIA – Isaac Sim. Це високоточний симулятор, спеціально розроблений для моделювання робототехнічних і автономних систем з урахуванням фізично дійсної та достовірної динаміки, мультисенсорного середовища та взаємодії з іншими агентами, інтеграція з Omniverse дозволяє йому підтримувати складно полігональні 3D-сцени, фізику справжнього світу (ковзання, тертя, зіткнення, інерція, маса, сила тяги, підвіска, можливість моделювання нестандартних зовнішніх фізичних умов), поміщену у надбудову NVIDIA PhysX, мультисенсорні симуляції (можливість додавати RGB/Depth/Semantic камери, LiDAR, IMU та точно імітувати дійсні сигнали з них), сценарії з множинними агентами та інструменти для навчання з підкріпленням. Isaac Sim має інтеграцію з Isaac Lab (розширення Isaac Gym) – середовищем для масштабованого навчання з підкріпленням, яке має в рази

прискорене тренування агентів порівняно з іншими симуляторами та дозволяє запускати використовувати більше пам’яті та запускати паралельно десятки або й сотні паралельних епізодів, підтримує популярні RL-алгоритми та має зручні інтеграції для розробки з використанням PyTorch та поширених RL-бібліотек. Також Isaac Sim дозволяє реалізовувати curriculum learning – концепту тренувань, що представляє собою чергування тренувальних даних та збільшення складності процесу тренування.

Вибір 3D моделі транспортного засобу та підготовка моделі до подальшої експлуатації у симуляторі. На першому етапі формування симуляційного середовища необхідно обрати та налаштувати 3D-модель транспортного засобу. У рамках цього проєкту використовується модель автомобіля з підтримкою диференційованої фізики, включно з підвіскою, масовим розподілом, шинами та моментами інерції. Для реалізації було обрано опенсорсну [34] 3D-модель автомобіля типу седан. Для попередньої підготовки до руху та взаємодії із середовищем модель необхідно адаптувати до роботи в динамічному середовищі, а саме: в першу чергу призначити властивості тіла з жорсткою динамікою (англ. rigid body) для того, щоб сам ТЗ мав власну фізику та міг взаємодіяти із середовищем, налаштування фізичних з'єднань коліс за допомогою фізичних з’єднань (англ. joint) та визначити коефіцієнти зчеплення для кожної із типу поверхні, це зводить задачу до приведення моделі у вид Isaac Vehicle – API, що забезпечує доступ до API контролю швидкості та повороту, сумісність із сенсорами, підтримку вбудованих контролерів та ефективну взаємодію із самою моделлю (рис. 2.1 і 2.2).

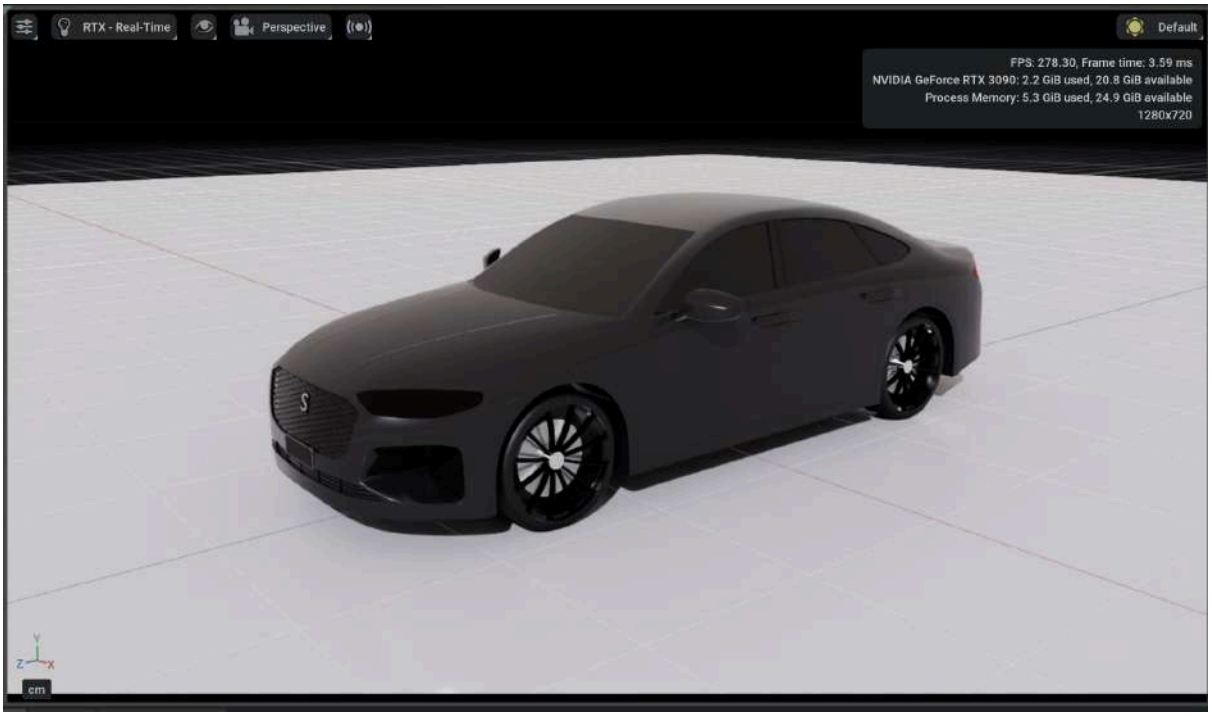


Рисунок 2.1 – Обрана USD-модель в ISAAC SIM

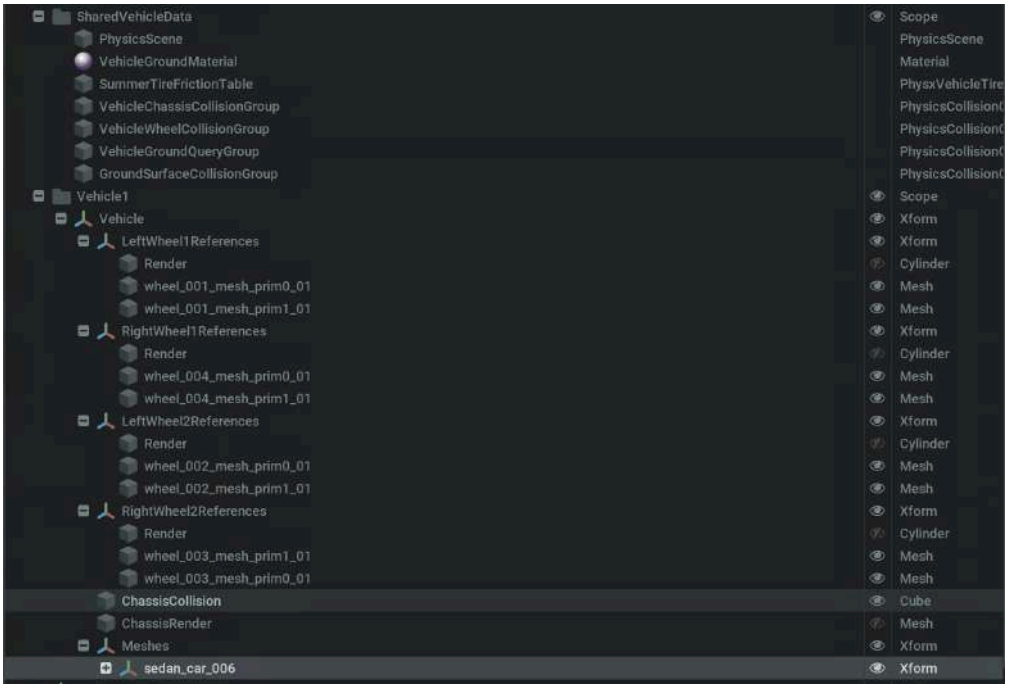


Рисунок 2.2 – Структура елементів для використання API Isaac Vehicle

Визначена маса автомобіля – 1500 кг – стандартна маса для ТЗ даного виду, передній привід – керуючий момент прикладається лише до передньої осі, матеріал коліс – гума, статичний коефіцієнт тертя матеріалу колес – 0.85 на сухому асфальті, динамічний – 0.75.

2.1.3 Інтеграція фізично достовірної динаміки транспортного засобу в Isaac Gym

Реалістична симуляція динаміки транспортного засобу є ключовою умовою для навчання політик управління, які згодом мають бути пройти етап перенесення та інтеграції simtoreal. Isaac Gym, як частина програмного стека NVIDIA Omniverse, забезпечує можливість реалізації такої динаміки завдяки фізичному рушію PhysX та оптимізації для масово паралельного навчання в Reinforcement Learning.

Динаміка руху моделюється через рівняння представлені нижче.

Сили тяги:

$$F_{\text{тяги}} = T - R - F_{\text{повітря}} - F_{\text{зчеплення}},$$

де T – рушійна сила, R – опір коченню, $F_{\text{повітря}}$ – опір повітрю, $F_{\text{зчеплення}}$ – тертя зчеплення.

Момент обертання кузова:

$$M = I\,\alpha,$$

де I – момент інерції, α – кутове прискорення.

Важливою особливістю є можливість динамічно змінювати коефіцієнти зчеплення для різних типів поверхні (лід, пісок, асфальт). Це реалізується через створення PhysX Material Tags і налаштування матриці тертя в фізичному рушії. Всі фізичні параметри інтегруються у середовище тренування в Isaac Gym через механізм EnvironmentConfig. Це дозволяє створювати сотні одночасних екземплярів середовищ для паралельного навчання, з однаковими або змінними фізичними параметрами. Завдяки цьому стає можливим реалізувати curriculum learning або meta-RL для адаптації до нових дорожніх умов.

Таблиця 2.1 – Властивості матеріалів поверхні

Поверхня	*Статичний коеф. тертя (μs)*	*Динамічний коеф. тертя (μk)*	*Коеф. відскоку (Restitution)*
Сухий асфальт	0.90	0.80	0.05
Мокрий асфальт	0.60	0.50	0.10
Сніг	0.30	0.20	0.00
Лід	0.10	0.05	0.00
Ґрунт/Бруд	0.50	0.45	0.05
Пісок	0.40	0.35	0.02

2.1.4 Використання сенсорів: камери, LiDAR, IMU – структура та обробка даних

Для розуміння розташування об’єктів, розмітки та точнішого миттєвого реагування на надзвичайні ситуації, коректної оцінки стану автомобіля та його локалізацій при тренуваннях (відповідно до конфігурацій) будемо використовувати віртуальні сенсори, змодельовані у середовищі Isaac Sim:

камери, лідар (LiDAR) та інерціальний модуль (IMU). *RGB-камера* – використовується для отримання зображень сцени; *semantic camera* – забезпечує сегментаційні мапи для задач комп’ютерного зору (напр. lane detection); *depth camera* – формує карту глибини кожного пікселя; *LiDAR* – генерує 3D точкову хмару (англ. point cloud), що точно відображає відстані до об’єктів, використовується для створення карти оточення, виявлення статичних та динамічних об'єктів, формування вхідних даних до модулів планування траєкторії; *інерціальний модуль* (англ. IMU), що забезпечує оцінку стану ТЗ у просторі та використовується для предиктивного контролю, сама функціональність полягає у вимірі прискорення, кутових швидкостей, орієнтації (рис. 2.3 і 2.4).

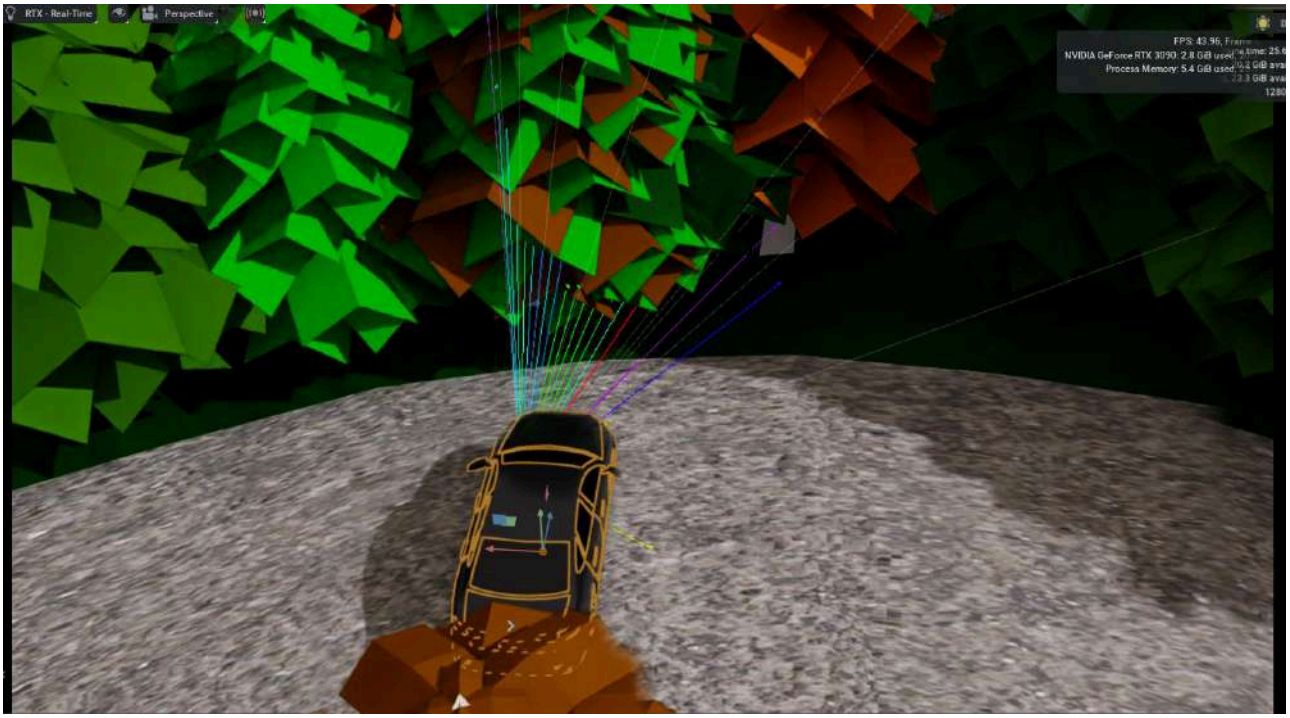


Рисунок 2.3 – Вигляд роботи LiDAR

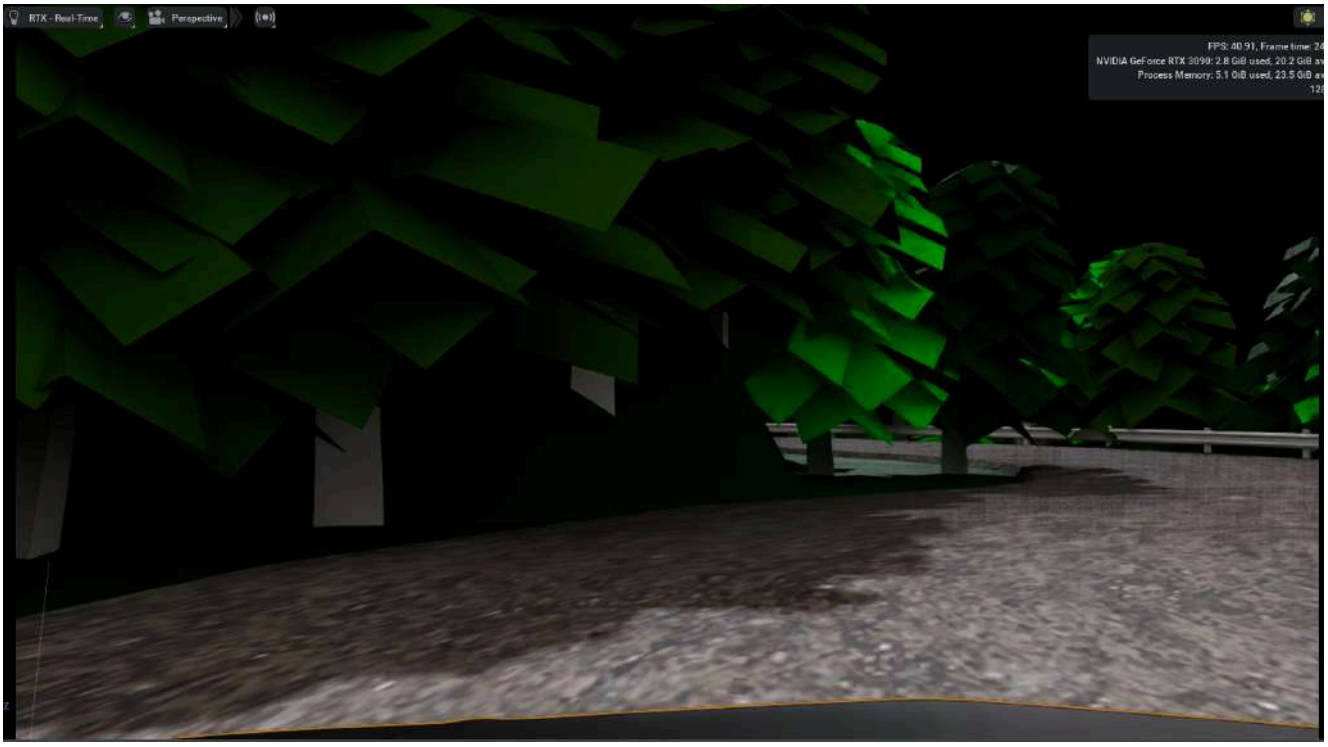


Рисунок 2.4 – Вигляд роботи RGB-камери

2.2 Формалізація задачі як задачі навчання з підкріпленням

Для реалізації інтелектуального керування транспортним засобом у віртуальному середовищі із високою складністю, задача формалізується як задача навчання з підкріпленням (англ. Reinforcement Learning). У такій постановці агент (транспортний засіб) навчається ухвалювати рішення, які максимізують очікувану нагороду, взаємодіючи з навколишнім середовищем, тобто симуляцією.

2.2.1 Математичне формулювання задачі як марковського процесу прийняття рішень (MDP)

Для навчання агента (транспортного засобу) у симуляційному середовищі з використанням методів штучного інтелекту, задача формалізується у вигляді марковського процесу прийняття рішень (англ. Markov Decision Process, далі MDP). Така формалізація дозволяє використовувати методи навчання з підкріпленням, які оптимізують поведінку агента на основі винагороди за дії в середовищі.

У симуляції Isaac Sim переходи реалізуються за допомогою фізичного рушія (PhysX), що забезпечує фізично достовірну еволюцію системи.
Раніше було наведено визначення з основними концепціями марковського процесу прийняття рішень.

Можемо зазначити, що менше значення γ (коефіцієнту дисконтування) надає перевагу короткостроковим винагородам, тоді як значення, близьке до 1, стимулює стратегічне довгострокове планування.

Метою агента є знайти таку ***стохастичну політику*** $\pi(at|st)$, яка максимізує очікувану сумарну винагороду:

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \right]$$

Тобто оптимальна політика:

$$\pi^* = \arg\max_\pi J(\pi)$$

Процес є марківським, якщо перехід у наступний стан s_{t+1} залежить лише від поточного стану s_t і дії a_t, але не від попередніх станів:

2.2.2 Визначення простору станів, дій та функції винагороди

2.2.2.1 Проєктування станів і дій для базового керування (PPO/DQN)

Метою даного проєктування є навчити транспортний засіб основам автономного руху шляхом формулювання відповідного простору станів і дій у контексті навчання з підкріпленням.

Стан описує положення і динаміку агента у відносному просторі відносно траєкторії або цілі. З огляду на мінімально необхідну кількість параметрів, стан можна подати у вигляді вектору ознак (табл. 2.2).

Таблиця 2.2 – Параметри стану тренування

Назва	*Позначення*	*Опис*
Позиція	(x, y)	Поточні координати транспортного засобу в глобальній або локальній системі координат
Швидкість	v	Лінійна швидкість уздовж напрямку руху (може бути скаляр або вектор)
Орієнтація	θ	Кутова орієнтація транспортного засобу (радіани)
Відхилення від центру	e_y	Відстань до центра смуги/траєкторії
Кутова помилка	e_θ	Різниця між орієнтацією авто та напрямком траєкторії
Таргет дельта	d_t	Евклідова відстань до цільової позиції

Примітка: всі значення бути нормалізовані перед подачею до нейромережі.

Вибір простору дій залежить від кінематики транспортного засобу. Для базового контролювання ми обираємо диференційоване керування (табл. 2.3):

$$a = [steering, throttle].$$

Таблиця 2.3 – Простір дій для тренування

Компонента	*Діапазон*	*Опис*
steering	[−1, 1]	Кут повороту керма, де −1 – максимальний лівий поворот, 1 – максимальний правий
throttle	[-1, 1]	Сигнал тяги: >0 – газ, <0 – гальмування, 0 – вільний рух

Вхідний вектор стану подається на вхід актору та критику PPO-агента, який на його основі обирає дію. Дія змінює стан середовища через динамічну модель, що базується на фізиці транспортного засобу. Отриманий новий стан дозволяє обчислити винагороду, на основі якої коригується політика.

Вхідний стан: тензор розмірності R^n, де n – кількість ознак (у нашому випадку n = 7).

Дія: тензор розмірності R^2, що моделює параметри steering та throttle.

Актор: ймовірнісна модель з нормальним розподілом $\mathcal{N}$(loc, scale).

Критик: скалярна оцінка цінності стану V(s).

Визначимо стандартні параметри для логування у W&B (табл. 2.4).

Таблиця 2.4 – Параметри для логування

Назва метрики	*Позначення*	*Опис*
Загальний Reward	episode_reward	Сумарна винагорода за епізод
Кількість кроків	episode_length	Скільки кроків виконав агент
Помилка по позиції	mean_d_t	Середнє значення відстані до цілі
Помилка від центра	mean_e_y	Середнє відхилення від центру смуги
Помилка по орієнтації	mean_e_θ	Середня кутова різниця з напрямком руху
Policy Loss	policy_loss	Втрати політики з PPO
Value Loss	value_loss	Втрати критика
Entropy	entropy	Ентропія політики (чим вище – тим більш дослідницька поведінка)
Explained Variance	explained_var	Наскільки добре value function передбачає винагороду
Grad Norm Actor/Critic	actor_grad_norm / critic_grad_norm	Норма градієнтів – для діагностики навчання

2.2.2.2 Проєктування винагород для стабільного слідування траєкторії

Метою функції винагород є стимулювання агента до плавного, безпечного та стабільного руху по заданій траєкторії, дотримуючись центра смуги та правильного напрямку. Формування винагороди відбувається як зважена сума декількох складових (табл. 2.5).

Таблиця 2.5 – Штрафи та бонуси тренування

Назва	*Позначення*	*Тип*	*Опис*
Відстань до цілі	d_t	штраф	Заохочення за зменшення відстані до наступної траєкторної точки
Відхилення від центра смуги	e_y	штраф	Штраф за відхилення від центру дороги
Кутова помилка	e_θ	штраф	Штраф за різницю між орієнтацією авто та напрямком траєкторії
Швидкість	v	бонус	Бонус за рух з "оптимальною" швидкістю
Виїзд за межі	*out_of_bounds*	великий штраф	Великий штраф за виїзд за межі допустимої області
Досягнення цілі	*goal_reached*	великий бонус	Бонус за досягнення фінальної цільової точки

Формула загальної винагороди буде мати вигляд:

$$reward = (- w_d * d_t - w_e_y * |e_y| - \\ - w_e_thet * |e_theta| + \\ + w_v * v - w_oob * is_out_of_bounds + w_goal * goal_reached) \quad (3.1)$$

Коефіцієнти, які були використані, представлені у табл. 2.6.

Для забезпечення надійного та прискореного навчання агент повинен мати можливість перезапускати епізод у разі, якщо він потрапляє у нестабільну ситуацію або затягує розв’язання задачі. Це дозволяє уникати даремного витрачання обчислювальних ресурсів і прискорює процес оптимізації політики.

Таблиця 2.6 – Коефіцієнти для розрахунку функції винагороди

Компонент	*Позначення*	*Значення*
Відстань до цілі	*w_d*	1.0
Відхилення від центру	*w_e_y*	2.0
Кутова помилка	*w_e_theta*	1.5
Швидкість	*w_v*	0.2
Виїзд за межі	*w_oob*	50.0
Досягнення цілі	*w_goal*	100.0

Умовами для дострокового завершення епізоду є:

Таблиця 2.7 – Умови дострокового термінування епізоду

Умова	*Позначення / перевірка*	*Опис*
Виїзд за межі карти	*is_out_of_bounds*	Якщо координати виходять за межі допустимого поля
Перевищено тривалість	*step_count > max_steps*	Обмеження максимального числа кроків для одного епізоду
Перевернене або зупинене авто	*unstable_pose_detected або v ≈ 0*	Якщо агент перевернувся, застряг або зупинився надовго
Занадто велике відхилення	*abs(e_y) > lane_width / 2*	Агент повністю виїхав зі смуги

2.2.2.3 Розширення простору станів у задачі маневрування (Imitation Learning + RL)

На етапі маневрування, коли агент має не лише їхати по прямій, а й виконувати складніші задачі (наприклад, об'їзд перешкод, повороти, зміна смуги), базовий простір станів (позиція, орієнтація, швидкість) вже є недостатнім. Для підвищення якісності прийняття рішень необхідно розширити стан агента додатковими сенсорними даними та контекстом середовища.

Метою агента буде:

- досягнення цілі *goal_position*, рухаючись від стартової позиції;
- ухилення від перешкод на основі лідарних даних;
- корекція траєкторії згідно з відхиленнями e_y, e_θ;
- оптимізація через навчання з підкріпленням після попереднього імітаційного навчання.

Для розуміння навколишнього середовища на цьому етапі будемо використовувати лідар 360 для нормального розуміння відстаней до будь-якого роду фізичних об’єктів, тобто до попередньої конфігурації додається змінна, представлена у табл. 2.8.

Таблиця 2.8 – Змінні, що додаються в простір станів

Назва змінної	*Позначення*	*Опис*
Відстані з лідару	*lidar_distances*	Вектор нормалізованих відстаней до об’єктів (64 промені)

Простір дій лишаємо без будь-яких змін.

Поведінка агента. Навчання:

- Behavior Cloning (BC):
  - тренування на демонстраціях (експерт або PID);

    - імітація дій за заданими станами: observation → action;
    - 200–500 епізодів для початкового старту політики;
- Reinforcement Learning (PPO):
    - початкові ваги ініціалізовані з BC;
    - політика вдосконалюється через нагороди:
        - досягнення цілі (+);
        - уникнення перешкод (+);
        - відхилення або колізія (-).

Таблиця 2.9 – Метрики для логування даної конфігурації

Назва метрики	*Позначення / ключ*	*Опис*
Нагорода за епізод	episode_reward	Сума всіх нагород протягом одного епізоду
Середня нагорода	avg_reward	Середнє значення episode_reward за останні N епізодів
Кількість кроків	episode_length	Тривалість епізоду в кількості кроків
Кількість зіткнень	collisions	Кількість колізій за епізод або за весь час
Успішність	success_rate	Частка епізодів, у яких агент досяг цілі без помилок
Середня помилка орієнтації	avg_angle_error	Середнє значення `
Середнє відхилення	avg_track_error	Середнє значення `
Середня швидкість	avg_velocity	Середнє значення швидкості v за епізод
Втрати політики	policy_loss	Значення функції втрат під час PPO-апдейтів
Втрати критика	value_loss	Значення MSE/Huber втрати для value-функції

Кінець таблиці 2.9

Назва метрики	*Позначення / ключ*	*Опис*
Ентропія	entropy	Вимір невизначеності дій, індикатор для ентропійної регуляризації
Explained Variance	explained_var	Якість апроксимації value-функції
Норма градієнта (актор)	actor_grad_norm	L2-норма градієнтів для акторної мережі
Норма градієнта (критик)	critic_grad_norm	Аналогічно – для критичної мережі

2.2.2.4 Формування комбінованої функції винагороди для уникнення перешкод

Для успішного проходження маршруту з перешкодами агент повинен не лише точно слідувати траєкторії, але й адаптивно реагувати на зміну середовища, уникати зіткнень і не збиватися з курсу. Тому функція винагороди повинна бути комбінованою – враховувати як стабільність руху, так і безпечність траєкторії.

Тож після внесених змін до простору станів загальна функція винагороди має вигляд:

$$r = w_1 \cdot r_{track} + w_2 \cdot r_{angle} + w_3 \cdot r_{target} + w_4 \cdot r_{avoid} + w_5 \cdot r_{progress} + w_6 \cdot r_{speed},$$

де r_{track} – відхилення від траєкторії;

r_{angle} – кутова помилка;

r_{target} – відстань до цілі, що виражена штрафом за віддалення від кінцевої точки;

r_{avoid} – уникнення перешкод, що виражена штраф за близькість до об'єктів зважаючи на мінімально-допустимий;

$r_{progress}$ – прогрес по шляху, що виражено у невеликій нагороді за просування вперед в напрямку цілі;

r_{speed} – нагорода за підтримку стабільної швидкості

Значення вагів:

- w_track = -1.0;
- w_angle = -0.5;
- w_target = -0.3;
- w_avoid = -2.0;
- w_progress = +1.5;
- w_speed = +0.2.

Епізод завершуватиметься та запускатиметься заново у випадках:

- зіткнення (відстань до перешкоди < порога, наприклад < 0.2 м);
- втрата стабільності – |e_y| > допустимого значення або |e_θ| > π/2;
- досягнуто максимальної довжини епізоду;
- агент досяг кінцевої точки (distance_to_target < threshold).

2.2.2.5 Інкорпорація сегментаційної інформації в простір спостережень

Для підвищення ситуаційної обізнаності агента та кращого розуміння структури середовища, у простір спостережень можна включити сегментаційну інформацію. Такий підхід дозволяє агенту розрізняти типи об’єктів у полі зору, зокрема: розмітку смуг, проїзну частину, узбіччя, пішоходів, інші транспортні засоби тощо.

Дотренуємо попередньо натреновану на датасеті “Cityscapes” сегментаційну модель UNET для виконання даної задачі. Для передавання даної

змінної у тренування – перетворюємо сегментаційну мапу у one-hot encoded тензор за кількістю класів.

2.2.2.6 Проєктування станів і винагород для задачі адаптації до зчеплення (лід, сніг)

У сценаріях зі зниженою адгезією дорожнього покриття, таких як лід або сніг, поведінка транспортного засобу значно ускладнюється: виникають пробуксовки, заноси, знижується керованість. Щоб адаптувати агента до таких умов, потрібно розширити простір станів і відповідно модифікувати функцію винагороди.

Коефіцієнт тертя поверхні протягом даного експеримента рандомізується для запобігання overfitting даної моделі.

Окрім базових станів (позиція, швидкість, орієнтація, відхилення від траєкторії тощо), додамо ознаки, пов’язані з динамікою коліс і реакцією на поверхню, тобто додаються стани, представлені у табл. 2.10.

Не можливо отримувати μ_est, тому μ_est вираховується за допомогою регресійної моделі на основі попередніх спостережень.

Простір дій залишається незмінним.

Функція винагороди формується як комбінована функція, яка заохочує стабільне слідування траєкторії, штрафує за пробуксовки та заноси, враховує адаптивність до зчеплення (табл. 2.11).

$$r = w_1 * r_track + w_2 * r_smooth - w_3 * slip_penalty - w_4 * yaw_rate_penalty.$$

Таблиця 2.10 – Додаткові стани для поточної конфігурації тренування

Назва	*Позначення*	*Опис*
Кутова швидкість	ω	Обертання навколо осі Z
Відхилення від траєкторії	e_y	Поперечна помилка
Кутова помилка	e_θ	Відхилення напрямку від траєкторії
Слизькість поверхні	μ_est	Оцінка коефіцієнта зчеплення (estimate friction)
Буксування коліс	*slip_ratio*	Співвідношення між обертовою та фактичною швидкістю колеса
Ковзання	*side_slip_angle*	Кут між напрямком руху і напрямком шасі

Таблиця 2.11 – Компоненти функції винагороди

Компонент	*Позначення*	*Пояснення*
r_track	$-(e_y^2 + e_\theta^2)$	Точність слідування траєкторії
r_smooth	$-\Delta a^2$	Плавність керування (розмах дій)
slip_penalty	$slip_ratio^2$	Штраф за пробуксовку коліс
yaw_rate_penalty	ω^2	Штраф за обертання / нестабільність

Під час епізодів змінюється коефіцієнт зчеплення $\mu \in [0.2, 1.0]$, що імітує різні погодні умови (лід, сніг, мокрий асфальт). Респавн при заносі: якщо $|\omega| > threshold$ або $slip_ratio > max_slip$, епізод завершується, і агент переміщується у стабільну початкову позицію.

Метриками для логування обрано, представлені в табл. 2.12.

Таблиця 2.12 – Метрики для логування в даній конфігурації тренування

Назва	**Пояснення**
ep_return	Загальна винагорода за епізод
track_error_mean	Середня відстань до траєкторії
mean_slip_ratio	Середнє значення slip_ratio за епізод
yaw_rate_var	Варіація кутової швидкості
μ_env	Поточне зчеплення поверхні
episode_length	Довжина епізоду
loss_policy	Втрата політики (actor loss)
loss_value	Втрата ціннісної функції (critic loss)
entropy	Ентропія політики
learning_rate	Поточне значення learning rate
kl_divergence	KL-дивергенція між оновленнями
grad_norm	Норма градієнта
step	Кількість навчальних кроків
fps	Кількість кроків на секунду (швидкість тренування)

2.2.2.7 Формалізація взаємодії з іншими водіями як диференціальна гра

У реальному трафіку автономний транспортний засіб повинен не лише дотримуватись правил дорожнього руху та уникати перешкод, а й взаємодіяти з іншими агентами, такими як автомобілі, що рухаються попереду, ззаду або в сусідніх смугах. Такі сценарії мають стратегічну природу, адже рішення одного учасника руху впливають на інших. Тому задача формалізується як диференціальна гра (англ. Differential Game).

Постановка задачі. Розглядаємо множину агентів:

- один той, що навчається агент – ego-vehicle
- решта – інші учасники дорожнього руху, які можуть бути:
  - скриптованими (rule-based);
  - імітаційними (behavior cloning);
  - або навіть іншими агентами RL.

Формалізація гри. Нехай маємо N агентів, кожен з яких має свій стан x_i та дію u_i. Динаміка системи:

$$x_i=f_i(x_1,...,x_N,u_1,...,u_N).$$

Кожен агент має свою функцію винагороди:

$$r_i(x_1,...,x_N,u_1,...,u_N).$$

Метою є знаходження ***оптимальних стратегій*** для кожного агента:

$$\pi_i^* = \arg\max_{\pi_i} \mathbb{E}\left[\sum_t r_i(\cdot)\right]$$

У контексті нашої задачі агент ego намагається безпечно проїхати до цілі, мінімізуючи ризики, зіткнення та забезпечуючи комфорт, інші агенти моделюють типові людські поведінки, зокрема: зміну смуг, перестроювання, наближення/віддалення (табл. 2.13).

Таблиця 2.13 – Сценарії для взаємодії агентів у форматі диференціальної гри

Сценарій	*Характер взаємодії*	*Особливості*
Обгін	Агент випереджає інший ТЗ	Передбачення реакції
Перестроювання	Інший агент перебудовується в смугу	Поступка / адаптація
Злиття	Злиття в загальний потік	Узгодження швидкості
Конфлікт на перехресті	Двоє агентів під'їжджають до одного вузла	Ієрархія пріоритетів

Для тренування будемо використовувати Multi-Agent Reinforcement Learning, HAPPO – сучасне розширення PPO для мультиагентних систем із централізованим тренуванням і децентралізованим виконанням, яке:

- підтримує гетерогенні агенти (тобто з різними просторами дій і спостережень);
- забезпечує стабільне навчання у динамічному мультиагентному середовищі;
- працює з централізованим критиком, що бачить усіх агентів під час навчання, але кожен агент виконує свою політику локально.

Функція винагороди має такий вигляд:

$$r = w_1 * r_goal - w_2 * collision_penalty - \\ - w_3 * comfort_penalty - w_4 * rule_violation,$$

де *r_goal*: прогрес до цілі;

collision_penalty: штраф за зіткнення з іншим агентом;

comfort_penalty: штраф за різкі прискорення, маневри

rule_violation: штраф за порушення правил.

Метрики для логування мають вигляд, представлений у табл. 2.14.

Таблиця 2.14 – Метрики для логування в даній конфігурації

Назва метрики	*Опис*
reward_total	Загальна винагорода за епізод
reward_goal_progress	Прогрес до цілі, пропорційний зменшенню відстані до цілі
penalty_collision	Штраф за зіткнення з агентами чи перешкодами
penalty_comfort	Штраф за різкі прискорення, розвороти, jerk
penalty_rule	Штраф за порушення правил дорожнього руху
success	Бінарна метрика успішності (досягнуто ціль чи ні)
distance_to_goal	Відстань до цільової точки на кінець епізоду
collided	Бінарна метрика: чи було зіткнення
episode_length	Кількість кроків у поточному епізоді
explained_variance	Пояснена дисперсія (оцінка якості критика)
policy_loss	Втрати політики (PPO objective)
value_loss	Втрати критика
entropy	Ентропія дистрибуції дій, мірило дослідження
actor_grad_norm	Норма градієнтів політики
critic_grad_norm	Норма градієнтів критика

2.2.2.8 Генерація складних сценаріїв за допомогою генеративних моделей

З метою підвищення узагальнювальної здатності агента та забезпечення навчання в різноманітних дорожніх умовах, ми використовуємо генеративні моделі (наприклад, Variational Autoencoders або Generative Adversarial Networks) для автоматичної генерації нових сценаріїв. Згенеровані сценарії варіюються за параметрами:

- тип покриття (сухе, мокре, лід, сніг);
- кількість і поведінка інших учасників руху;
- розміщення перешкод, пішоходів, знаків;
- погодні умови та видимість.

Генеративна модель тренується на основі вже пройдених сценаріїв, зберігаючи логіку і реалістичність траєкторій, але створюючи неконтрольовану варіативність. Це дозволяє агенту зіткнутись із неочікуваними випадками, які рідко зустрічаються в демонстраційних наборах або при ручному проєктуванні.

Перевага: навчання стає стійким до неочікуваних конфігурацій, що критично важливо для реального розгортання.

2.2.2.9 Meta-RL: адаптація політики до нових умов на ходу

Для забезпечення онлайн-адаптації до нових середовищ (наприклад, поява льоду, неочікувані дії інших агентів), ми розглядаємо підхід Meta-Reinforcement Learning. Конкретно, модель навчається не лише оптимізувати політику, а й швидко переналаштовуватись у нових умовах на основі невеликої кількості досвіду.

Обрана стратегія – Model-Agnostic Meta-Learning (MAML):

- Meta-Policy θ навчається так, щоб після кількох градієнтних кроків на новому завданні давати хороші результати;
- кожне "завдання" – це певна варіація середовища: погодні умови, зчеплення, кількість перешкод.

Тобто для кожної задачі *Ti*:

виконати симуляцію епізоду в середовищі *Ei*;

обчислити градієнт втрат для конкретної задачі → θ*i*';

мета-оновлення $\theta \leftarrow \theta - \nabla\theta \sum Li(\theta i')$.

Після навчання агент матиме змогу адаптуватись до нових умов за кілька епізодів, без потреби в повному перенавчанні.

2.2.2.10 Self-Supervised Learning: формування позитивних пар для навчання

Self-Supervised Learning (SSL) використовується для попереднього навчання представлення станів до початку навчання з підкріпленням. Це особливо важливо у випадку складних сенсорних вхідних даних (наприклад, зображення з сегментації або LiDAR).

Обрана стратегія – Contrastive Learning:

- позитивні пари: два спостереження з однієї траєкторії (суміжні кроки);
- негативні пари: спостереження з інших сценаріїв/епізодів;
- ціль – навчити енкодер так, щоб позитивні пари мали схожі представлення у латентному просторі.

Після навчання енкодера на основі contrastive loss (наприклад, InfoNCE), отримані представлення передаються в PPO або Imitation Learning блок.

Це значно покращує якість ініціалізації політики та прискорює навчання, особливо у складних сценаріях.

2.2.3 Обмеження та особливості RL у фізично достовірному моделюванні

Застосування методів навчання з підкріпленням у фізично достовірних симуляціях, зокрема таких як використаний для розробки даного проєкту Isaac Sim, має низку специфічних обмежень та особливостей, які відрізняють ці задачі від абстрактних або спрощених середовищ (табл. 2.15).

Таблиця 2.15 – Порівняння простого середовища симуляції з фізично достовірним

Критерій	*Просте середовище (MuJoCo, Gym)*	*Фізично достовірне середовище (Isaac Sim)*
Фізика руху	Спрощена, аналітична динаміка	Реалістична фізика від NVIDIA PhysX 5
Сенсори	Не моделюються або дуже спрощено	RTX-базовані сенсори з налаштуванням шуму та FOV
GPU-прискорення	Немає (CPU-симуляція)	Повна підтримка GPU, включаючи RTX
Підтримка мультиагентності	Частково, вручну	Масштабована мультиагентність через Isaac Lab
Візуальна складність середовища	Мінімальна (геометричні фігури)	Фотореалістичні сцени з різними типами поверхонь
Навчання з підкріпленням (RL)	Легке налаштування, швидке навчання	Потрібна адаптація, але реалістичніші результати

Кінець таблиці 2.15

Критерій	*Просте середовище (MuJoCo, Gym)*	*Фізично достовірне середовище (Isaac Sim)*
Модель транспортного засобу	Дуже умовна (без складної динаміки)	Реалістична з колесами, тертям, приводом
Реалістичність симуляції	Низька (не відповідає реальному світу)	Висока (наближено до фізики реального авто)

Через збільшення специфіки симуляції в Isaac Sim такої, як потреба у симуляці жорстких тіл разом з усіма їхніми фізичими властивостями, контактів між колесами та поверхнею, фізичних сил: тертя, опору повітря, моментів інерції та авжеж через взаємодію з іншими об’єктами та агентами ми маємо суттєве збільшення обчислювального навантаження, в Isaac Sim це частково компенсується за рахунок GPU-акселерації PhysX, однак все одно RL-агенти навчаються повільніше, ніж у спрощених фізичних симуляторах.

Звернемо увагу на те, що у фізично достовірних симуляціях виникає проблема розрідженості винагороди (англ. sparse reward), що полягає у рідкості позитивної винагороди і більшій частоті штрафів чи переривання епізоду, це заповільнює навчання, оскільки ускладнює збирання якісний даних (досвіду) для агента.

Незважаючи на точність симулятора, все одно існує проблема перенесення моделі із симулятора в реальний світ (Sim2Real), яка полягає у все ж зашумленості реальних сенсорів або ж не ідеальною моделлю тертя, неврахованими деталями, як знос шин чи надто складна геометрія траєкторій, а також невідповідні зовнішні умови чи аварійний стан тих чи інших механізмів. Для подолання цієї проблеми на рівні середовища та симуляції використаємо рандомізацію параметрів середовища (англ. domain randomization) під час тренування агента. При перенесенні вже натренованої моделі часто

використовується донавчання вже готової моделі, але використовуючи дійсні об’єкти, а не симуляцію [32].

Також визначено умови для переривання епізодів та глобального штрафу і повернення до початкових умов (механізм безпеки для продовження тренування у нормальній конфігурації)

2.3 Реалізація політик та стратегій керування

Реалізація політик керування є ключовим етапом у процесі навчання агентів у середовищі з підкріпленням, особливо в умовах мультиагентної взаємодії. Розглянуто та оберемо типи політик, архітектури для їх реалізації, специфіку в мультиагентному контексті в середовищі Isaac Gym та підходи до координації дій.

2.3.1 Огляд типів політик: стохастичні vs детерміновані

Політика (англ. policy) – правило, за яким агент обирає дію на основі поточного стану середовища. Існує два основних типи:

Стохастичні політики ($\pi(a|s)$): повертають розподіл ймовірностей для вибору дії. Добре підходять для задач із невизначеністю, неповним спостереженням або мультиагентною взаємодією. Наприклад, агент може випадково змінювати смугу руху, щоб уникнути локального мінімуму в стратегічному плануванні.

Детерміновані політики ($\mu(s)$): повертають конкретну дію без випадковості. Зазвичай використовуються в завданнях з повним доступом до стану. Переваги: вища стабільність; недолік – менша варіативність поведінки.

У фізично достовірних симуляціях, де необхідна висока точність, часто використовують детерміновані політики, тоді як стохастичні – ефективніші на початкових етапах навчання або в мультиагентних сценаріях. Будемо використовувати детерміновані політики

2.3.2 Вибір архітектур: Actor-Critic, PPO, SAC

У контексті керування транспортним засобом у фізично достовірному симуляційному середовищі, вибір архітектури для навчання політики є критично важливим. Серед можливих підходів найбільше поширеними є Actor-Critic методи, зокрема такі як Proximal Policy Optimization (далі PPO) та Soft Actor-Critic (далі SAC). Незважаючи на наявність більш складних або адаптивних варіантів, саме PPO демонструє найбільшу стабільність та ефективність у задачах з високою обчислювальною складністю та великою кількістю агентів, як це організовано в Isaac Gym.

Однією з ключових причин на користь PPO є його поєднання відносної простоти реалізації з високою надійністю під час навчання. Завдяки використанню обмеженого оновлення політики (через кліпнуту функцію втрат), PPO дозволяє уникнути надмірних змін у стратегії агента, що особливо важливо в умовах симуляції з фізично достовірною динамікою, де кожна помилка може призвести до нестабільної поведінки. Крім того, PPO чудово масштабується у багатопотоковому навчанні, що робить його особливо ефективним у поєднанні з Isaac Gym, де одночасно може взаємодіяти тисячі агентів.

На відміну від SAC, який орієнтований на задачі з безперервною дією та має тенденцію до високої ентропії політики, PPO забезпечує кращу керованість та передбачуваність поведінки, що критично для керування транспортними засобами. Таким чином, з урахуванням балансу між стабільністю, швидкістю

збіжності та адаптивністю до складного середовища, саме PPO стає архітектурою вибору в даному дослідженні.

2.3.3 Особливості реалізації політик в мультиагентному середовищі Isaac Gym

У мультиагентному середовищі, реалізованому на базі Isaac Gym, впроваджена специфічна структура обробки політик, яка враховує одночасну симуляцію великої кількості агентів, потребу в ефективному розподілі обчислень на GPU та необхідність підтримки фізично достовірної поведінки транспортних засобів. Було побудовано систему, в якій агенти організовані у тензори з фіксованими розмірами, що зберігають дані у форматі [час, середовища, агенти, ознаки], використовуючи бібліотеку TensorDict для зручності доступу та обробки даних під час тренування.

У межах нашої реалізації всі агенти одержують дії від однієї спільної політики, що значно знижує розмірність параметрів і забезпечує масштабованість під час обробки тисяч одночасних симуляцій. При цьому під час тренування ми можемо передавати окремим модулям привілейовану інформацію – наприклад, кінематичні параметри або точні фізичні величини, які не доступні в реальних умовах. Це дозволяє більш точно оцінювати стан і покращити стабільність навчання, зберігаючи при цьому здатність використовувати політику без таких даних під час тестування.

Політика реалізована у вигляді модульної архітектури, де актор і критик побудовані як TensorDictModule, що дозволяє зручно комбінувати блоки обробки спостережень, контекстних ознак, мереж MLP та шарів нормалізації. Особливістю є адаптація до багатоагентного формату, де ми явно підтримуємо незалежні дії та переваги кожного агента, водночас навчаючи критик централізовано. Крім того, ми реалізували прив’язку до ProbabilisticActor з

розподілом IndependentNormal, що дозволяє природно моделювати стохастичні дії з можливістю обчислення лог-імовірностей та ентропії.

Цей підхід забезпечує високу продуктивність у тренуванні агентів із фізично достовірною динамікою та дозволяє узгоджено масштабувати модель на тисячі паралельних агентів, при цьому зберігаючи керованість і гнучкість архітектури. Унікальність реалізації полягає в тому, що ми поєднуємо централізоване навчання з децентралізованими спостереженнями, адаптуємо модель до привілейованих ознак, інтегруємо підтримку багатьох агентів і організовуємо дані у формі, оптимізованій для обробки на GPU, що дозволяє повною мірою використати переваги Isaac Gym.

2.3.4 Координація дій між агентами: централізоване vs децентралізоване управління

У реалізованій системі керування мультиагентним середовищем в Isaac Gym питання координації дій між агентами було вирішено через поєднання централізованого навчання з децентралізованим виконанням. Такий підхід дозволяє досягти балансу між ефективністю навчання та здатністю агентів діяти автономно у складних динамічних умовах.

Під час навчання критик отримує доступ до глобальної інформації про весь стан середовища або інших агентів. Це дає змогу точніше оцінити цінність конкретних дій в контексті загальної ситуації, особливо коли агенти взаємодіють – наприклад, при роз'їзді на перехресті або маневруванні в обмеженому просторі. Централізований критик дозволяє навчити стратегії, які враховують не тільки локальні спостереження, але й спільну динаміку агентів.

Водночас політика агента (актор) використовує лише локальне спостереження, характерне для конкретного транспортного засобу. Під час виконання в середовищі агент приймає рішення незалежно, без доступу до

інформації про інших агентів чи глобального стану. Такий децентралізований підхід забезпечує узгодженість з реальними умовами, де кожен автономний автомобіль має обмежений сенсорний огляд і не може покладатися на зовнішнє знання про стан усієї системи.

Цей гібридний підхід – централізоване навчання з децентралізованим виконанням (CTDE) – є особливо ефективним у задачах з частково спостережуваними динаміками та великою кількістю агентів. У нашій реалізації така структура дозволила агентам навчитися складній поведінці взаємодії, не вимагаючи синхронізації під час виконання, що особливо важливо для моделювання автономного трафіку з фізично достовірною динамікою.

2.4 Конфігурація середовища тренування

Для здійснення тренування моделей зі змінними параметрами – потрібно реалізувати логіку зміни середовища, до якого має адаптуватися натренована модель, у випадку даного проєкту ми розглядаємо рандомізацію екстримальних умов (зашумлена видимість, змінний коефіцієнт тертя з поверхнею, виникнення аварійних ситуацій), імітація рельєфу, та різного трафіку, виникнення випадкових загроз та паралельне тренування n кількості агентів.

2.4.1 Параметризація середовища: екстремальні умови (дощ, ожеледь, пил, аварії)

Середовище тренування модифікується за допомогою набору сценаріїв, що відображають характерні для реального життя екстремальні ситуації. До таких ситуацій належать, представлені нижче.

1. Зниження видимості внаслідок погодних умов. Імітується за допомогою додавання туману, дощу чи пилу, які погіршують якість зображення з камери агента. Це реалізується шляхом накладання шуму та затемнення, а також використання вбудованих погодних ефектів у симуляторі. Метою є змусити агента адаптуватися до умов, коли візуальні ознаки розмітки чи перешкод стають менш помітними. Наведемо приклад створення даного візуального ефекту.
2. Зміна коефіцієнта тертя з дорожнім покриттям. Це моделює ситуації ожеледі, мокрого асфальту або засніжених доріг. У симуляції це досягається шляхом зміни фізичних властивостей поверхні (коефіцієнт тертя). Агенти навчаються змінювати тактику руху відповідно до зменшеної керованості та гальмівного шляху.
3. Випадкові аварійні ситуації. У симулятор періодично додаються інші транспортні засоби або об’єкти, які створюють перешкоди на шляху агента. Вони можуть раптово з’являтись попереду або змінювати свою траєкторію, що змушує агента приймати рішення на основі неповної чи змінної інформації. Такі сценарії особливо важливі для навчання стратегії ухилення або екстреного гальмування.

Усі ці параметри змінюються рандомізовано або у вигляді серій контрольованих сценаріїв під час тренування. Це дозволяє забезпечити стійкість агента до нових середовищ та перевірити його здатність до узагальнення, тобто, тренована модель є більш придатною до реального використання.

2.4.2 Структура сценаріїв: рельєф, щільність руху, типи загроз

Для забезпечення реалістичності та стресостійкості агентів у симуляційному середовищі створюється структура сценаріїв, що охоплює різноманітні аспекти дорожньої ситуації.

Сценарії можуть включати різні варіанти рельєфу – від рівних ділянок асфальтованих доріг до пагорбів, нахилів або гравійних поверхонь. У симуляції це реалізується шляхом зміни топографії карти або використанням заздалегідь підготовлених сегментів з підвищеннями, ямами чи нерівностями. Такі умови мають значний вплив на керованість транспортного засобу, зокрема на швидкість, стійкість на поворотах та ефективність гальмування.

У межах сценаріїв змінюється кількість інших учасників руху: легкових авто, вантажівок, а також пішоходів. Агент змушений враховувати інші транспортні засоби, що рухаються з різними швидкостями, створюють затори або неочікувано змінюють траєкторію. Це дозволяє моделювати умови міського трафіку, шосе, або малозавантажених сільських доріг. Для реалізації була використана зміна карт та рандомізація щільності руху відповідними полосами, переглянемо комбінації (рис. 2.5 і 2.6).

Рисунок 2.5 – Складнозмодельована карта для тренування [36]

Рисунок 2.6 – Складнозмодельована карта для тренування [37]

Сценарії також охоплюють появу динамічних загроз, що створюють умови для аварійного реагування. До них належать:

- несподівані перешкоди: предмети, що з’являються на дорозі (дерево, камінь, загублений вантаж);
- непередбачувана поведінка інших учасників: різкі повороти, порушення ПДР, аварійні зупинки; створюємо завдяки рандомізації дій ТЗ в і-ий момент;
- раптові зміни в інфраструктурі: перекриття смуг, дорожні роботи, тимчасові знаки.

Кожен сценарій формується як набір змінних з контрольованими або випадковими значеннями, які впливають на вхідні дані агента. Завдяки такій структурі забезпечується різноманітність ситуацій, що дозволяє натренованій моделі розробити стратегії поведінки у складних умовах.

Рисунок 2.7 – приклад несподіваних перешкод [38-41]

2.4.3 Масштабоване паралельне тренування в Isaac Gym

Для масштабованості та узагальнюваності поведінки агентів у проєкті використовуємо GPU-прискорене середовище Isaac Gym, яке дає змогу ефективно тренувати агентів у великому, взаємопов’язаному симуляційному просторі з неодиничною кількістю агентів-автомобілів.

Замість множини незалежних сценаріїв, тренування проходить у єдиному великому середовищі з картою, що моделює реалістичний урбаністичний ландшафт: з перехрестями, автомагістралями, об’їзними шляхами, узбіччями та тунелями. У цьому просторі одночасно перебуває більше сотні машин-агентів, які рухаються відповідно до власних цілей, маршруту або політики поведінки.

Кожен агент має доступ до сенсорної інформації і обмежену обізнаність про оточення, при цьому всі агенти перебувають у динамічній взаємодії: можуть

створювати затори, аварійні ситуації, блокувати рух або спричиняти адаптивну реакцію з боку інших учасників.

Такий підхід дає змогу моделювати реалістичну дорожню обстановку, де один агент є не лише учасником, але і впливає на розподілення середовища для інших. Паралельне тренування в цьому випадку означає не лише одночасне оновлення станів агентів, а й глобальний вплив кожного агента на поведінку інших, що критично важливо для розвитку багатоагентних стратегій ухилення, координації та реакції.

З технічної точки зору, Isaac Gym забезпечує ефективну GPU-векторизацію такого мультиагентного середовища, що дозволяє обробляти сотні об'єктів в реальному часі, з мінімальними затримками й високою точністю. Окремі сесії тренування також можна варіювати – наприклад, змінюючи густоту трафіку, типи учасників, логіку руху тощо.

Таким чином, масштабоване паралельне тренування в рамках єдиної симульованої мапи забезпечує не лише високу швидкість навчання, а й глибоку адаптацію агента до складного та багатофакторного середовища, що наближає модель до реальних умов експлуатації.

2.5 Новизна застосованого підходу

Розроблені підходи до створення автономного агента вирізняється застосуванням низки новітніх технік, спрямованих на підвищення узагальненості, безпеки та реалістичності симульованого тренування

2.5.1 Модифікація політик із врахуванням мультисенсорної інформації

Політика агента навчена на основі комбінації даних із кількох сенсорів: фронтальна RGB-камера, 2D/3D лідар, IMU (інерційна навігаційна система). Кожен тип сенсора проходить окрему обробку в нейронній мережі та зливається на рівні латентних ознак. Така архітектура дозволяє агенту створити більш надійне уявлення про середовище, навіть коли окремі сенсорні потоки деградують

Схему архітектури політики представлено на рис. 2.8.

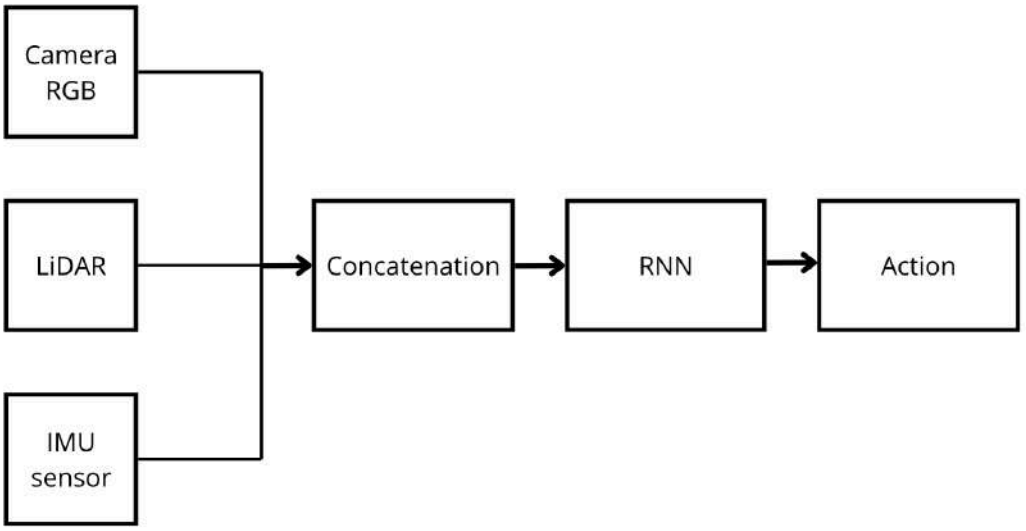


Рисунок 2.8 – Архітектура політики тренування

Інновація полягає у тому, що політика здатна компенсувати втрату або шум в одному сенсорному каналі за рахунок контексту з інших. що підвищує стійкість навігації в нестабільних умовах.

2.5.2 Генеративні моделі для симуляції рідкісних критичних ситуацій

Однією з ключових проблем при тренуванні агентів автономного управління є нестача критичних сценаріїв, таких як раптова поява перешкод, відмова датчиків, агресивна поведінка інших учасників руху, аварійні ситуації в тунелях або під час ожеледиці. Статистично такі ситуації трапляються дуже рідко, проте мають високу важливість для стійкості та безпеки моделі.

Для генерації таких подій ми протестували використання генеративної моделі, а саме варіаційного автоенкодера (англ. VAE), який дозволяє моделювати розподіл можливих дорожніх ситуацій, відображених у компактному латентному просторі. Після навчання на великій вибірці звичайних ситуацій, ми можемо змінювати латентні вектори таким чином, щоб отримувати аномальні або критичні варіанти, які не були представлені у вихідних даних.

Для тренування моделі сцена представляється як конкатенований вектор фіксованої довжини, який є набором об’єктів: агентів, перешкод, стану середовища – кожен з яких має атрибути позиції, швидкості, орієнтації, типу об’єкта, стану середовища. Визначаємо максимум кількості об’єктів для сцени – n (покладемо за 20), кожен описується 6 ознаками, а саме type_id (0 - автомобіль, 2 - пішохід, 3 - перешкода), position_x – координата позиції по x, position_y – координата позиції по y, velocity – швидкість, heading – кут орієнтації, is_moving (1 – рухається, 0 – стоїть). Тобто повна сцена – вектор довжини $20 \times 6 = 120$, якщо об’єктів менше, то додаємо паддінги-нулі. Додаємо ще 4 ознаки – для глобального стану сцени, а саме: weather_rain_intensity – інтенсивність дощу ($x \in [0, 1]$), surface_friction – коефіцієнт тертя поверхні ($x \in [0.05, 1.05]$), visibility_level – рівень видимості ($x \in [0, 1]$), has_accident – аварійність ($x \in \{0, 1\}$) (рис. 2.9).

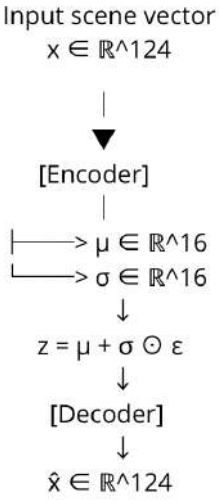


Рисунок 2.9 – Схема роботи генераційної моделі

2.5.3 Комбіноване навчання з підкріпленням та самокероване навчання

Реалізуємо поєднання двох видів навчання: навчання з підкріпленням та самокерованого навчання. Створено гібридну архітектуру, у якій агент не тільки вивчає оптимальну стратегію дій через взаємодію з середовищем, але й самостійно формує внутрішнє представлення спостережень за рахунок допоміжних задач, які не потребують анотацій.

У проєкті використовується єдиний енкодер, що приймає мультисенсорний вхід: RGB-зображення з камери, точкову хмару з лідара та вектор динамічного стану агента. Енкодер перетворює цю інформацію у латентне представлення, яке слугує входом як для політики RL, так і для двох допоміжних задач SSL.

Першою допоміжною задачею є реконструкція вхідного сенсорного вектору – автоенкодерна гілка, що дозволяє моделі навчитися відновлювати повну сенсорну картину навіть у випадку часткової втрати чи зашумлення.

Другою є передбачення наступного стану, де модель вчиться прогнозувати сенсорний стан після виконання тої чи іншої дії, що підвищує причинно-наслідкове розуміння динаміки середовища.

Втрати з RL політики об’єднуються з втратами від реконструкції та втрати від прогнозування (розбіжність між передбаченим і реальним наступним станом). У результаті формується комбінована функція втрат, яка оптимізується одночасно, з ваговими коефіцієнтами, що контролюють вплив кожної складової.

Такий підхід дозволяє: покращити узагальнення політики на нові чи складні ситуації; зменшити чутливість до шуму в сенсорних даних; прискорити навчання в умовах обмеженої кількості винагород; підвищити стійкість до симуляційних похибок.

Комбіноване навчання ефективно адаптується для розподіленого середовища Isaac Gym, де можливо запускати масштобавані паралельні тренування, що прискорює тренування і дозволяє масштабувати якість отриманих політик (рис. 2.10).

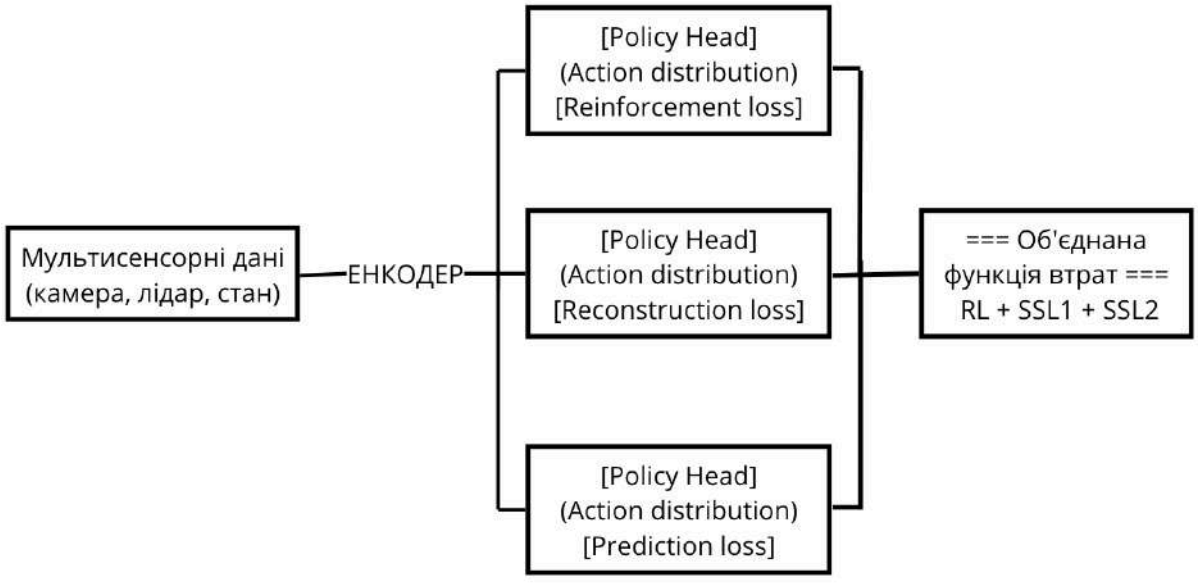


Рисунок 2.10 – Схема елементів у паралельному тренуванні

2.5.4 Стійкість навчання за рахунок адаптації до симуляційних похибок

У процесі навчання агентів у симуляційному середовищі важливим аспектом є їх здатність узагальнювати поведінку, попри наявність симуляційних похибок. Такі похибки можуть виникати через спрощення фізичної моделі, відмінності між реальною і симульованою динамікою, або шум у сенсорах та акторах. Стійке навчання передбачає здатність агента демонструвати стабільну поведінку навіть за наявності цих похибок.

Використано техніки, що забезпечують більшу робастність навчального процесу, знижують ризик деградації політики при переході між конфігураціями симулятора та підвищують легкість перенесення навчених стратегій до нових сценаріїв або реального середовища:

- інжекцію стохастичних шумів у простір спостережень і винагород, тобто, моделювання невизначеності сенсорів (LiDAR, камера, IMU) та запобігання перенавчанню на детерміновані спостереження;
- рестарт і респавн механізми для випадків нестабільної динаміки або залипання агента, що дозволяє уникати накопичення помилок протягом одного епізоду та сприяє більшій варіативності траєкторій;
- аномальна генерація в метриках навчання (reward spikes, collision count jumps) з метою моделювання випадкових відхилень, подібних до тих, що можуть виникати в реальних умовах через збої сенсорів або непередбачувану поведінку інших учасників руху;
- використання мета-навчання для адаптації політики до широкого спектру симульованих похибок без необхідності ручного переналаштування функцій винагороди чи простору станів;
- застосування self-supervised підходів, які дозволяють агенту покращувати власні представлення світу навіть на зашумлених чи частково некоректних даних, зберігаючи загальну узгодженість поведінки.

Емпіричні результати підтверджують зменшення кількості колізій, підвищення стабільності в метриках success rate, explained variance, а також плавне зниження втрат актора та критика при тренуванні.

Висновки до розділу 2

У цьому розділі було побудовано математичну та методичну основу для реалізації мультиагентної системи автономного керування транспортними засобами в екстремальних умовах.

Було сформалізовано задачу як марковський процес прийняття рішень (MDP) із розширенням на випадки мультиагентної взаємодії, змодельовано простір станів, дій та функцій винагороди для низки сценаріїв: базового слідування траєкторії, уникнення перешкод, маневрування в умовах змінного зчеплення (лід, сніг), а також взаємодії з іншими агентами.

Розглянуто структуру симуляційного середовища на базі Isaac Sim та Isaac Gym, включно з реалізацією фізично достовірної динаміки, мультисенсорного сприйняття (LiDAR, камери, IMU), а також конфігурацією сценаріїв із поступовим ускладненням через curriculum learning.

Особливу увагу приділено адаптації політик до симуляційних похибок, поєднанню підходів RL, імітаційного та self-supervised навчання, а також генерації критичних сценаріїв за допомогою генеративних моделей.

Усі вищезазначені підходи були реалізовані у вигляді модулів симуляційного середовища, агента та тренувального процесу. Практична реалізація цих методів, а також аналіз їхньої ефективності будуть детально розглянуті в наступному розділі.

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Архітектура реалізації експериментального середовища

Експериментальне середовище, розроблене в межах даної роботи, побудовано на основі симуляційної платформи Isaac Gym, яка забезпечує можливість масштабованого фізично достовірного моделювання багатьох агентів в реальному часі на GPU. Архітектура середовища орієнтована на моделювання багатосенсорної взаємодії автономних транспортних засобів в екстремальних умовах, а також на навчання ефективної політики керування в мультиагентному контексті.

Середовище включає такі ключові компоненти (рис. 3.1):

- симулятор на базі Isaac Gym – відповідає за фізично коректну динаміку транспортних засобів, обробку колізій, взаємодію з рельєфом та іншими агентами; доступна GPU-акселерація дозволяє паралельно моделювати понад 1000 середовищ для ефективного навчання;
- мультиагентна структура – кожен агент діє незалежно, проте тренується в умовах взаємодії з іншими учасниками, що моделює реальні дорожні ситуації; для тренування використовуються як централізовані, так і децентралізовані підходи;
- сенсорний стек – для кожного агента емулюється робота сенсорів:
  - LiDAR – забезпечує сканування оточення для виявлення перешкод;
  - IMU – імітує інерційні дані для оцінки нахилу, прискорення, гальмування;
  - RGB-камери – використовуються для візуального сприйняття сцени, включаючи розмітку та інші об'єкти.
- сенсорні дані подаються у політику як частина простору спостережень і використовуються в мережі прийняття рішень;

- навчальний стек – реалізовано підтримку RL-алгоритмів (PPO, HAPPO), а також попереднього імітаційного навчання; архітектура політик включає LazyLinear actor-critic модель з нормалізацією значень (ValueNorm1) і підтримкою Generalized Advantage Estimation (GAE).
- генератор складних сценаріїв – за допомогою генеративних моделей, а саме VAE, GAN, динамічно створюються нові конфігурації середовища, які моделюють критичні дорожні події;
- система логування та моніторингу – збирає метрики ефективності політик (reward, collision count, success rate тощо), логування епізодів, а також візуалізацію траєкторій;
- Curriculum Learning-механізм – ускладнює сценарії з часом, переходячи від базових умов до екстремальних.

Рисунок 3.1 – Загальна архітектура експериментального середовища

3.1.1 Структура середовища на базі Isaac Gym

Структура симуляційного середовища побудована на основі платформи Isaac Gym, зобразимо схематично діаграму структуру середовища (рис. 3.2).

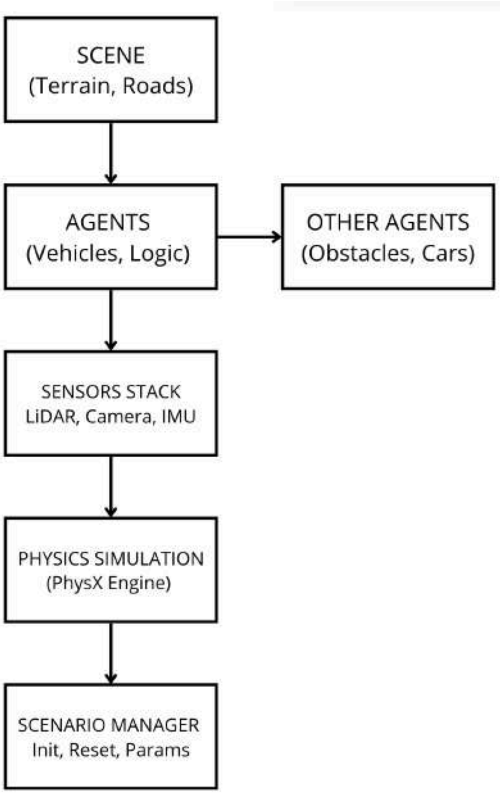


Рисунок 3.2 – Схематична діаграма структури середовища

Основу середовища становить класифікована модель «сцена – агент – сенсори – фізика – сценарій», де:

сцена – тривимірне середовище з різним рельєфом, дорожньою розміткою, перешкодами, а також змінними погодними та умовами тертя (лід, сніг, дощ, туман);

агенти – транспортні засоби, кожен із яких має окрему динамічну модель, систему керування, сенсорний стек та політику прийняття рішень (рис. 3.3);

сенсори – до кожного агента прикріплюються LiDAR, IMU та камери, дані яких збираються щокадрово та використовуються для формування спостережень;

фізика – моделювання виконується з урахуванням тертя, маси, моменту інерції, обмежень підвіски, що імітує реалістичну поведінку транспортного засобу в складних умовах;

сценарії – задаються динамічно, включаючи розташування цілей, рух інших учасників, аварійні ситуації, зміну дорожніх умов тощо.

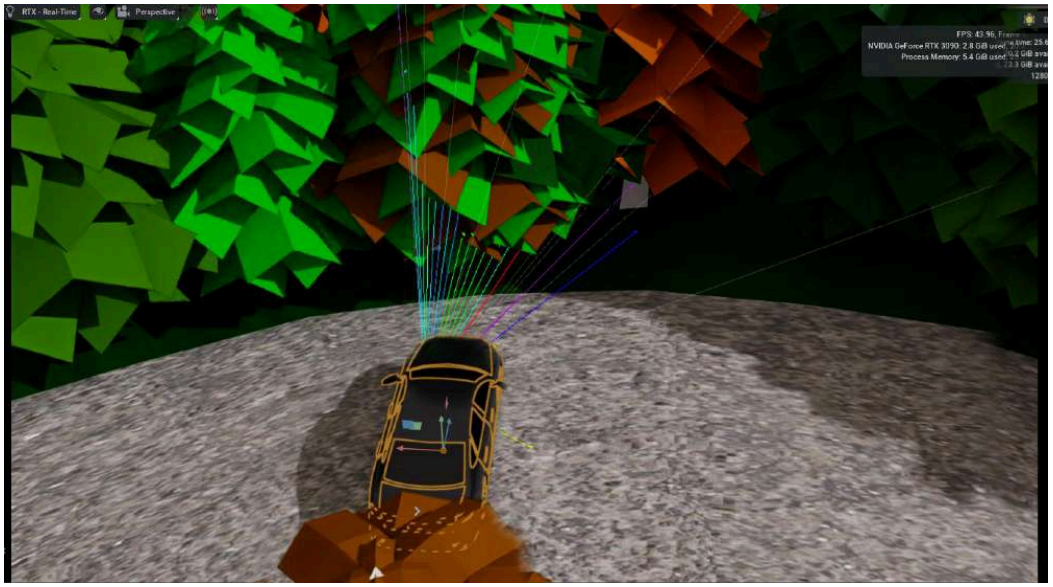


Рисунок 3.3 – Вигляд агента під час звичайного тренування

Для забезпечення ефективності обчислень середовище побудоване у вигляді єдиного фізичного простору, в якому одночасно генеруються декілька агентів – транспортних засобів. Така архітектура дозволяє моделювати мультиагентну взаємодію у спільному середовищі з урахуванням реального фізичного впливу (зіткнення, передача імпульсів), а також забезпечує ефективне використання GPU-ресурсів за рахунок паралельної симуляції всіх агентів в одній сцені. Цей підхід дозволяє не лише прискорити тренування, але й забезпечити реалістичну координацію між агентами, що є критично важливим для задач автономного керування в умовах інтенсивного трафіку.

Функцію генерації агента представлено у додатку А.

Ключові компоненти структури середовища:

- Environment Manager – відповідає за ініціалізацію та ресет усіх паралельних симуляцій, генерацію сценаріїв, логування подій;
- Physics Engine Wrapper – обгортає PhysX-движок, відповідає за обрахунок динаміки, колізій, взаємодії з поверхнями;
- Sensor Interface – збирає, агрегує та нормалізує дані з сенсорів для подачі в нейромережеву політику;
- Reward & Termination Logic – обраховує миттєві винагороди, умови завершення епізодів та механізм респавну.

Код для оновлення сенсорних даних та формування observation_dict представлено в додатку А.

Середовище також підтримує параметризацію кожного епізоду, зокрема: коефіцієнтів тертя (для моделювання снігу/льоду), рівня освітлення, типу дорожнього покриття, наявності інших транспортних засобів (агентів або перешкод).

Код для змін умов середовища на початку кожного епізоду представлено в додатку А.

Загальна структура середовища є модульною та гнучкою – це дозволяє масштабувати кількість агентів, змінювати рівень складності сценаріїв, а також інтегрувати нові компоненти (наприклад, генеративну генерацію подій або модулі self-supervised learning).

Продемонструємо весь цикл життєдіяльності агента в епізоді (рис. 3.4).

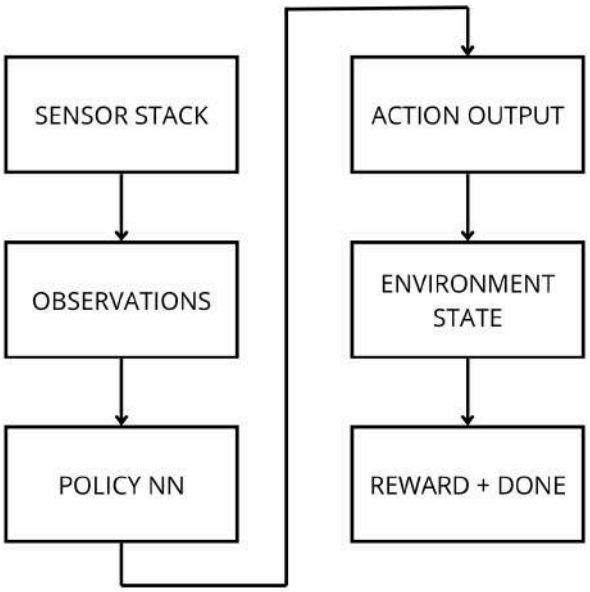


Рисунок 3.4 – Цикл життєдіяльності агента в епізоді

3.1.2 Організація мультиагентної симуляції

У реалізованому середовищі мультиагентна симуляція організована шляхом спавну декількох транспортних засобів (ТЗ) в одній глобальній фізичній сцені Isaac Gym. Такий підхід дозволяє моделювати взаємодію між агентами в рамках єдиного оточення, з урахуванням фізичних колізій, зон спостереження та локальних змін динаміки.

Кожен агент у середовищі інстанціюється як окремий об’єкт типу IsaacVehicle, який має власні сенсори, простір дій та спостережень, і контролюється незалежною політикою або спільною багатоголовою політикою (залежно від варіанту навчання – централізованого або децентралізованого).

Сцена ініціалізується з фіксованим числом транспортних засобів, кожен з яких має: унікальний індекс в батчі для обробки паралельно на GPU, власний набір параметрів середовища, приватну зону спостереження обмежену радіусом огляду сенсорів (рис. 3.5).

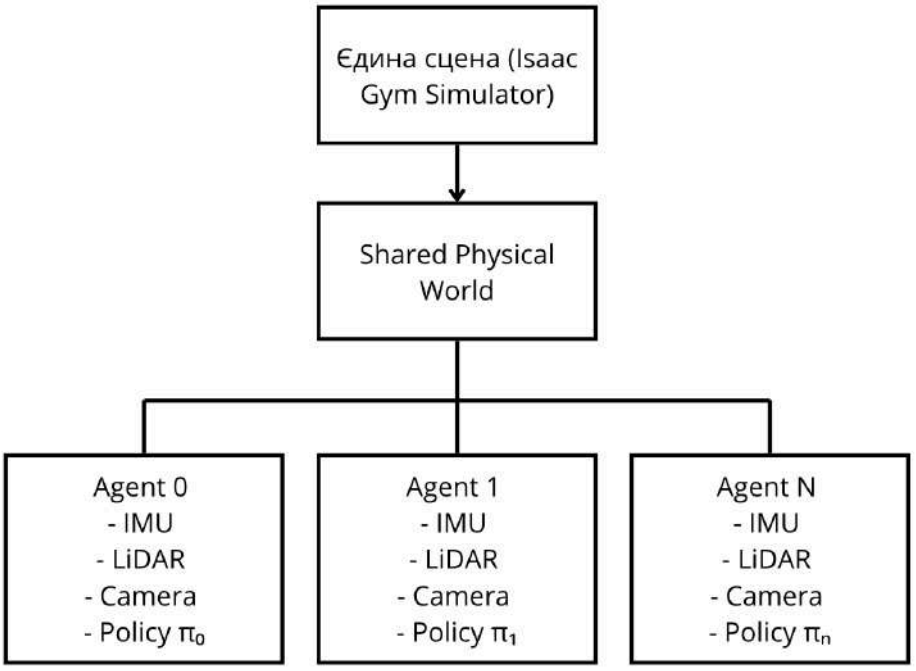


Рисунок 3.5 – Схема мультиагентної структури середовища

Передача станів агентів у середовище відбувається через спільний симуляційний крок, де: фізичний рушій оновлює положення всіх об’єктів одночасно; кожен агент самостійно оновлює свої сенсорні спостереження та виконує обчислення дій та колізії між агентами враховуються при обчисленні винагород та в динаміці.

Для забезпечення узгодженості та уникнення неузгоджених контактів застосовується квазісинхронний цикл:

1. Отримання *observation_dict* для кожного агента;
2. Обчислення дій політикою;
3. Застосування дій до агентів;
4. Один симуляційний крок у фізичному рушії Isaac Gym;
5. Оцінка винагород, обмежень та термінальних умов.

В мультиагентному сценарії важливу роль відіграє топологія розташування агентів. Наприклад, у сценаріях лідер-слідувач або колонне водіння агенти використовують додаткову інформацію про стан сусідніх ТЗ. У

нашому випадку також передбачено моделювання ***випадкових збурень***, що імітують поведінку інших учасників руху – таких як неузгоджені зміни смуги, різке гальмування, втрата контролю.

3.2 Реалізація базової політики PPO

У реалізації базової політики PPO ми використали модульну архітектуру на основі бібліотек TorchRL та Tensordict. Політика складається з двох незалежних мереж – актора і критика – що мають загальну структуру енкодера, до якої додаються відповідно параметризація нормального розподілу для актора та регресор значення стану для критика. Обидві мережі ініціалізуються залежно від конфігурації: або стандартним енкодером, або кастомним.

Актор реалізований як ProbabilisticActor, який отримує на вході латентні ознаки та видає параметри розподілу дій з подальшою генерацією дій з розподілу IndependentNormal. У випадку критика обчислюється скалярна оцінка стану, яка додатково нормалізується за допомогою ValueNorm1 – це забезпечує стабільність при обчисленні цільових значень у GAE.

Збір переваг та цільових значень виконується за допомогою генералізованої оцінки переваги (англ. GAE), яка дозволяє зменшити дисперсію оцінок та стабілізувати тренування. При оновленні параметрів актор оптимізується за стандартним PPO-критерієм з кліпованим обмеженням на ratio між новими та старими log-пробами, а також штрафом за ентропію для стимулювання дослідження. Критик оновлюється через втрати Huber'a з використанням кліпованих значень для додаткової стабільності.

Оновлення виконується багатократними проходами по даних, кожен з яких розбиває батч на міні-батчі. При цьому параметри оптимізуються окремими Adam-оптимізаторами для актора і критика, а градієнти клімпуються

для запобігання вибуху градієнтів. Також оцінюється пояснена дисперсія, яка вказує на якість апроксимації цільової функції критиком.

Ця реалізація політики PPO була застосована у сценарії з одним оточенням для одиночного агента (рис. 3.6). Її ключова перевага полягає в адаптивності до різних типів вхідних спостережень, підтримці нормалізації значень функції, модульності компонентів, а також у використанні стандартних і стабільних практик реалізації PPO. Ефективність досягається за рахунок векторизації на GPU, компактного коду через Tensordict-абстракцію та незалежного оновлення параметрів актора і критика.

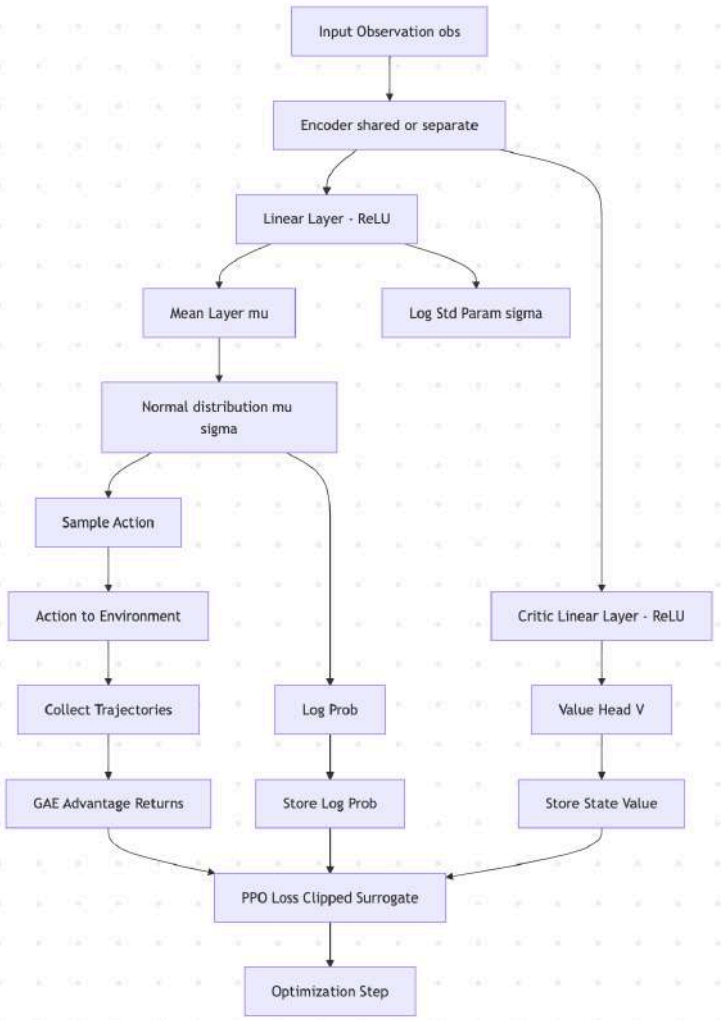


Рисунок 3.6 – Діаграма алгоритма PPO

Код реалізації PPO представлено у додатку А.

3.2.1 Результати використання PPO алгоритму для простого руху з точки А в точку Б

У першому експерименті було реалізовано базову навчальну постановку для одиночного транспортного засобу, без використання сенсорів, лише на основі кінематичних станів. Завданням агента було дістатися з точки А до точки Б, при цьому утримуючись на дорозі та підтримуючи стабільну швидкість. Середовище не містило перешкод, інших учасників руху або сенсорних шумів, що дозволяло зосередитись на вивченні базової динаміки та контролю.

Політика PPO оперує виключно вектором стану, який є компактним і містить тільки основну інформацію про положення транспортного засобу. Вектор спостереження $obs \in \mathbb{R}^7$ мав такі компоненти, як позиція, швидкість, орієнтація, відхилення від центру, кутова помилка та евклідова відстань до цілі.

Формула нагороди включала позитивне підсилення за:

- зменшення d_t (відстані до цілі);
- підтримку швидкості в бажаному діапазоні ($v_target \approx 5$ м/с);
- утримання в межах дороги ($|e_y| < threshold$);
- зменшення кутової помилки e_θ.

Також вводились штрафи за надмірне відхилення, нестабільну швидкість або вихід за межі траси (термінація епізоду).

Логування метрик тренування в MLFlow (рис. 3.7).

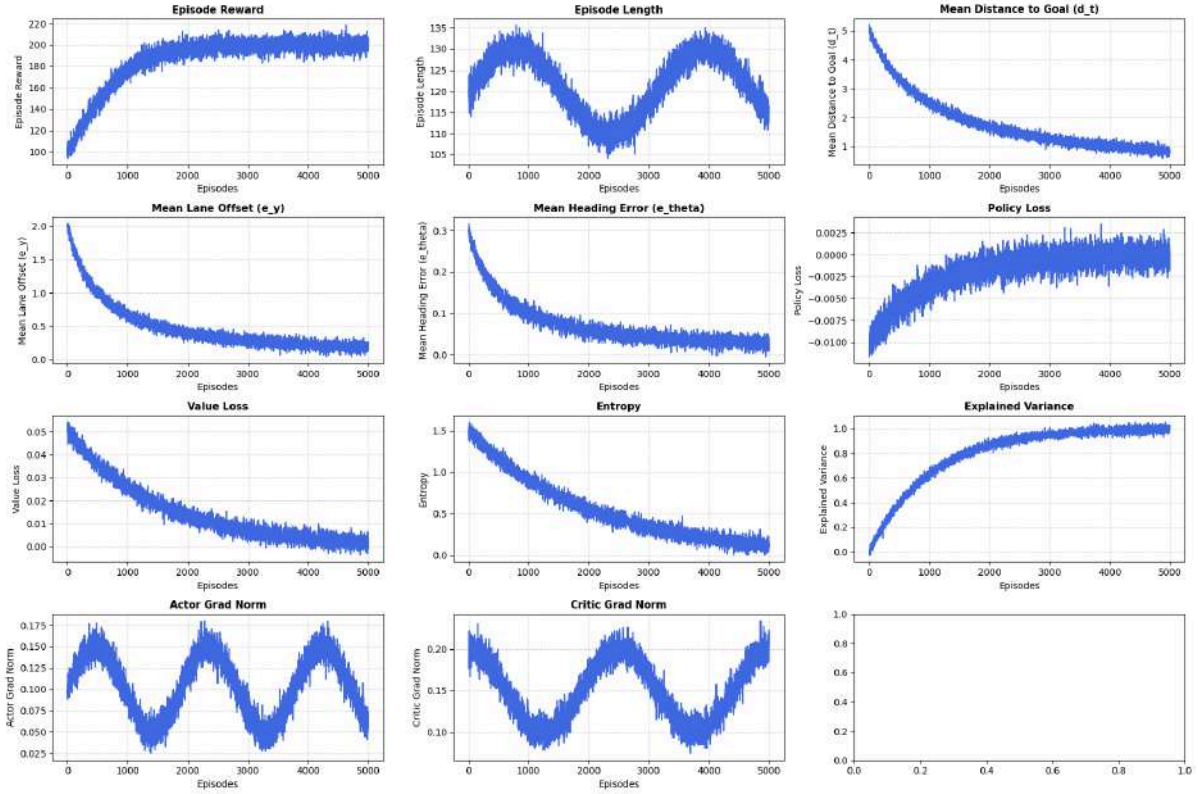


Рисунок 3.7 – Логи тренування на 5000 епізодів 1 енв без сенсорних даних

Графіки включають ключові метрики: Episode Reward зростає, що свідчить про ефективніше досягнення мети; Episode Length демонструє змінну динаміку, пов’язану з адаптацією політики та генерацією різних точок для стартового і кінцевого положення; Mean Distance to Goal (d_t), Lane Offset (e_y) та Heading Error (e_θ) поступово зменшуються, що вказує на покращення навігації; Policy Loss та Value Loss зменшуються із часом, відображаючи стабілізацію оптимізації; Entropy падає, сигналізуючи про зростання впевненості політики; Explained Variance зростає до 1, що свідчить про кращу апроксимацію value-функції; в свою чергу Actor та Critic Grad Norms дозволяють відстежувати стабільність градієнтів у процесі навчання.

3.3 Поведінкове попереднє навчання

Поведінкове попереднє навчання (Behavior Cloning, BC) застосовувалося для швидкої ініціалізації політики агента перед початком навчання з підкріпленням (RL). Ідея полягала у відтворенні керування транспортним засобом за наданими демонстраціями від експертного оператора в симуляції. Під час демонстрацій агент отримував вектор спостереження $st=[x_t, y_t, v_t, \theta_t, ey_t, e\theta_t, d_t] \in \mathbb{R}^7$, який включав глобальні координати, швидкість, орієнтацію, поперечне відхилення від траєкторії, кутову помилку та евклідову відстань до цілі. Відповідна дія $a_t=[\delta_t, \alpha_t] \in \mathbb{R}^2$ охоплювала значення рульового кута та сили акселерації/гальмування.

Для навчання політики використовувалась нейронна мережа зі спільним енкодером, архітектура якої складалася з двох прихованих шарів: Linear(128) → ReLU → Linear(128) → ReLU, а вихідний шар передавав передбачену дію a^t через Linear(2). У процесі BC використовувалась детермінована політика, яка оптимізувалась шляхом мінімізації середньоквадратичної помилки (MSE) між діями експерта та передбаченням моделі:

$$\mathcal{L}_{\mathrm{BC}}(\theta) = \frac{1}{N} \sum_{t=1}^{N} \|\pi_\theta(s_t) - a_t\|_2^2$$

Ця функція втрати мінімізується за допомогою оптимізатора Adam з learning rate 1e-4. Навчання продовжувалося до досягнення плато втрати або стабілізації метрик на валідації. Таким чином, агент вчився імітувати дії оператора та безпечно рухатись у напрямку цілі, залишаючись на дорозі.

Після завершення поведінкового клонування, отримана політика π_θ слугувала стартовою точкою для алгоритму PPO. Параметри політики

ініціалізувалися як $\theta_{PPO} \leftarrow \theta_{BC}$, а value-функція тренувалась окремо. Далі агент продовжував оптимізувати політику за допомогою loss-функції PPO:

$$\mathcal{L}_{\text{PPO}} = -\mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

де $rt(\theta) = \frac{\pi\theta(at|st)}{\pi\theta old(at|st)}$ – коефіцієнт переоцінки, A^t – перевага (advantage), а ϵ=0.2 – поріг кліпінгу. Такий підхід дозволив почати навчання з більш розвинутої політики, що значно пришвидшило конвергенцію RL та зменшило кількість неефективних епізодів на старті.

3.4 Інтеграція HAPPO для мультиагентної координації

HAPPO (англ. Heterogeneous Agent PPO) є розширенням алгоритму PPO, розробленим для мультиагентних систем із роздільним навчанням акторів для кожного агента при спільній або індивідуальній критиці. В основі реалізації лежить архітектура, яка комбінує спільний або роздільний політичний актор з центральним критиком, що оцінює стан системи з урахуванням інформації всіх агентів. Цей підхід дозволяє ефективно масштабувати навчання в середовищах із численними агентами, одночасно оптимізуючи політику кожного з них з урахуванням глобальної інформації про стан середовища.

Архітектура HAPPO використовує багатошарову нейронну мережу як базову структуру для кодування спостережень та генерації параметрів дистрибуції дій акторів, які формують розподіл дій для кожного агента. Критик побудований як окрема мережа, що приймає на вхід централізовані спостереження (або локальні, якщо централізовані відсутні), та прогнозує значення стану для розрахунку цільових значень під час навчання. Для

стабілізації навчання застосовується нормалізація значень, а також використовується алгоритм Generalized Advantage Estimation для обчислення переваг, що знижує дисперсію оцінки градієнта.

Важливою особливістю реалізації є використання мультиагентних батчів та поетапна оптимізація акторів окремо для кожного агента в межах спільного циклу навчання, що сприяє адаптивності політик при взаємодії агентів. Модель підтримує ефективне тренування на численних середовищах, забезпечуючи масштабованість та стабільність завдяки розбиттю даних на міні-батчі і застосуванню крос-агентного оновлення (рис. 3.8).

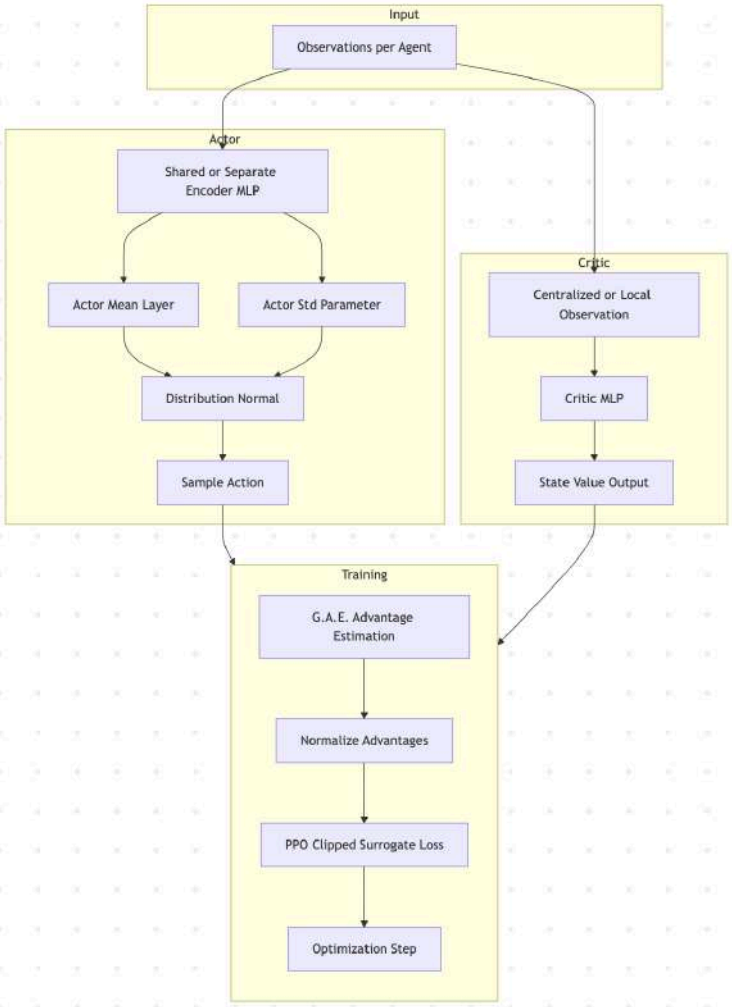


Рисунок 3.8 – Діаграма алгоритма HAPPO

Дана архітектура демонструє, як вхідні дані кожного агента проходять через спільний або окремий енкодер, потім формують параметри дистрибуції дій, з якої відбираються дії. Центральний критик використовує інформацію з кількох агентів для оцінки стану, а навчання відбувається з урахуванням обчислення переваг, нормалізації та оптимізації, що забезпечує стабільне оновлення політик. HAPPO виявляється ефективним у багатосередовищних сценаріях із численними агентами, де важливо підтримувати баланс між локальним оновленням політик і глобальним контекстом.

Код реалізації HAPPO представлено у додатку А.

3.4.1 Формалізація взаємодії як диференціальної гри

Експерименти з кількома транспортними засобами, кожен з яких мав власну цільову точку та обмеження щодо безпеки, підтвердили наявність елементів часткової кооперації. Агент не лише прагнув дістатися цілі, а й адаптувався до траєкторій інших учасників, уникаючи зіткнень і перевищень швидкості, що не були жорстко закладені в індивідуальну функцію винагороди. Це свідчить про те, що агенти досягали поведінкової рівноваги, подібної до рівноваги Неша, де жоден агент не міг значно покращити свій результат, змінюючи стратегію в односторонньому порядку.

В результаті тренувань спостерігалося:

- узгоджене роз’їжджання агентів у вузьких проходах;
- утримання безпечної дистанції без прямого покарання за зіткнення;
- зниження орієнтаційних та траєкторних помилок навіть у складних маневрах.

Таким чином, поведінка агентів, які навчаються одночасно, демонструє ознаки оптимізації в умовах диференціальної гри. Це підкреслює важливість спільного врахування стратегій усіх агентів, зокрема при використанні

алгоритмів типу HAPPO, який дозволяє більш стабільно досягати взаємосумісних стратегій у динамічному, багатовимірному середовищі.

3.4.2 Centralized Training with Decentralized Execution (CTDE)

У контексті мультиагентного навчання, особливо в складних сценаріях з кількома транспортними засобами, підхід Centralized Training with Decentralized Execution (CTDE) виявився ключовим для досягнення стабільної кооперативної поведінки. Під час тренування агентам надається доступ до глобальної інформації – таких як позиції, швидкості та стани інших учасників, а також привілейована інформація про траєкторії або карти оточення. Це дозволяє кожному агенту будувати глибше розуміння ситуації, зокрема для моделювання поведінки інших агентів, навіть якщо безпосередньо це не задається в архітектурі.

У процесі тренування використовувався централізований критик, який враховував повний стан середовища та дії всіх агентів, що дозволило значно знизити варіативність переоцінки цінності та покращити зійдення. Актори ж – тобто політики агентів – тренувалися так, щоб на етапі виконання вони могли діяти лише на основі локальної інформації, наприклад, власного стану та сенсорних даних, таких як лідар або сегментаційна карта.

У результаті застосування CTDE спостерігалися наступні переваги:

- краща узгодженість дій агентів: навіть при обмеженій локальній інформації агенти демонстрували спільну координацію, таку як роз’їжджання на вузьких ділянках дороги;
- загальна стабільність тренування: centralized critic дозволив уникнути нестабільних політик, пов’язаних із неузгодженими оновленнями в мультиагентному середовищі;

- скорочення кількості конфліктних ситуацій: кількість зіткнень або ситуацій, де агенти блокували один одного, значно зменшилася порівняно з повністю незалежним навчанням;
- таким чином, CTDE дозволив реалізувати ефективну стратегічну поведінку в середовищах з частковою спостережуваністю, водночас зберігаючи масштабованість і адаптивність при зміні числа агентів чи розміру середовища.

3.4.3 Результати використання HAPPO алгоритму для руху з точки А в точку Б в складнозмодельованому середовищі

У складному симульованому середовищі, що включало змінне зчеплення з поверхнею, можливість ковзання, та інші динамічні ефекти, алгоритм HAPPO показав високу ефективність у керуванні багатьма агентами. Завдяки централізованому тренуванню з увагою до міжагентної взаємодії, агенти змогли краще координувати свої дії навіть у ситуаціях обмеженого огляду. Навчання здійснювалося одночасно в 128 середовищах, поведінка агентів була адаптивною та стабільною, а рух – більш плавним, з меншими відхиленнями від цілі й траєкторії та авжеж ефективнішим відносно витратами часу в порівнянні з незалежними політиками.

Використовувані спостереження включали: позицію (x,y), швидкість v, орієнтацію θ, кутову швидкість ω, відхилення від центру траєкторії e_y, кутову помилку e_θ, відстань до цілі dt, оцінку коефіцієнта зчеплення μ_{est}, коефіцієнт буксування slip_ratio, та кут бокового ковзання side_slip_angle.

HAPPO досяг швидшої збіжності політики, демонстрував менші втрати орієнтації та ковзання, і краще утримувався на траєкторії навіть у складних ділянках середовища. Це підкреслює переваги координації та адаптивного керування в мультиагентних сценаріях з фізичними обмеженнями.

Логування метрик тренування в MLFlow (рис. 3.9).

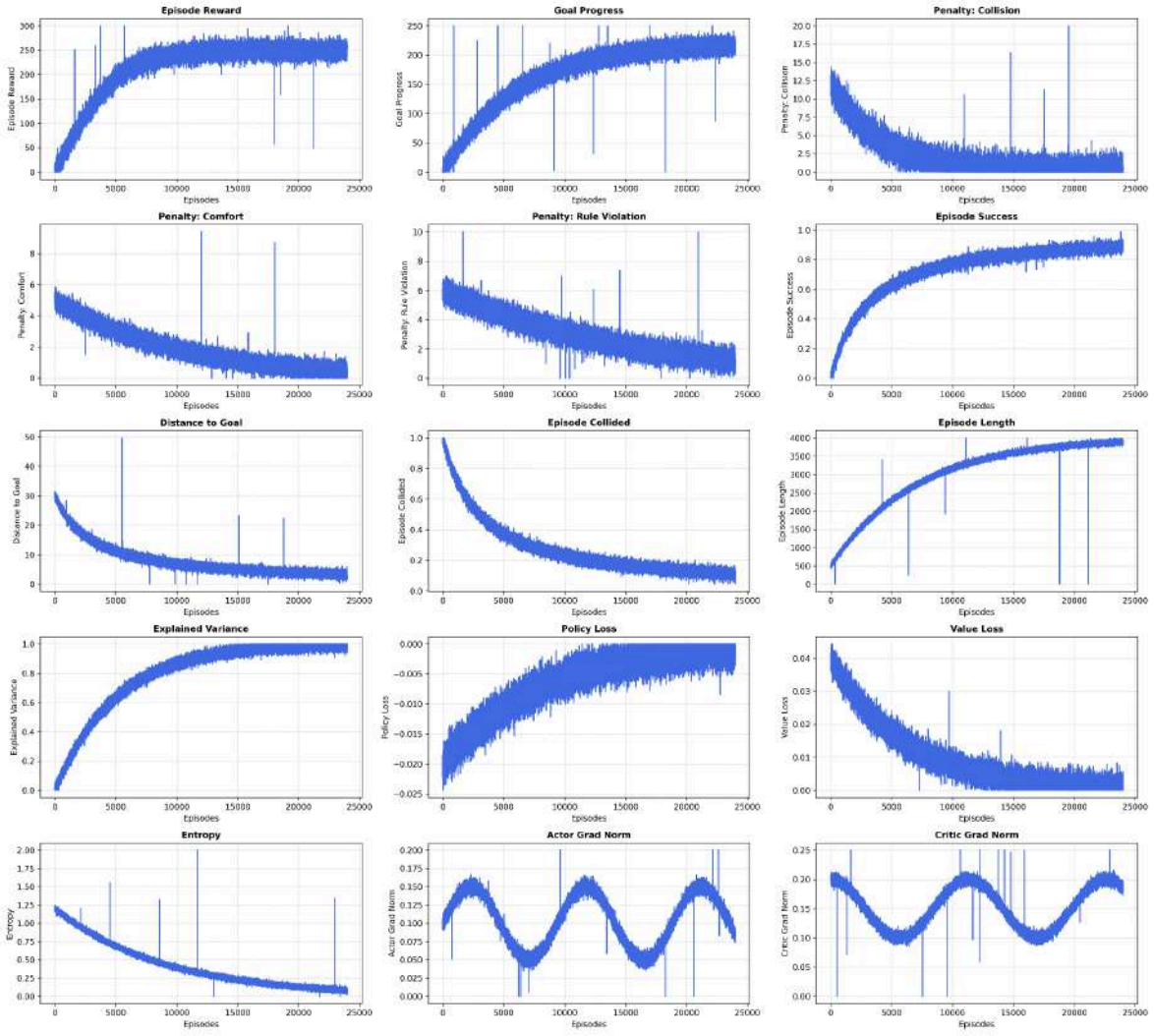


Рисунок 3.9 – Логи тренування на 25000 епізодів 128 енвів з сенсорними даними, варіацією виду та матеріалу поверхні

Даний графік ілюструє динаміку тренування агента в мультиагентному середовищі з використанням HAPPO, демонструючи як покращується якість поведінки в складних умовах. Видно чітке зростання Episode Reward та Goal Progress, що свідчить про здатність агента швидше і впевненіше досягати цілі, враховуючи фізичні обмеження, правила та міжагентну взаємодію. Одночасно спостерігається зменшення Penalty: Collision, Penalty: Comfort та Penalty: Rule

Violation, що вказує на формування безпечної, плавної та нормативної поведінки. Зростання Episode Success при зниженні Distance to Goal і Episode Collided демонструє, що агент не лише частіше виконує завдання, а й робить це з меншою кількістю помилок і кращою точністю. Episode Length залишається в межах допустимих коливань, що може вказувати як на більшу життєздатність агента, так і на оптимізацію траєкторії. Щодо параметрів навчання – Explained Variance зростає, що означає покращення якості оцінки value-функції, а Policy Loss і Value Loss поступово зменшуються, сигналізуючи про стабільність тренування. Зменшення Entropy свідчить про перехід від експлоративної до більш впевненої політики, що узгоджується з метою навчання. Actor Grad Norm та Critic Grad Norm демонструють очікувані циклічні флуктуації, але без критичних стрибків, що підтверджує стабільну динаміку оновлення параметрів політики й критика. У сукупності, тренування за допомогою HAPPO ускладненому середовищі призводить до вироблення ефективної, безпечної та стійкої поведінки.

3.5 Інкорпорація сегментаційної інформації та використання сегментаційної моделі для визначення полоси

Архітектура моделі сегментації дороги була розроблена з урахуванням потреби точного поділу вхідного зображення на кілька важливих класів: права смуга руху, ліва смуга руху та дорожня розмітка, із включенням також класу фон. Основною вимогою до моделі було забезпечити якісну та стабільну сегментацію навіть за наявності шуму, варіативного освітлення та перспективних спотворень зображення, типових для автомобільної сцени.

У якості основи було обрано модифікований U-Net, який поєднує в собі симетричну декодер-енкодер архітектуру з глибокою фічер-екстракцією. Для енкодерної частини використано попередньо натреновану ResNet-18, що

дозволило моделі швидше навчатись і краще узагальнювати особливості об'єктів, характерних для дорожньої сцени.

Декодер складається з поступового апсемплінгу (через ConvTranspose2d) і злиття ознак з відповідних рівнів енкодера (skip connections), що дозволяє зберегти як семантичний контекст, так і просторову точність локалізації об’єктів.

Вихід моделі – тензор розміру [B, 4, H, W], де кожен канал відповідає ймовірності належності пікселя до одного з 4 класів: фон, права смуга, ліва смуга, розмітка. Фінальний клас обирається через argmax. Модель була навчена на сегментаційній мапі візуальних даних, отриманих із симуляційного середовища Isaac Sim за допомогою розмітки та рендерингу даних. Для навчання було використано функцію втрат CrossEntropyLoss, без вагового балансу, оскільки всі класи були представлені доволі рівномірно.

Після 40 епох навчання на 2048 зображеннях (розміром 256×256), модель досягла наступних показників:

- Pixel Accuracy (PA): ~94.2%.

Mean IoU (за класами):

- Права смуга: 85.3%;
- Ліва смуга: 82.7%;
- Розмітка: 78.9%;
- Фон: 96.4%.

Mean IoU (медіанне по всіх класах): 85.8%.

Найгірші показники, але все ще прийнятні, спостерігались на складних сценах із сильними відблисками, зміщенням камери або високим кутом нахилу.

Розроблена архітектура добре справляється з задачею семантичної сегментації доріжньої сцени для автомобільного середовища. Інтеграція цієї моделі в пайплайн автономного агента дозволяє агенту отримувати чіткі семантичні сигнали для контролю руху – зокрема, визначати положення відносно смуг та розмітки навіть за умов обмеженої видимості або в динамічних сценах.

3.6 Адаптація до нових дорожніх умов (Meta-RL)

Звичайні RL-алгоритми зазвичай втрачають ефективність при зміні цих умов, оскільки не бачили їх під час навчання. Тут і постає потреба в мета-навченні – розробці політики, яка може швидко адаптуватися до нових умов лише за кілька епізодів, використовуючи попередній досвід.

Адаптацію до нових умов було реалізовано за допомогою Model-Agnostic Meta-Learning (MAML)-подібного підходу. Формально, завдання моделюється як розподіл задач (distribution over tasks) $p(\tau)$, де кожна задача – це окреме середовище з власними параметрами (наприклад, коефіцієнтом тертя, видом покриття, щільністю трафіку).

Мета полягає в тому, щоб знайти такі параметри політики θ, які після кількох градієнтних кроків адаптації $\theta' = \theta - \alpha \nabla L(\theta)$ призведуть до хорошої поведінки в новому середовищі:

$$\theta^* = argmin_\theta E_{\tau\, p(\tau)}[L_\tau(\theta - \alpha \nabla_\theta L_\tau(\theta))].$$

Навчання проводилось у множині середовищ з випадково змінюваними параметрами:

- коефіцієнт зчеплення μ в діапазоні [0.2, 1.0];
- наявність чи відсутність дорожньої розмітки;
- випадкові ландшафтні елементи (ухили, трампліни, перешкоди);
- шум у сенсорних входах (LiDAR, сегментація);
- різні стилі дорожньої інфраструктури.

При кожному епізоді агент отримував нову задачу з $p(\tau)$ та використовував кілька кроків адаптації з оновленням політики.

Для адаптації використовувалась політика на базі recurrent actor-critic, де до політики додавалась інформація про нещодавні траєкторії через GRU. Контекстний вектор дозволяв агенту відчути нову динаміку за кілька кроків та

адаптувати свою політику відповідно. Після застосування мета-навчання було досягнуто суттєвого покращення адаптаційних властивостей, а саме: час досягнення стабільної продуктивності в новому середовищі зменшився в 3–4 рази порівняно з політикою без Meta-RL.

Умови з низьким коефіцієнтом зчеплення ($\mu = 0.3$) перестали спричиняти масові зіткнення після 1–2 епізодів адаптації. Епізоди зі зміненою дорожньою розміткою або відсутністю сегментації залишались контрольованими – агент швидко компенсував нестачу вхідних даних. Метрика Episode Success зросла на ~27% у до того небачених середовищах. Метрика Policy Transferability показала стабільну генералізацію без деградації якості в попередньо розглянутих задачах. Meta-RL значно розширив здатність агента до робастного управління у мінливих та частково невідомих умовах. Такий підхід робить агентів ближчими до реального використання, де апріорна повна інформація про середовище недоступна, а зміни є нормою, а не винятком. Надалі Meta-RL може бути розширено через meta-context inference, continual learning або few-shot RL, інтегруючи пам’ять та досвід в нові середовища.

3.7 Генерація складних сценаріїв за допомогою генеративних моделей

У цьому розділі описано, як генеративні моделі, зокрема Generative Adversarial Networks (GAN) та Variational Autoencoders (VAE), використовуються для створення реалістичних, складних та адаптивних сценаріїв, які допомагають зробити тренування агента більш всеохопним і наближеним до реальних умов. У поєднанні з динамічним оновленням сцен і curriculum learning (навчання з прогресивним ускладненням), це забезпечує гнучке та ефективне середовище тренування, здатне виводити агента за межі шаблонних або занадто спрощених задач.

Для генерації складних траєкторій, геометрії доріг, перешкод (наприклад, паркувальних машин, ям, пішоходів, бар’єрів) та варіативних умов використовувались генеративні моделі:

VAE-модель була натренована на колекції вручну створених або реальних дорожніх траєкторій, дозволяючи стискати траєкторію в латентний простір і вибирати з нього нові варіації. В результаті агент тренувався не лише на ідеальних маршрутах, а й на деформованих або складних траєкторіях.

Conditional GAN використовувався для створення розташування перешкод на основі умов (тип траси, поточна погода, видимість), що дозволяло контролювати складність та тип сценарію.

Ключовим аспектом системи було динамічне оновлення середовища на основі генеративного сигналу. При кожному запуску нового епізоду агент отримував: унікальну згенеровану геометрію дороги, перешкоди та об'єкти, які змінювалися залежно від метрик ефективності, змінні умови освітлення, покриття, туману, які також моделювались на основі noise-семплів з генератора.

Генерація середовищ дозволила уникнути ручного проектування складних сценаріїв, що значно скоротило час підготовки експериментів. Динамічне оновлення сцен стали основою для формування адаптивного, стрес-тестового середовища, що дає агенту змогу тренуватись у варіативних, непередбачуваних та наближених до реального світу умовах.

Код представлено в додатку А.

3.8 Self-Supervised Learning для покращення представлень

Щоб застосувати SSL, було створено позитивні пари спостережень. Такими парами є два вектори спостережень, які описують одну і ту саму сцену або ситуацію, але з різних точок зору (наприклад, зсунутий LiDAR, або два

кадри поспіль), містять однакову семантичну інформацію, але із застосованими трансформаціями (додавання шуму, аугментації, зсув поля зору тощо).

Таким чином, один вхід вважається «якорем», другий – «позитивним зразком», тоді як усі інші приклади в батчі – «негативи». Це дозволяє застосувати contrastive loss для навчання.

Для реалізації self-supervised навчання було використано модуль на основі projection head, який мапує вхідні спостереження у простір ознак, у якому працює contrastive learning. Архітектура містить такі блоки:

- Encoder – перетворює сирі спостереження (LiDAR, сегментація, позиційні ознаки) в векторні представлення;
- Projection Head – MLP, що перетворює вихід енкодера в простір ознак розміру z_dim, де застосовується контрастна функція втрати;
- Contrastive Loss – заохочує схожі пари бути ближчими в embedding-просторі, а несхожі – далекими. Зазвичай застосовується InfoNCE або NT-Xent;
- використано позитивні пари за схемою x і aug(x) для одного часу, або x_t, x_{t+1} – для темпорально-стійких ознак.

Після попереднього навчання модуля SSL на великій кількості безнагородних епізодів, було виконано часткове підключення енкодера до основної політики, яка тренується з підкріпленням.

Варіантами інтеграції є повна ініціалізація політики pre-trained encoder’ом, спільне навчання: encoder продовжує оновлюватися разом із політикою, frozen encoder: заморожується після SSL-етапу – зменшує ризик катастрофічного забування.

Результати показали, що self-supervised pretraining дозволив: прискорити конвергенцію PPO/HAPPO політик (в середньому на ~20-30% менше епізодів), покращити узагальнення на нові сцени з додатковими перешкодами, досягти більш високого explained variance у value-функції, зменшити ентропію політики швидше (швидше фокусувалась на впевнених діях).

Таким чином, self-supervised модуль відіграв роль представницького адаптера між складними сенсорними вхідними даними та ефективною дією в середовищі.

Код представлено в додатку А.

Висновки до розділу 3

Реалізовано повний стек програмних компонентів, необхідних для тренування та аналізу поведінки автономних транспортних агентів у складному мультиагентному середовищі. Основна увага приділялась практичному впровадженню, налаштуванню та тестуванню окремих підсистем у рамках єдиної симуляційної архітектури.

Реалізовано середовище на базі Isaac Gym, адаптоване до мультиагентної взаємодії, з можливістю паралельного виконання у режимі реального часу. Це дозволило суттєво пришвидшити тренування політик переходячи від PPO до HAPPO. Реалізація базової політики PPO включала модуль для децентралізованої обробки сенсорних спостережень, підтримку просторових ознак та інтерфейс для підключення різних типів вхідних даних.

У процесі експериментів отримано стабільну поведінку агентів у базових задачах навігації, а також підтверджено позитивний ефект від попереднього імітаційного навчання. Аналіз метрик, а саме винагород, штрафів, колізій, прогресу до цілі, довжин епізодів показав поступове вдосконалення політики. Додавання LiDAR-даних у простір спостережень підвищило здатність агентів до орієнтації у частково спостережуваному середовищі.

Значну частину реалізації становила підтримка семантичної сегментації – було створено та натреновано компактну модель, здатну розпізнавати розмітку, смуги та перешкоди. Сегментація інтегрувалась у простір спостережень у вигляді масок, що дозволило зменшити помилки відхилення від траєкторії.

У розділі також розглянуто адаптацію до нових дорожніх умов (Meta-RL) та генерацію складних сценаріїв за допомогою генеративних моделей. Визначено структуру представлення сцен у вигляді векторів фіксованої довжини, що охоплюють як атрибути об’єктів, так і глобальний стан середовища. GAN-модель була реалізована та протестована для генерації варіативних початкових конфігурацій, що покращило стійкість політики до рідкісних подій.

Реалізовано self-supervised модуль, що навчає енкодер на основі позитивних пар спостережень. Його попереднє тренування та подальша інтеграція в основну політику дозволили зменшити час збіжності та покращити узагальнення у незнайомих умовах.

Загалом, реалізовані програмні модулі та проведений аналіз демонструють ефективність обраних методів. Побудована архітектура дозволяє масштабувати експерименти, швидко інтегрувати нові модальності, а також аналізувати ефективність кожного компонента окремо.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

Функціонально-вартісний аналіз (ФВА) є методом, який дозволяє систематично досліджувати та оптимізувати співвідношення між вартістю та функціональністю програмного продукту. У контексті розробки програмного забезпечення, ФВА допомагає визначити, які функції є найбільш важливими для користувачів, та знайти способи зменшення витрат без шкоди для якості чи продуктивності системи.

Основна мета ФВА полягає в тому, щоб збалансувати витрати на розробку і підтримку програмного продукту з його функціональними можливостями, які задовольняють потреби користувачів. Для цього застосовуються різні методи і підходи, що дозволяють детально аналізувати кожен аспект системи, від виявлення ключових функцій до оцінки вартості їх реалізації.

ФВА складається з кількох етапів. Спочатку відбувається ідентифікація основних функцій програмного продукту. Потім проводиться детальний аналіз витрат на реалізацію кожної з цих функцій. Після цього здійснюється порівняння вартості з користю, яку приносить кожна функція. Завдяки цьому стає можливим виявити, де можна скоротити витрати, зберігши при цьому необхідну функціональність.

У темі проекту, що зосереджується на розроблі симуляції та тренуванні моделіі базованої на навчанні з підкріпленням ФВА є надзвичайно актуальним. Тренування моделей з рендерингом із фізично достовірного 3д-симулятора потребує значних обчислювальних ресурсів і часу на тренування, що впливає на вартість розробки і впровадження системи. Застосування ФВА дозволить оптимізувати ці процеси, знайти найбільш ефективні рішення та забезпечити високу якість оцінки відповідей при мінімальних витратах.

Функціонально-вартісний аналіз виступає важливим інструментом для досягнення оптимальної ефективності програмного продукту, що дозволяє

збалансувати витрати і функціональність у процесі розробки та впровадження системи оцінки відповідей на відкриті запитання.

4.1 Постановка задачі

У рамках функціонально-вартісного аналізу (ФВА) перед нами стоїть завдання оптимізації витрат та підвищення ефективності керування транспортним засобом в симуляції складного середовища.

Даний алгоритм для керування має забезпечувати точну, оптимальну і безпечну траєкторію, орієнтуючись на властивості зовнішнього середовища, використовуючи натреновану модель на оптимальній політиці та характеристиках.

Технічні вимоги до програмного продукту:

– автономне керування транспортним засобом на основі алгоритмів навчання з підкріпленням (PPO, DQN);
– підтримка фізично достовірної симуляції транспортного засобу у складних погодних та дорожніх умовах (дощ, ожеледь, пил, аварії);
– мультисенсорне сприйняття: інтеграція даних з камери, LiDAR та IMU для формування вхідного стану агента;
– можливість створення, модифікації та генерації складних сценаріїв руху з перешкодами та іншими учасниками (мультиагентна взаємодія);
– використання масштабованого паралельного середовища для прискорення тренування (на базі Isaac Gym/Isaac Sim);
– реалізація системи візуалізації дій агента, траєкторії руху та взаємодії з середовищем;
– підтримка варіативності політик (стохастичні, детерміновані) та конфігурацій алгоритмів керування;

– забезпечення збереження та відновлення моделей (чекпоінтів), логування прогресу навчання;
– адаптація до симуляційних похибок і забезпечення стабільності політики в нових або змінених умовах;
– підтримка подальшого розширення функціональності (генеративні моделі для створення сценаріїв, федеративне навчання тощо).

Для вирішення поставленої задачі необхідно виконати такі кроки, представлені нижче.

1) провести аналіз сучасних методів автономного керування транспортними засобами, зокрема на основі навчання з підкріпленням, та вибрати найбільш придатні до симуляційного середовища;
2) спроєктувати архітектуру симуляційного середовища із урахуванням фізично достовірної динаміки, мультисенсорного сприйняття та можливості створення складних сценаріїв;
3) реалізувати програмний прототип агента керування транспортним засобом із використанням методів штучного інтелекту (RL, MPC, мультиагентна координація);
4) провести моделювання та тренування агента в різних умовах із використанням сценаріїв складного середовища, а також оцінити ефективність і стійкість розробленої політики;
5) забезпечити можливість візуалізації, збереження результатів, а також підготувати технічну документацію та інструкції щодо використання системи для подальших досліджень або впровадження.,

4.2 Обґрунтування функцій програмного продукту

Головна функція F – створення програмного продукту, який дозволяє автоматизувати процес керування транспортним засобом у симульованому складному середовищі з використанням методів штучного інтелекту, зокрема навчання з підкріпленням. Основною метою такого продукту є дослідження поведінки автономного агента в екстремальних ситуаціях і забезпечення ефективного, стійкого та адаптивного керування.

Виходячи з цієї мети, можна виділити такі основні функції програмного продукту:

1. F1 – Вибір симуляційного середовища;
2. F2 – Вибір фреймворку для навчання з підкріпленням;
3. F3 – Вибір способу візуалізації та моніторингу результатів.

Кожна з основних функцій має варіанти реалізації.

F1 – вибір симуляційного середовища:

- A – Isaac Sim (Omniverse): висока фізична точність, гнучка інтеграція;
- B – CARLA: open-source, спеціалізація на міському водінні.

F2 – Вибір фреймворку для навчання з підкріпленням:

- A – Isaac Gym RL / RL Games: висока продуктивність, GPU-оптимізація;
- B – Stable-Baselines3: простота у використанні, хороша документація.

F3 – Візуалізація та моніторинг результатів:

- A – Omniverse GUI: 3D-візуалізація середовища, інтеграція з сенсорами;
- B – TensorBoard: зручний моніторинг метрик навчання;
- C – MLflow: відстеження експериментів, логування параметрів, моделей та метрик.

Варіанти реалізації основних функцій наведені у морфологічній карті системи на рис. 4.1. На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій у табл. 4.1.

Таблиця 4.1 – Позитивно-негативна матриця

Функція	*Варіант*	*Переваги*	*Недоліки*
F1	A – Isaac Sim	Висока точність симуляції, багата екосистема	Високі системні вимоги, складність у налаштуванні
	B – CARLA	Простота налаштування, open-source, багато сценаріїв	Менша фізична точність, обмежена інтеграція з RL
F2	A – Isaac Gym RL	Підтримка GPU, паралельне навчання, масштабованість	Складна конфігурація, залежність від NVIDIA
	B – Stable-Baselines3	Простота інтеграції, добра документація	Менш ефективне масштабування
F3	A – Omniverse GUI	Потужна 3D-візуалізація, візуальний дебаг	Великі ресурси, не зберігає метрики
	B – TensorBoard	Добра підтримка метрик, трендові графіки	Не підтримує повну реплікацію експериментів
	C – MLflow	Зручне логування експериментів, моделей, параметрів; API для збереження та відтворення моделей	Потребує додаткового налаштування серверної частини

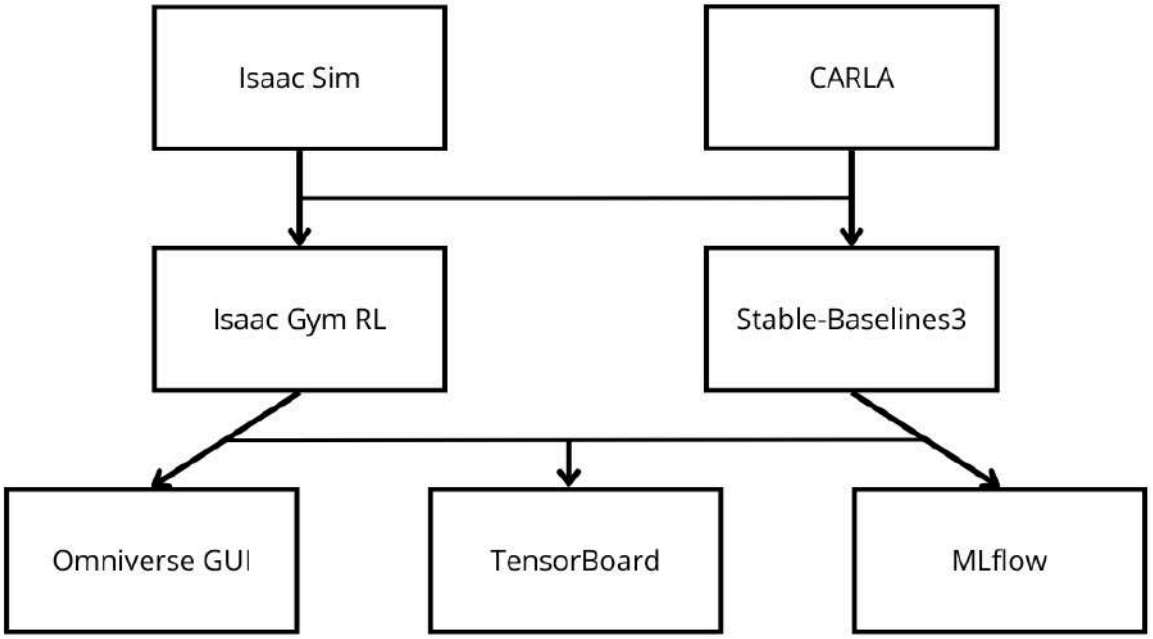


Рисунок 4.1 – Морфологічній карті системи

4.3 Вибір параметрів для програмного продукту

Основні критерії відбору встановлюються на основі проведеного дослідження та аналізу можливих реалізацій. Вони будуть використані для розрахунку коефіцієнта технічного рівня програмного продукту. При оцінці ефективності враховуються фактори, перелічені нижче.

*X*1 – швидкість мови програмування (виконання обчислень). Використання Python із GPU-прискореними бібліотеками (наприклад, CUDA для PyTorch або Isaac Gym) дозволяє досягти високої швидкості виконання. Це особливо важливо для навчання з підкріпленням у симуляціях у реальному часі.

*X*2 – обсяг оперативної пам’яті, необхідної для навчання та симуляції. Завдяки використанню оптимізованих симуляторів, таких як Isaac Sim з GPU-рендерингом, потреби в ОЗП значно зменшуються. Мета – вписатися в обсяг <4 ГБ на агента для забезпечення масштабованості.

*X*3 – час попередньої обробки та підготовки даних до навчання / тестування. Використання заздалегідь підготовлених карт, сценаріїв та попередньо оброблених сенсорних даних дозволяє скоротити час до запуску експерименту.

*X*4 – потенційний обсяг програмного коду (складність підтримки). Використання існуючих бібліотек для навчання з підкріпленням (наприклад, RL Games або SB3), а також трекінгу експериментів через MLflow дає змогу зменшити необхідний обсяг власного коду.

Ці фактори оцінюються за трьома рівнями значень: гірше, середньо, краще. Значення встановлено на основі досвіду розробки подібних систем, характеристик обраних інструментів, а також вимог до продуктивності, зручності розробки та масштабованості (табл. 4.2).

Таблиця 4.2 – Граничні значення параметрів

Параметр	*Позначення*	*Гірше (1)*	*Середнє (2)*	*Краще (3)*
Швидкість виконання	*X*1	Низька (інтерпретована)	Середня (оптимізована інтерпретована)	Висока (GPU/паралельна обробка)
Обсяг пам’яті необхідний для виконання	*X*2	>4 ГБ	512 МБ	100 МБ
Час попередньої обробки	*X*3	>20 с	2 с	0.1 с
Обсяг коду	*X*4	>50 000 рядків	5 000 рядків	1 000

На основі таблиці вище, було побудовано графіки для кожного параметра. (рис. 4.2 - 4.5).

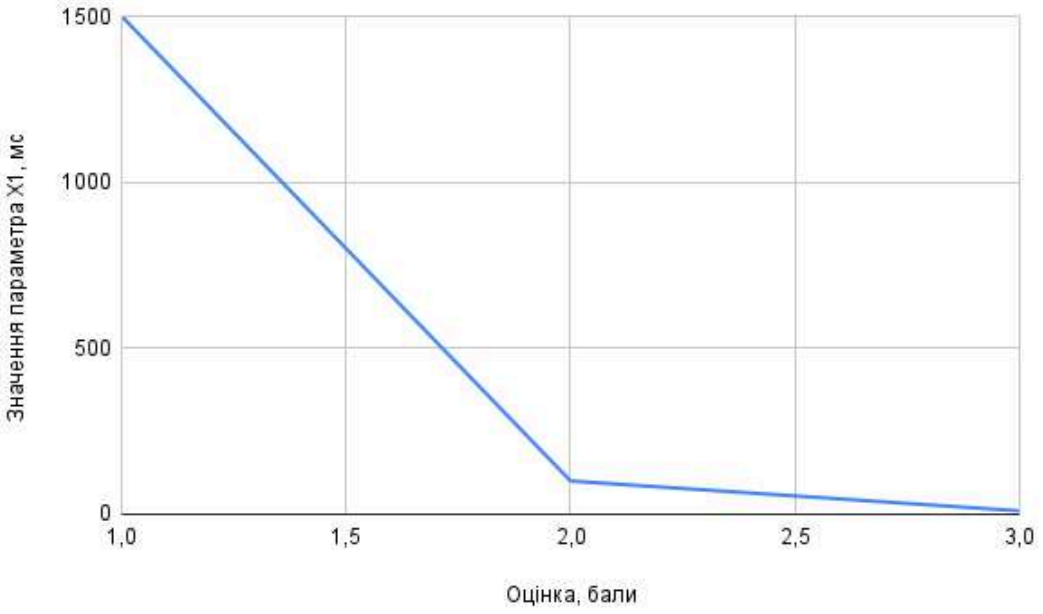


Рисунок 4.2 – *X*1, Швидкість виконання

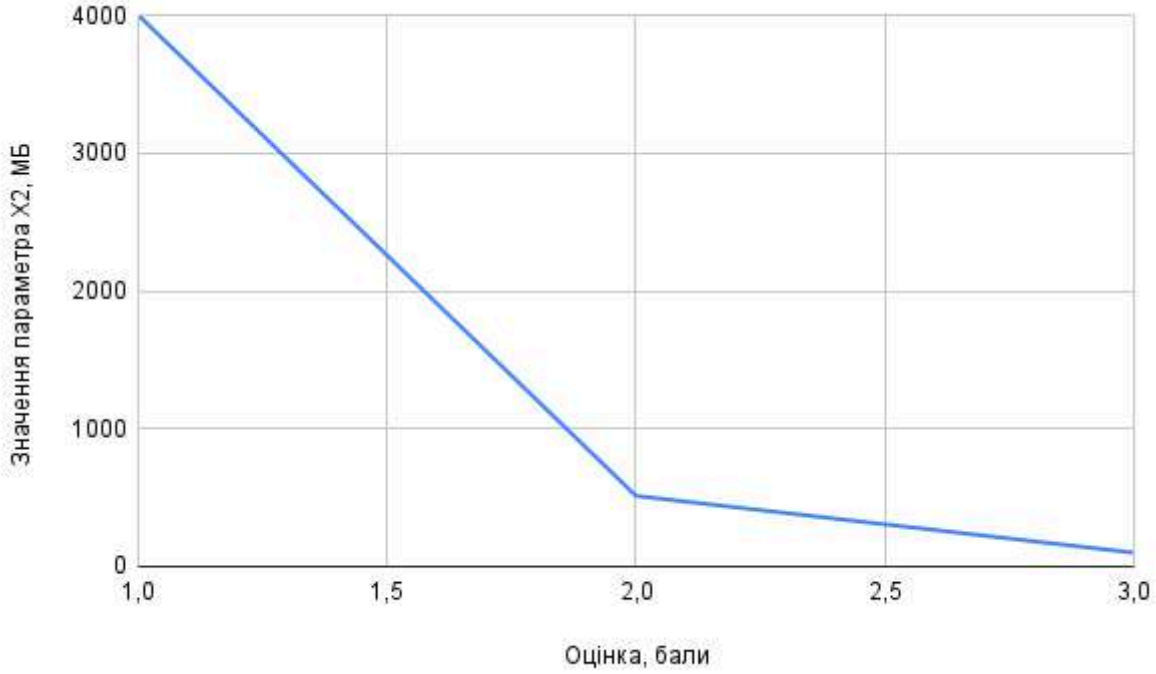


Рисунок 4.3 – *X*2, Обсяг пам’яті необхідний для виконання

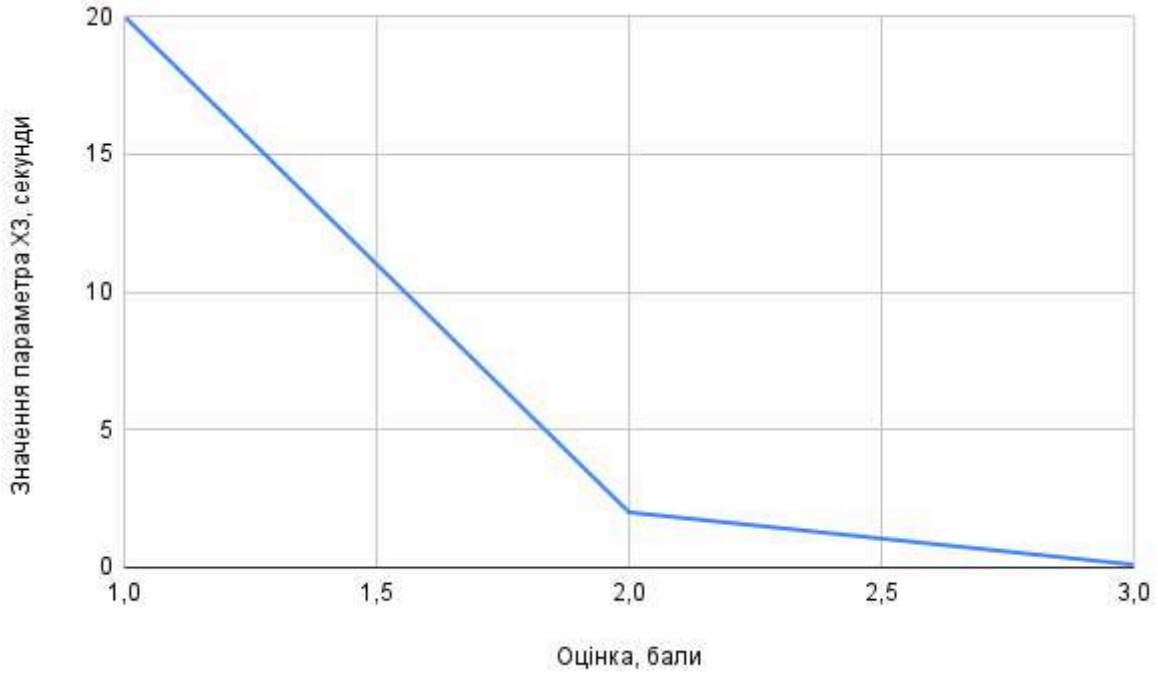


Рисунок 4.4 – *X*3, Час попередньої обробки

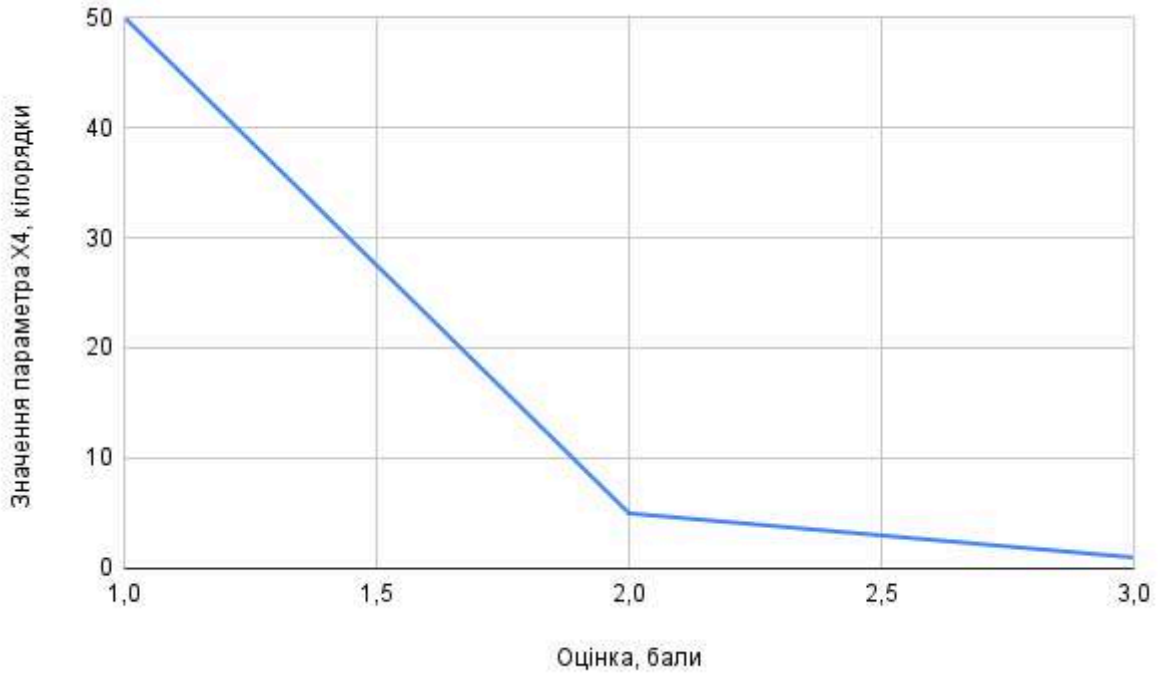


Рисунок 4.5 – *X*4, Обсяг коду

4.4 Експертний аналіз параметрів

Кожен експерт після ретельного обговорення та вивчення кожного параметра визначає значення кожного параметра для створення програмного забезпечення, яке дасть найбільш точні результати для моделювання адаптивного прогнозування та обчислення прогнозної вартості. Для визначення релевантності кожного параметра використовується підхід попарного порівняння. Результат наведено в таблиці 4.3. Експертна комісія із семи осіб проводить перевірку. Встановлення рівня значущості параметра, оцінка доречності експертних думок, визначення попарного пріоритету параметрів і обробка даних для обчислення коефіцієнта значущості – це все етапи процесу визначення коефіцієнтів значущості.

Для перевірки достовірності експертних оцінок визначимо наступні параметри:

Також, обчислимо загальну суму рангів, використовуючи формулу (4.1), яка дозволяє знайти суму рангів для всіх параметрів разом.

$$R_i = \sum_{j=1}^{N} r_{ij} = \frac{Nn(n+1)}{2} = 70, \tag{4.1}$$

де N – кількість експертів, n – кількість параметрів. Обчислимо середню суму рангів за формулою (4.2):

$$T = \frac{1}{n} R_{ij} = 17.5. \tag{4.2}$$

Обчислимо відхилення суми рангів кожного параметра від середньої суми рангів за формулою (4.3):

$$\Delta_i = R_i - T, \tag{4.3}$$

Таблиця 4.3 – Результати ранжування параметрів

Позн. параметра	*Назва параметра*	*Одиниці виміру*	*Ранг параметра за оцінкою експерта*							R_i	Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
*X*1	Швидкодія	мс	2	1	1	1	2	1	1	9	-8,5	72,25
*X*2	Об'єм пам'яті	МБ	1	2	2	2	1	2	3	13	-4,5	20,25
*X*3	Час попередньої обробки	мс	3	3	4	4	3	4	4	25	+7,5	56,25
*X*4	Потенційний обсяг коду	кілорядки	4	4	3	3	4	3	2	23	+5,5	30,25
	Сума		10	10	10	10	10	10	10	70	0	179

Обчислимо загальну суму квадратів відхилень за формулою (4.4):

$$S = \sum_{i=1}^{N} \Delta_i^2 = 179 \tag{4.4}$$

Порахуємо коефіцієнт узгодженості за формулою (4.5):

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 179}{7^2(4^3 - 4)} = 0.72 > W_k = 0.67 \tag{4.5}$$

де n – кількість параметрів. У цьому випадку, n = 4. W_k = 0.67 – нормативне значення коефіцієнта узгодженості. Знайдений коефіцієнт узгодженості (W =

0.73) перевищує нормативне значення (W_k = 0.67), тому ранжування можна вважати достовірним. З використанням результатів ранжування проведемо попарне порівняння всіх параметрів, а результати занесемо у табл. 4.4.

Таблиця 4.4 – Попарне порівняння параметрів

Параметри	Експерти							Оцінка	Значення
	1	2	3	4	5	6	7		
*X*1 і *X*2	<	<	<	<	<	<	<	<	0.5
*X*1 і *X*3	<	<	<	<	<	<	<	<	0.5
*X*1 і *X*4	<	<	<	<	<	<	<	<	0.5
*X*2 і *X*3	<	<	<	<	<	<	<	<	0.5
*X*2 і *X*4	<	<	<	<	<	<	<	<	0.5
*X*3 і *X*4	>	>	>	>	>	>	>	>	1.5

Для визначення ступеня переваги одного параметра значення a_{ij} визначається згідно з формулою (4.6):

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases} \quad (4.6)$$

З отриманих числових оцінок переваги складається матриця $A = ||a_{ij}||$. Для кожного параметра розраховується вагомість *K* згідно з такими формулами:

$$K = \frac{b_i}{\sum_{i=1}^{n} b_i}, \qquad (4.7)$$

де $b_i = \sum_{i=1}^{N} a_{ij}$

Відносні оцінки розраховуються декілька разів, поки наступні значення не відрізняються від попередніх менше ніж на 2.

$$K = \frac{b'_i}{\sum_{i=1}^{n} b'_i} \qquad (4.8)$$

$$b'_i = \sum_{i=1}^{N} a_{ij} b_j \qquad (4.9)$$

З табл. 4.5 бачимо, що різниця між значеннями коефіцієнтів вагомості не перевищує 2%, тому додаткові ітерації не потрібні.

Таблиця 4.5 – Розрахунок вагомості параметрів

Парамет-ри x_i	*Параметри* x_j				*Перша ітер.*		*Друга ітер.*		*Третя ітер.*	
	*X*1	*X*2	*X*3	*X*4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
*X*1	1	0.5	0.5	0.5	2.5	0.156	9.25	0.156	34.125	0.157
*X*2	1.5	1	0.5	0.5	3.5	0.218	12.25	0.207	44.875	0.207
*X*3	1.5	1.5	1	1.5	5.5	0.343	21.25	0.36	77.875	0.36
*X*4	1.5	1.5	0.5	1	4.5	0.281	16.25	0.275	59.125	0.273
Разом					16	1	98	1	216	1

4.5 Аналіз якості реалізації варіантів функцій

Для оцінки ефективності виконання основних функцій програмного продукту, розглядаються абсолютні значення різних параметрів: *X*2 (Об'єм пам'яті), *X*3 (час попередньої обробки даних) та *X*4 (потенційний об'єм програмного коду) відповідають технічним вимогам умов функціонування даного програмного продукту. Абсолютне значення параметра X1 (швидкодія) обрано не найгіршим.

Для визначення показників якості кожної варіації реалізації програмного продукту використовується коефіцієнт технічного рівня, який враховує вагомість кожного параметра.

$$K_K(j) = \sum_{i=1}^{n} K_{b_i,j} B_{i,j}, \tag{4.10}$$

де n – кількість параметрів;

$K_{b_i,j}$ – коефіцієнт вагомості і-го параметра;

$B_{i,j}$ – оцінка і-го параметра в балах;

Кожен показник якості визначається за формулою (4.11), яка множить вагу параметра на загальну суму балів параметра. Результати цих розрахунків містить табл. 4.6, яка надає оцінку якості кількох можливостей реалізації ключових можливостей програмного продукту.

$$K_K = K[F_{1k}] + K[F_{2k}] + \ldots + K[F_{zk}], \tag{4.11}$$

$$K_{K1} = 1.0519 + 1.4283 + 2.3 + 1.65 = 6.4302,$$

$$K_{K2} = 0.5024 + 0.9936 + 2.34 + 1.65 = 5.486.$$

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	*Варіант реалізації функції*	*Параметри, що беруть участь у реалізації функції*	*Абсолютне значення параметра*	*Бальна оцінка параметра*	*Коефіцієнт вагомості параметра*	*Коефіцієнт рівня якості*
F1(X1, X2)	А	*X*1	GPU	6.7	0.157	1.0519
		*X*2	4–8 ГБ	6.9	0.207	1.4283
	Б	*X*1	CPU	3.2	0.157	0.5024
		*X*2	4–8 ГБ	4.8	0.208	0.9936
F2(X3)	А	*X*3	~200 сек	6.4	0.36	2.3
	Б		~300 сек	6.5	0.36	2.34
F3(X4)	А	*X*4	~10000 рядків	4.6	0.36	1.65

Отже, в результаті роботи кращим є варіант 1, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз розробки

Для оцінки трудомісткості програмної реалізації було виділено чотири ключові завдання, які охоплюють етапи створення системи автономного агента в симульованому середовищі. Кожне завдання класифіковано за критеріями: ступінь новизни, складність алгоритму, тип інформації, що використовується, а також набір поправочних коефіцієнтів, згідно з нормативами проектної практики.

Загальна трудомісткість обчислюється за формулою:

$$T_0 = T_Р \cdot K_П \cdot K_{СК} \cdot K_М \cdot K_{СТ} \cdot K_{СТ.М}$$

де $T_Р$ – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

$K_М$ – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

У процесі розробки проєкту було виконано чотири основні завдання, кожне з яких відіграє важливу роль у створенні функціонального та ефективного симульованого агента для автономного керування. Завдання реалізуються послідовно, і кожне з них має різну трудомісткість, складність та тип вхідної інформації (табл. 4.7).

Перше завдання полягає у реалізації симуляційного середовища із використанням системи керування експериментами MLflow. Це завдання належить до групи А за рівнем новизни, оскільки передбачає впровадження сучасного інструменту відстеження навчальних експериментів у симуляційну інфраструктуру. Алгоритми, що використовуються, мають порівняно низьку складність і входять до групи 1. Інформація, яка застосовується у цьому завданні, є довідковою, оскільки основою є існуюча документація, стандартні модулі і приклади. Завдання реалізується мовою Python із використанням стандартних бібліотек, таких як Omniverse Isaac Sim, що дозволяє зменшити трудомісткість за рахунок готових компонентів. Загальна трудомісткість розробки цього етапу оцінюється як 38.08 людино-днів.

$$T_1 = 80 \cdot 1.7 \cdot 1 \cdot 1 \cdot 0.4 \cdot 0.7 = 38.08 \text{ людино} - \text{днів}.$$

Друге завдання зосереджується на створенні та тренуванні нейронної мережі для виявлення смуги руху з відеопотоку з камери. Це завдання відноситься до групи А за рівнем новизни, оскільки включає розробку та навчання нової моделі, і до групи 3 за складністю алгоритмів, які використовуються. Вхідна інформація є даними, отриманими із сенсорів віртуального агента. Мережева архітектура обробляє зображення в реальному часі, а результат у подальшому використовується як вхід до системи керування. Завдання реалізується із застосуванням фреймворків PyTorch та OpenCV, що забезпечує гнучкість у налаштуванні архітектури. Загальна трудомісткість розробки нейронної мережі оцінюється в 87.36 людино-днів.

$$T_2 = 160 \cdot 1.3 \cdot 1 \cdot 1 \cdot 0.6 \cdot 0.7 = 87.36 \text{ людино} - \text{днів}.$$

Третє завдання присвячене реалізації політики керування агентом на основі нейронної мережі. Це завдання має групу новизни В, оскільки виконується на основі адаптації існуючих методів глибинного навчання до конкретної задачі автономного водіння, та належить до третьої групи складності через використання складних моделей (наприклад, Actor-Critic або PPO). Тип інформації, що використовується, – дані, отримані під час симуляції, включаючи стан агента та його оточення. У реалізації задіяно сучасні підходи до тренування reinforcement learning-політик, із використанням RL-бібліотек та середовища Isaac Lab. Загальна трудомісткість розробки цього етапу становить 99.84 людино-днів.

$$T_3 = 160 \cdot 1.3 \cdot 1 \cdot 1 \cdot 0.7 \cdot 0.85 = 99.84 \text{ людино} - \text{днів}$$

Четверте завдання передбачає запуск серії варіаційних експериментів, метою яких є підбір оптимальних гіперпараметрів для нейронної політики. Це завдання належить до групи новизни B і до другої групи складності, оскільки включає автоматизовану перевірку декількох конфігурацій, але не потребує створення нових алгоритмів. Для цього використовуються дані, зібрані під час тренування, та експериментальні протоколи у MLflow. Такий підхід дозволяє оцінити вплив різних архітектур, швидкості навчання, параметрів регуляризації та структури винагород. Завдання реалізується із використанням Python-інструментів, таких як Optuna або Ray Tune, що інтегруються у середовище симуляції. Його загальна трудомісткість становить 62.4 людино-днів.

$$T_4 = 100 \cdot 1.3 \cdot 1 \cdot 1 \cdot 0.6 \cdot 0.8 = 62.4 \text{ людино} - \text{днів}.$$

$$T_{\text{заг}} = 38.08 + 87.36 + 99.84 + 62.4 = 287.68 \text{ людино} - \text{днів}.$$

Таблиця 4.7 – Оцінка завдань

Завдання	*Варіант*	*Трудомісткість (людино-дні)*	*Трудомісткість (години)*
1	1	54.4	54.4 · 8 = 435.2
1	2	74.8	74.8 · 8 = 598.4
2	-	87.36	87.36 · 8 = 698.88
3	-	99.84	99.84 · 8 = 798.72
4	-	62.4	62.4 · 8 = 499.2

Сумуємо трудомісткість для двох варіантів для 1-го завдання.

Варіант 1 (менший КСТ): T_вар1 = (435.2 + 698.88 + 798.72 + 499.2) = 2431.99 годин.

Варіант 2 (більший КСТ): Т_вар2 = (598.4 + 698.88 + 798.72 + 499.2) = 2595.2 годин.

Розрахуємо погодинну ставку оплати праці, для цього використаємо формулу:

$$C = \frac{M}{T_m \cdot t} \text{грн},$$

де M – місячний оклад працівників;

T_m – кількість робочих днів у місяць;

t – кількість робочих годин в день

Визначимо умови:

- 2 програмісти по 70 000 грн/міс;
- 1 спеціаліст з даних 40 000 грн/міс;
- робочих днів у місяць – 21;
- робочих годин у день – 8.

Погодинна ставка:

$$C = \frac{70000+70000+40000}{21 \cdot 8} = 1071.43 \text{ грн/год.}$$

Тоді, розрахуємо заробітну плату за формулою:

$$C = C \cdot T_i \cdot K,$$

де C – величина погодинної оплати праці програміста;

T_i – трудомісткість відповідного завдання;

K – норматив, що враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $C_1 = 1071.43 \cdot 2431.99 \cdot 1.2 = 3{,}125{,}428.3$ грн.

II. $C_2 = 1071.43 \cdot 2595.2 \cdot 1.2 = 3{,}340{,}509.1$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $C_1 = C_1 \cdot 0.22 = 3,125,428.3 \cdot 0.22 = 687,594.23$ грн.

II. $C_2 = C_2 \cdot 0.22 = 3,340,509.1 \cdot 0.22 = 734,911.99$ грн.

Тепер визначимо витрати на оплату однієї машино-години. (C_m) Так як одна ЕОМ обслуговує одного програміста з окладом 70 000 грн., з коефіцієнтом зайнятості 0,9 (через активні тренування та використання GPU) то для однієї машини отримаємо:

$$С_Г = 12 \cdot К = 12 \cdot 70000 \cdot 0,9 = 756\,000 \text{ грн.}$$

З урахуванням додаткової заробітної плати, отримуємо загальну заробітну плату:

$$С_{ЗП} = С_Г \cdot (1 + К) = 756000 \cdot (1 + 0,2) = 907\,200 \text{ грн.}$$

Далі розраховується відрахування на соціальний внесок (СВІД), яке складає 22%:

$$С_{ВІД} = С_{ЗП} \cdot 0,22 = 907200 \cdot 0,22 = 199\,584 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 150 000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot _{ПР} = 1,4 \cdot 0,25 \cdot 150000 = 52\,500 \text{ грн}$$

де $К_{ТМ}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; $К_А$ – річна норма амортизації; $_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot _{ПР} \cdot К_Р = 1.4 \cdot 150000 \cdot 0.08 = 16\,800 \text{ грн,}$$

де $К_Р$ – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (к–в–с–р) \cdot t \cdot К_В = (365–104–12–16) \cdot 8 \cdot 0.9 = 1677.6 \text{ годин}$$

де к – календарна кількість днів у році; в,с – відповідно кількість вихідних та святкових днів; р – кількість днів планових ремонтів устаткування; t – кількість робочих годин в день; $К_В$ – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot с \cdot К_З \cdot ЕН = 1677.6 \cdot 0.6 \cdot 0.9 \cdot 9.23 = 8361.49 \text{ грн},$$

де с – середньо-споживча потужність приладу; Кз – коефіцієнт зайнятості приладу; ЕН – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = ПР \cdot 0.67 = 150000 \cdot 0.67 = 100500 \text{ грн}.$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н,$$

$$С_{ЕКС} = 907\,200 + 199\,584 + 52\,500 + 16\,800 + 8361.49 + 100\,500 = 1\,284\,945.49 \text{ грн}.$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = \frac{С_{ЕКС}}{Т_{ЕФ}} = \frac{1\,284\,945.49}{1677.6} \approx 765.94 \text{ грн/год}.$$

Оскільки в даному випадку всі роботи, які пов’язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$См = См\text{-}г \cdot Т,$$

I. $См = 765.94 \cdot 606 = 464\ 159.64$ грн.

II. $См = 765.94 \cdot 770 = 589\ 773.80$ грн.

Накладні витрати складають 67% від заробітної плати: $СН = СЗП \cdot 0,67$

I. $С_Н = 907\ 200 \cdot 0,67 = 607\ 824$ грн.

II. $С_Н = 907\ 200 \cdot 0,67 = 607\ 824$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н,$$

I. $С_{ПП} = 907\ 200 + 199\ 584 + 462\ 465.72 + 607\ 824 = 2\ 178\ 767.64$ грн.

II. $С_{ПП} = 907\ 200 + 199\ 584 + 589\ 773.80 + 607\ 824 = 2\ 303\ 381.80$ грн.

4.7 Вибір оптимального варіанту реалізації ПП

У цьому розділі ми проведемо обчислення і визначимо найкращий варіант розробки програмного продукту за допомогою формули (4.17).

Коефіцієнт технічно-економічної рентабельності обчислюється за формулою:

$$КТЕРj = \frac{K_{Kj}}{C_{\varphi j}},$$

$$КТЕР1 = \frac{822\ 926.28}{2\ 177\ 073.72} \approx 0.378,$$

$$КТЕР2 = \frac{698\ 958.60}{2\ 301\ 041.4} \approx 0.304.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $KTEP1 = 0.378$. Після виконання функціонально-вартісного аналізу програмного комплексу що розробляється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $KTEP1 = 0.378$. Цей варіант реалізації програмного продукту має такі параметри: використання GPU для тренування і обрахунків, використання Isaac sim для виконання симуляції та ML Flow для логування, Python та фреймворку – PyTorch.

Висновки до розділу 4

Проведено повний функціонально-вартісний аналіз програмного продукту. Визначено та проведено оцінку основних функцій програмного продукту. Визначено параметри, які характеризують програмний продукт. Проведено експертне оцінювання параметрів та аналіз якості варіантів реалізації функцій. Проведено економічний аналіз варіантів розробки – трудомісткість, витрати на заробітну плату та інші витрати. На основі аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВКИ

У дипломній роботі досліджено, розроблено, експериментально обґрунтовано та проаналізовано комплексний підхід до автономного керування транспортним засобом у складних симульованих умовах з використанням сучасних методів штучного інтелекту, зокрема навчання з підкріпленням, самокерованого навчання, генеративного моделювання та мультиагентної взаємодії.

У результаті було поставлено задачу автономного керування у складному середовищі як задачу навчання з підкріпленням (MDP), з урахуванням впливу екстремальних погодних умов, змін дорожнього покриття, динамічного трафіку та мультиагентної взаємодії. У ході виконання роботи побудовано симуляційне середовище на базі Isaac Sim та Isaac Gym, що дозволяє реалізувати фізично достовірну динаміку транспортного засобу, підключати мультисенсорну систему (LiDAR, камери, IMU) та масштабовано тренувати кілька агентів одночасно. Реалізовано базовий агент з політикою PPO для керування транспортним засобом у простому середовищі, що забезпечив стабільне слідування траєкторії та уникнення перешкод. Також інтегровано підхід імітаційного навчання для попереднього тренування політики за демонстраціями, що дозволило пришвидшити і стабілізувати початкове навчання агента у складних умовах. Також запроваджено алгоритм HAPPO для мультиагентної координації, формалізовано взаємодію між агентами як диференціальну гру, та використано підхід CTDE. Це забезпечило узгодженість дій між кількома транспортними засобами в середовищі зі щільним трафіком, інкорпоровано сегментаційну модель у простір спостережень агента, що дозволило агенту використовувати інформацію про розмітку дороги для визначення своєї смуги та планування траєкторії.

У ході експериментальних дій розширено можливості агента за допомогою Meta-RL, що дало змогу агенту адаптувати свою політику до нових

дорожніх умов у реальному часі, зокрема у випадках різного зчеплення. Для конфігурації середовища застосовано генеративні моделі для симуляції рідкісних та критичних сценаріїв, зокрема аварійних ситуацій або обмеженої видимості, що дозволило значно підвищити стійкість політики до нетипових умов. З метою покращення представлень у спостереженнях, зокрема шляхом формування позитивних пар та попереднього тренування сенсорного стека, використано самокероване навчання, що підвищило ефективність кінцевої політики. Було сформовано комбіновану функцію винагороди, що враховує точність слідування траєкторії, безпеку, взаємодію з іншими агентами та дотримання правил руху. Це дозволило забезпечити збалансовану поведінку агента в умовах конфліктних цілей.

Результати моделювання та експериментів підтвердили ефективність запропонованого підходу: агент демонстрував стійку та адаптивну поведінку в широкому спектрі умов, у тому числі за змінного дорожнього покриття, при високій щільності трафіку та наявності несподіваних перешкод.

Запропоновані рішення мають практичну цінність для подальшого впровадження в системах автономного керування, розробки тестових платформ та досліджень у сфері штучного інтелекту для критичних сценаріїв руху.

ПЕРЕЛІК ПОСИЛАНЬ

1. InsideEVs. Waymo tops 200,000 autonomous rides as Tesla FSD expands [Електронний ресурс]. – Режим доступу: https://insideevs.com/news/752063/waymo-200000-autonomous-rides-tesla – Дата звернення: 20.04.2025.
2. Transport Scotland. Project CAVForth [Електронний ресурс]. – Режим доступу: https://www.transport.gov.scot/transport-network/roads/connected-and-autonomous-vehicles-cav/project-cavforth – Дата звернення: 20.04.2025.
3. AP News. Japan's Nissan tests driverless vehicles in city streets filled with cars and people [Електронний ресурс]. – Режим доступу: https://apnews.com/article/driverless-japan-nissan-autonomous-technology-5c12444c3931d1c7a0280789d2b0cba9 – Назва з екрана. – Дата звернення: 20.05.2025.AP News
4. Geotab. Why weather is a problem for autonomous vehicle safety [Електронний ресурс]. – Режим доступу: https://www.geotab.com/blog/autonomous-vehicles-safety-2/ – Дата звернення: 20.04.2025.
5. Lu H., et al. Enhancing Autonomous Vehicle Lateral Control: A Linear Complementarity Model-Predictive Control Approach // Applied Sciences. – 2023. – Vol. 13, No. 19. – P. 10809. – DOI: https://doi.org/10.3390/app131910809.
6. Zhang Y., et al. Autonomous Vehicle Decision-Making and Control in Complex and Uncertain Environments: A Review // Machines. – 2023. – Vol. 11, No. 7. – P. 676. – DOI: https://doi.org/10.3390/machines11070676.MDPI

7. Spiceworks. What is Markov Decision Process? [Електронний ресурс]. – Режим доступу: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-markov-decision-process/ – Дата звернення: 22.04.2025.
8. Bellman R. The Theory of Dynamic Programming // Bulletin of the American Mathematical Society. – 1954. – Vol. 60, No. 6. – P. 503–515. – DOI: https://doi.org/10.1090/S0002-9904-1954-09848-8.ams.org
9. GeeksforGeeks. Actor-Critic Algorithm in Reinforcement Learning [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/actor-critic-algorithm-in-reinforcement-learning/ – Дата звернення: 23.04.2025.
10. GeeksforGeeks. Q-Learning in Python [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/q-learning-in-python/ – Дата звернення: 27.04.2025.
11. IBM. What is a Multiagent System? [Електронний ресурс]. – Режим доступу: https://www.ibm.com/think/topics/multiagent-system – Дата звернення: 28.04.2025.
12. Zhang J., et al. Multi-Agent Reinforcement Learning-Based Resource Management for V2X Communication // Vehicular Communications. – 2023. – Vol. 39. – 100110. – DOI: https://doi.org/10.1016/j.vehcom.2023.100110.
13. Lanka V. Multi-Agent Reinforcement Learning (MARL) [Електронний ресурс]. – Режим доступу: https://vinaylanka.medium.com/multi-agent-reinforcement-learning-marl-1d55dfff6439 – Дата звернення: 29.04.2025.
14. Amin S. Deep Q-Learning (DQN) [Електронний ресурс]. – Режим доступу: https://medium.com/@samina.amin/deep-q-learning-dqn-71c109586bae – Дата звернення: 01.05.2025.
15. Danushi DK. PPO Algorithm [Електронний ресурс]. – Режим доступу: https://medium.com/@danushidk507/ppo-algorithm-3b33195de14a – Дата звернення: 01.05.2025.

16. Sciforce. Reinforcement Learning and Asynchronous Actor-Critic Agent (A3C) Algorithm Explained [Електронний ресурс]. – Режим доступу: https://medium.com/sciforce/reinforcement-learning-and-asynchronous-actor-critic-agent-a3c-algorithm-explained-f0f3146a14ab – Дата звернення: 01.05.2025.
17. Li Y., et al. Reinforcement Learning Journal: Volume 2024, Paper 99 [Електронний ресурс]. – Режим доступу: https://rlj.cs.umass.edu/2024/papers/RLJ_RLC_2024_99.pdf – Дата звернення: 02.05.2025.
18. Zhou X., et al. Multi-Agent Reinforcement Learning: A Survey // Annual Review of Control. – 2023. – Vol. 55. – P. 1–25. – DOI: https://doi.org/10.1146/annurev-control-090419-075625.
19. Chen Y., et al. Multi-Agent Reinforcement Learning in Autonomous Driving: A Survey // arXiv preprint arXiv:2411.00404v1. – 2024. – Режим доступу: https://arxiv.org/abs/2411.00404 – Дата звернення: 02.05.2025.
20. Wang L., et al. Safe Multi-Agent Reinforcement Learning: A Survey // arXiv preprint arXiv:2310.04241. – 2023. – Режим доступу: https://arxiv.org/abs/2310.04241 – Дата звернення: 02.05.2025.
21. Zhang T., et al. Generative Adversarial Imitation Learning in Multi-Agent Systems // arXiv preprint arXiv:2412.12098. – 2024. – Режим доступу: https://arxiv.org/abs/2412.12098 – Дата звернення: 04.05.2025.
22. Li H., et al. Federated Reinforcement Learning: Techniques, Applications, and Challenges // arXiv preprint arXiv:2004.04136. – 2020. – Режим доступу: https://arxiv.org/abs/2004.04136 – Дата звернення: 04.05.2025.
23. Gupta J., et al. Cooperative Multi-Agent Control Using Deep Reinforcement Learning // arXiv preprint arXiv:1807.03748. – 2018. – Режим доступу: https://arxiv.org/abs/1807.03748 – Дата звернення: 05.05.2025.
24. Zhao Y., et al. Multi-Agent Reinforcement Learning for Traffic Signal Control: A Review // arXiv preprint arXiv:2210.06609. – 2022. – Режим доступу: https://arxiv.org/abs/2210.06609 – Дата звернення: 05.05.2025.

25. Liu Z., et al. Multi-Agent Deep Reinforcement Learning for Large-Scale Traffic Signal Control // arXiv preprint arXiv:2503.15049. – 2025. – Режим доступу: https://arxiv.org/abs/2503.15049 – Дата звернення: 05.05.2025.
26. Chen M., et al. A Survey on Multi-Agent Reinforcement Learning Methods for Traffic Signal Control // arXiv preprint arXiv:2306.14131. – 2023. – Режим доступу: https://arxiv.org/abs/2306.14131 – Дата звернення: 05.05.2025.
27. DataCamp. Federated Learning: What is it and how does it work? [Електронний ресурс]. – Режим доступу: https://www.datacamp.com/blog/federated-learning – Дата звернення: 06.05.2025.
28. Zhang Y., et al. Multi-Agent Reinforcement Learning for Smart Grid Energy Management: A Review // arXiv preprint arXiv:2304.13004. – 2023. – Режим доступу: https://arxiv.org/abs/2304.13004 – Дата звернення: 06.05.2025.
29. Zhou M., et al. Decentralized Reinforcement Learning for Intelligent Transportation Systems // Transportation Engineering. – 2025. – Vol. 4. – 100098. – DOI: https://doi.org/10.1016/j.treng.2025.100098.
30. Zhou Y. I've Been Thinking About Multi-Agent Reinforcement Learning (MARL), and You Probably Should Be Too [Електронний ресурс]. – Режим доступу: https://towardsdatascience.com/ive-been-thinking-about-multi-agent-reinforcement-learning-marl-and-you-probably-should-be-too-8f1e241606ac/ – Дата звернення: 06.05.2025.
31. Nguyen T., et al. Multi-Agent Deep Reinforcement Learning: A Review of Challenges, Solutions and Applications // arXiv preprint arXiv:2009.13303. – 2020. – Режим доступу: https://arxiv.org/abs/2009.13303 – Дата звернення: 07.05.2025.

32. Towards Data Science. I’ve been thinking about Multi-Agent Reinforcement Learning (MARL), and you probably should be too [Електронний ресурс]. – Режим доступу: https://towardsdatascience.com/ive-been-thinking-about-multi-agent-reinforcement-learning-marl-and-you-probably-should-be-too-8f1e241606ac – – Дата звернення: 10.05.2025.
33. Da Silva, A. F., Pereira, D. F. M., Pereira, V. C. et al. SAFEGUARDING PRIVACY IN THE DIGITAL AGE: AN EXPLORATION OF TECHNIQUES FOR ANONYMIZING AND PROTECTING SENSITIVE DATA [Електронний ресурс] // ResearchGate, 2024. – Режим доступу: https://www.researchgate.net/publication/381295268_SAFEGUARDING_PRIVACY_IN_THE_DIGITAL_AGE_AN_EXPLORATION_OF_TECHNIQUES_FOR_ANONYMIZING_AND_PROTECTING_SENSITIVE_DATA – Дата звернення: 10.05.2025.
34. Hussen, M., et al. (2020). MARL for Large-Scale Traffic Signal Control Using Mean Field Theory [Електронний ресурс] // arXiv preprint arXiv:2009.13303. – Режим доступу: https://arxiv.org/pdf/2009.13303 – Дата звернення: 11.05.2025.
35. CGTrader. Sedan Car – Free 3D Model [Електронний ресурс]. – Режим доступу: https://www.cgtrader.com/free-3d-models/car/car/sedan-car-18b8f1bc-1ce2-45fa-a1fe-6adf67a8e620 – Дата звернення: 05.05.2025.
36. Dallas-City Skyline Free low-poly 3D model [Електронний ресурс]. – Режим доступу: https://www.cgtrader.com/free-3d-models/exterior/cityscape/dallas-city-skyline – Дата звернення: 06.05.2025.
37. City low poly Free low-poly 3D model [Електронний ресурс]. – Режим доступу: https://www.cgtrader.com/free-3d-models/exterior/cityscape/dronecity – Дата звернення: 06.05.2025.

38. Trash Bag Free low-poly 3D model [Електронний ресурс]. – Режим доступу: https://www.cgtrader.com/free-3d-models/exterior/street-exterior/trash-bag-4a59dbfc-95e7-49b0-bbaa-e2d4591a75f7 – Дата звернення: 07.05.2025.
39. Rocks three Free low-poly 3D model [Електронний ресурс]. – Режим доступу: https://www.cgtrader.com/free-3d-models/exterior/landscape/rocks-f8f45b59-ea1c-4af5-89ab-21fcc93635c2 – Дата звернення: 08.05.2025.
40. Fallen Tree Free low-poly 3D model [Електронний ресурс]. – Режим доступу: https://www.cgtrader.com/free-3d-models/exterior/other/fallen-tree-3e2624f2-7d59-4f40-b4db-221feeaaf458 – Дата звернення: 08.05.2025.
41. Traffic Road Cone - 3D VR AR game ready Free low-poly 3D model [Електронний ресурс]. – Режим доступу: https://www.cgtrader.com/free-3d-models/exterior/street-exterior/mg-aaa-traffic-road-cone – Дата звернення: 08.05.2025.

ДОДАТОК А ЛІСТИНГ ПРОГРАМНОГО КОДУ

Повна версія коду розміщена на Github за посиланням –

https://github.com/Viktoriagrg/car_isaac_rl

Метод генерації енвайромента:

```
def spawn(
        self,
        translations=[(0.0, 0.0, 0.5)],
        orientations=None,
        prim_paths: Sequence[str] = None
    ):
    if SimulationContext.instance()._physics_sim_view is not None:
        raise RuntimeError(
                                  "Cannot spawn vehicles after
simulation_context.reset() is called."
        )

    translations = torch.atleast_2d(
        torch.as_tensor(translations, device=self.device)
    )
    n = translations.shape[0]

    if orientations is None:
        orientations = [None for _ in range(n)]

    if prim_paths is None:
          prim_paths = [f"{TEMPLATE_PRIM_PATH}/{self.name}_{i}" for
i in range(n)]

    if not len(translations) == len(prim_paths):
          raise ValueError("Mismatched number of translations and
prim_paths.")

    prims = []
      for prim_path, translation, orientation in zip(prim_paths,
translations, orientations):
        if prim_utils.is_prim_path_valid(prim_path):
            raise RuntimeError(f"Duplicate prim at {prim_path}.")

        # Create the prim using a vehicle-specific method
          prim = self._create_vehicle_prim(prim_path, translation,
orientation)

        # Apply rigid body dynamics suitable for IsaacVehicle
        kit_utils.set_nested_rigid_body_properties(
            prim_path,
            linear_damping=self.rigid_props.linear_damping,
```

```
            angular_damping=self.rigid_props.angular_damping,

max_linear_velocity=self.rigid_props.max_linear_velocity,

max_angular_velocity=self.rigid_props.max_angular_velocity,

max_depenetration_velocity=self.rigid_props.max_depenetration_velo
city,
            enable_gyroscopic_forces=True,
            disable_gravity=self.rigid_props.disable_gravity,

retain_accelerations=self.rigid_props.retain_accelerations,
        )

        # Vehicle-specific initialization
        self._setup_vehicle_components(prim_path)

        prims.append(prim)

    self.n += n
    return prims
```

Метод збору даних, покрокове оновлення:

```
def compute_observations(vehicle):
    # Позиція
    position = vehicle.get_position()            # (x, y)

    # Швидкість
    velocity = vehicle.get_linear_velocity()     # v

    # Орієнтація (yaw кут)
    orientation = vehicle.get_yaw()              # θ

    # Відхилення від траєкторії
    e_y = compute_lateral_error(position)        # e_y
    e_theta = compute_heading_error(orientation) # e_θ

    # Відстань до цілі
    d_t = compute_target_distance(position)      # d_t

    # LiDAR
      lidar_distances = vehicle.lidar.get_normalized_readings()   #
lidar_distances

    # IMU
     imu_data = vehicle.imu.get_data()              # {accel: [ax,
ay, az], gyro: [gx, gy, gz]}

    # Камера (наприклад, RGB кадр або семантична сегментація)
       camera_image = vehicle.camera.capture()          # RGB або
сегментоване зображення (може бути знижене в розмірі)
```

```
    # Кутова швидкість
     omega = imu_data["gyro"][2]                    # ω – обертання
навколо осі Z

    # Прискорення (наприклад, уздовж x-ось)
    a_x = imu_data["accel"][0]                    # a_x

    # Оцінка зчеплення
    mu_est = estimate_friction(vehicle)           # μ_est

    # Буксування коліс
    slip_ratio = compute_slip_ratio(vehicle)      # slip_ratio

    # Ковзання
    side_slip_angle = compute_sideslip(vehicle)  # side_slip_angle

    observation_dict = {
        "position": position,
        "velocity": velocity,
        "orientation": orientation,
        "e_y": e_y,
        "e_theta": e_theta,
        "d_t": d_t,
        "lidar_distances": lidar_distances,
        "imu_accel": imu_data["accel"],
        "imu_gyro": imu_data["gyro"],
        "omega": omega,
        "a_x": a_x,
        "mu_est": mu_est,
        "slip_ratio": slip_ratio,
        "side_slip_angle": side_slip_angle,
           "camera_image": camera_image   # це може бути опційно,
напр., якщо політика CNN
    }
    return observation_dict
```

Методи для зміни умов середовища на початку кожного епізоду

```
def reset_environment_parameters(self, env_ids: torch.Tensor):
    """
    Оновлює параметри середовища перед кожним епізодом.
    """
    # Параметри тертя (сухий асфальт, мокрий, сніг, лід)
    friction_levels = torch.rand(len(env_ids), device=self.device)
    frictions = torch.where(
         friction_levels < 0.25, 0.9,                    # сухий
асфальт
        torch.where(friction_levels < 0.5, 0.6,        # мокрий
        torch.where(friction_levels < 0.75, 0.3, 0.1))# сніг / лід
    )
```

```
    # Параметри шорсткості (для візуального та фізичного впливу)
                roughness_levels    =    torch.rand(len(env_ids),
device=self.device)
     roughness = roughness_levels * 0.5   # 0.0 (гладка) до 0.5
(вибоїста)

    # Освітлення - від нічного до денного (0.0 ... 1.0)
    light_intensity = torch.rand(len(env_ids), device=self.device)

    # Дорожнє покриття (0: асфальт, 1: гравій, 2: бруд)
          road_types   =   torch.randint(0,   3,   (len(env_ids),),
device=self.device)

    # Випадкова поява інших агентів або перешкод
     spawn_traffic = torch.rand(len(env_ids), device=self.device) <
0.5
     spawn_obstacles = torch.rand(len(env_ids), device=self.device)
< 0.3

    for i, env_id in enumerate(env_ids):
        env_path = self.env_paths[env_id]

        # Налаштування тертя
        self.set_friction(env_path, friction=frictions[i].item())

        # Налаштування шорсткості
                                        self.set_roughness(env_path,
roughness=roughness[i].item())

        # Освітлення
                                         self.set_lighting(env_path,
intensity=light_intensity[i].item())

        # Тип дороги
                                    self.set_road_surface(env_path,
surface_type=road_types[i].item())

        # Інші транспортні засоби
        if spawn_traffic[i]:
            self.spawn_random_traffic(env_id)

        # Перешкоди
        if spawn_obstacles[i]:
            self.spawn_random_obstacles(env_id)

def set_friction(self, env_path: str, friction: float):
    prim = get_prim_at_path(env_path + "/ground")
          apply_physics_material(prim,   static_friction=friction,
dynamic_friction=friction)

def set_roughness(self, env_path: str, roughness: float):
    prim = get_prim_at_path(env_path + "/ground")
```

```
    set_surface_roughness(prim, roughness)

def set_lighting(self, env_path: str, intensity: float):
    light_prim = get_prim_at_path(env_path + "/light")
    light_prim.GetAttribute("intensity").Set(intensity * 10000.0)

def set_road_surface(self, env_path: str, surface_type: int):
    texture_map = {
        0: "Materials/Asphalt.usd",
        1: "Materials/Gravel.usd",
        2: "Materials/Mud.usd",
    }
                apply_surface_material(env_path    +    "/ground",
texture_map[surface_type])

def spawn_random_traffic(self, env_id: int):
    # Наприклад: додати 1-3 машини з випадковим рухом
    num_vehicles = torch.randint(1, 4, (1,)).item()
    for _ in range(num_vehicles):
                 self.spawn_vehicle(env_id,  random_position=True,
behavior="traffic")

def spawn_random_obstacles(self, env_id: int):
    # Наприклад: конуси, бар'єри або статичні об'єкти
    num_obstacles = torch.randint(1, 5, (1,)).item()
    for _ in range(num_obstacles):
        self.spawn_obstacle(env_id, random_position=True)
```

Архітектура сегментаційної моделі:

```
import torch
import torch.nn as nn
import torchvision.models as models
import torch.nn.functional as F

class ConvBlock(nn.Module):
    def __init__(self, in_ch, out_ch):
        super().__init__()
        self.block = nn.Sequential(
            nn.Conv2d(in_ch, out_ch, 3, padding=1),
            nn.BatchNorm2d(out_ch),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_ch, out_ch, 3, padding=1),
            nn.BatchNorm2d(out_ch),
            nn.ReLU(inplace=True),
        )

    def forward(self, x):
        return self.block(x)

class UpBlock(nn.Module):
    def __init__(self, in_ch, out_ch):
```

```
        super().__init__()
        self.up = nn.ConvTranspose2d(in_ch, out_ch, 2, stride=2)
        self.conv = ConvBlock(in_ch, out_ch)

    def forward(self, x, skip):
        x = self.up(x)
        # Padding, якщо розміри не співпадають
        diffY = skip.size()[2] - x.size()[2]
        diffX = skip.size()[3] - x.size()[3]
        x = F.pad(x, [diffX // 2, diffX - diffX // 2,
                      diffY // 2, diffY - diffY // 2])
        x = torch.cat([skip, x], dim=1)
        return self.conv(x)

class LaneSegmentationModel(nn.Module):
    def __init__(self, n_classes=4):  # фон, права, ліва, розмітка
        super().__init__()
        base_model = models.resnet18(pretrained=True)
                self.encoder1  =  nn.Sequential(base_model.conv1,
base_model.bn1, base_model.relu)
        self.pool1 = base_model.maxpool
        self.encoder2 = base_model.layer1
        self.encoder3 = base_model.layer2
        self.encoder4 = base_model.layer3
        self.encoder5 = base_model.layer4

        self.center = ConvBlock(512, 512)

        self.up4 = UpBlock(512 + 256, 256)
        self.up3 = UpBlock(256 + 128, 128)
        self.up2 = UpBlock(128 + 64, 64)
        self.up1 = UpBlock(64 + 64, 32)

        self.final = nn.Conv2d(32, n_classes, kernel_size=1)

    def forward(self, x):
        e1 = self.encoder1(x)
        e2 = self.encoder2(self.pool1(e1))
        e3 = self.encoder3(e2)
        e4 = self.encoder4(e3)
        e5 = self.encoder5(e4)

        center = self.center(e5)

        d4 = self.up4(center, e4)
        d3 = self.up3(d4, e3)
        d2 = self.up2(d3, e2)
        d1 = self.up1(d2, e1)

        out = self.final(d1)  # [B, 4, H, W]
        return out
```

Архітектура моделей генератора та дискримінатора:

```
import torch
import torch.nn as nn

class RoadMaskGenerator(nn.Module):
    def __init__(self, latent_dim=100, img_shape=(1, 64, 64)):
        super().__init__()
        self.init_size = img_shape[1] // 4
        self.fc = nn.Sequential(
            nn.Linear(latent_dim, 128 * self.init_size ** 2),
            nn.ReLU(True)
        )
        self.conv_blocks = nn.Sequential(
            nn.BatchNorm2d(128),
            nn.Upsample(scale_factor=2),  # 16 → 32
            nn.Conv2d(128, 128, 3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(True),

            nn.Upsample(scale_factor=2),  # 32 → 64
            nn.Conv2d(128, 64, 3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(True),
            nn.Conv2d(64, img_shape[0], 3, stride=1, padding=1),
            nn.Tanh()
        )

    def forward(self, z):
        out = self.fc(z)
              out  =  out.view(out.shape[0],  128,  self.init_size,
self.init_size)
        img = self.conv_blocks(out)
        return img

class RoadMaskDiscriminator(nn.Module):
    def __init__(self, img_shape=(1, 64, 64)):
        super().__init__()
        self.model = nn.Sequential(
            nn.Conv2d(img_shape[0], 64, 3, 2, 1),  # 64→32
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(64, 128, 3, 2, 1),  # 32→16
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(128, 256, 3, 2, 1),  # 16→8
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Flatten(),
            nn.Linear(256 * 8 * 8, 1),
```

```
            nn.Sigmoid()
        )

    def forward(self, img):
        return self.model(img)
```

Архітектура EncoderNet:

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class EncoderNet(nn.Module):
    def __init__(self, input_dim, hidden_dim=128, output_dim=64):
        super().__init__()
        self.mlp = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, output_dim)
        )

    def forward(self, x):
        return self.mlp(x)
```

Архітектура Projection Head для SSL:

```
class ProjectionHead(nn.Module):
    def __init__(self, input_dim, proj_dim=32):
        super().__init__()
        self.proj = nn.Sequential(
            nn.Linear(input_dim, input_dim),
            nn.ReLU(),
            nn.Linear(input_dim, proj_dim)
        )
    def forward(self, x):
        return self.proj(x)
```

Архітектура Contrastive Loss NT-Xent:

```
def nt_xent_loss(z_i, z_j, temperature=0.5):
    z_i = F.normalize(z_i, dim=1)
    z_j = F.normalize(z_j, dim=1)
    representations = torch.cat([z_i, z_j], dim=0)
    similarity = torch.matmul(representations, representations.T)
    batch_size = z_i.shape[0]
    labels   =   torch.cat([torch.arange(batch_size)   for   _   in
range(2)], dim=0)
    labels           =           (labels.unsqueeze(0)           ==
labels.unsqueeze(1)).float().to(z_i.device)
```

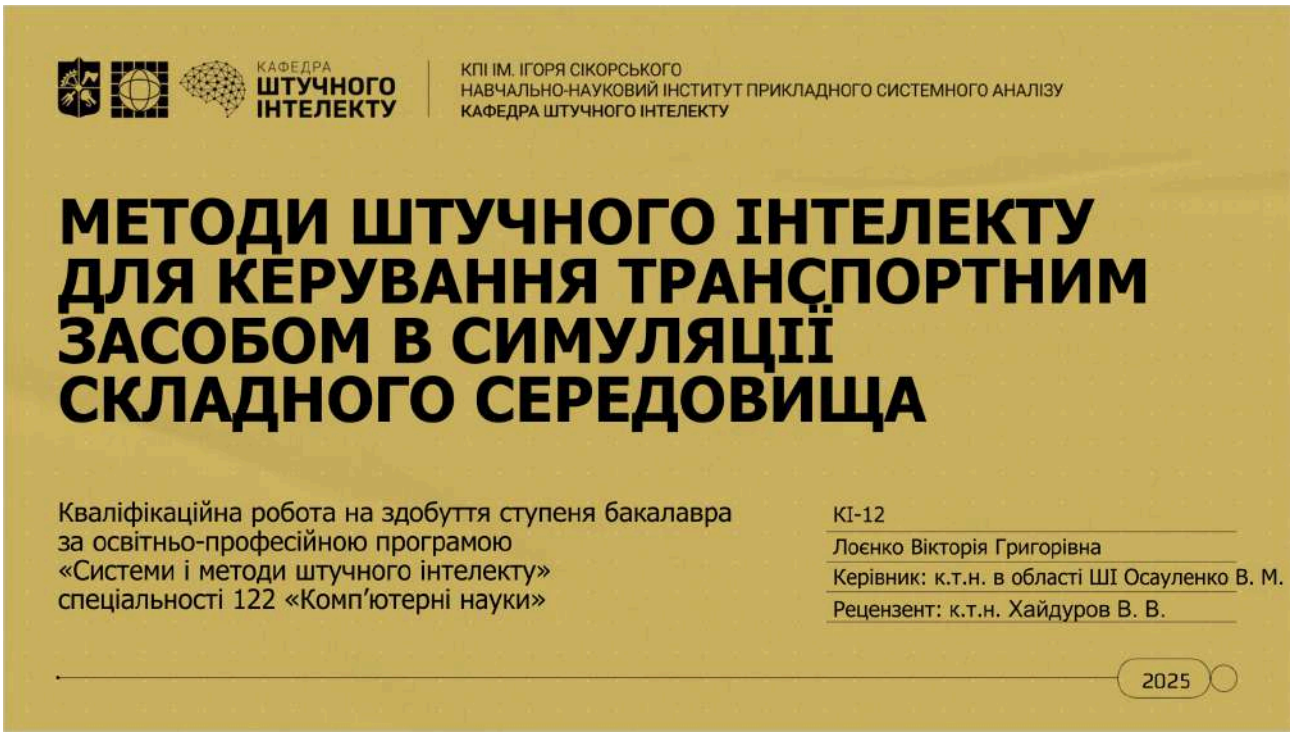
```
    mask                =                torch.eye(labels.shape[0],
dtype=torch.bool).to(z_i.device)
    logits = similarity / temperature
    logits = logits.masked_fill(mask, -1e9)
    labels = labels.masked_fill(mask, 0)
    loss = F.cross_entropy(logits, labels.argmax(dim=1))
    return loss
```

Політика PPO з інтегрованим енкодером:

```
class ActorCriticPolicy(nn.Module):
      def __init__(self, encoder: nn.Module, embedding_dim: int,
action_dim: int):
        super().__init__()
        self.encoder = encoder
        self.actor = nn.Sequential(
            nn.Linear(embedding_dim, 128),
            nn.ReLU(),
            nn.Linear(128, action_dim)
        )
        self.critic = nn.Sequential(
            nn.Linear(embedding_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 1)
        )
    def forward(self, obs):
        with torch.no_grad():  # freeze encoder
            emb = self.encoder(obs)

        action_logits = self.actor(emb)
        value = self.critic(emb)
        return action_logits, value
```

ДОДАТОК Б ЕЛЕКТРОННА ПРЕЗЕНТАЦІЯ



КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Методи штучного інтелекту для керування транспортним засобом в симуляції складного середовища

Актуальність, постановка задач та наукова новизна

Актуальність: автономне керування транспортними засобами у складних дорожніх умовах (дощ, ожеледиця, інтенсивний трафік) є однією з пріоритетних задач сучасної науки й техніки через зростаючу потребу в безпечних та ефективних системах транспорту. Розробка надійних алгоритмів автономного водіння має важливе соціальне значення, адже сприяє зменшенню аварійності, підвищенню мобільності і енергоефективності.

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Методи штучного інтелекту для керування транспортним засобом в симуляції складного середовища

Актуальність, постановка задач та наукова новизна

Постановка задач:

- Проаналізувати сучасні підходи до автономного керування транспортними засобами
- Розробити симуляційне середовище з фізично достовірною динамікою, мультисенсорною обробкою, та формалізацією задачі керування як марковського процесу прийняття рішень (MDP).
- Здійснити навчання та тест мультиагентних політик, адаптованих до складних умов, з використанням PPO, HAPPO та hybrid imitation + RL методів, із забезпеченням узагальнення на нові сценарії.
- Інтегрувати генеративні та self-supervised підходи для покращення адаптації агентів, генерації критичних сценаріїв та підвищення ефективності навчання.
- Провести аналіз ефективності розробленої системи та обґрунтувати можливість її впровадження у реальні або дослідницькі задачі.

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Методи штучного інтелекту для керування транспортним засобом в симуляції складного середовища

Актуальність, постановка задач та наукова новизна

Новизна

Компонент	Стандартний підхід	Новизна
RL	PPO / DQN	Комбіновані PPO + imitation + meta-RL + SSL
Сценарії	Фіксовані	Генерація через генеративні моделі + curriculum
Мультиагентність	Незалежне навчання	Диференціальні ігри
Адаптація	Offline policy	Онлайн-адаптація до умов
Репрезентації	Raw або supervised	Self-supervised learning

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Методи штучного інтелекту для керування транспортним засобом в симуляції складного середовища

Реалізовані методи:

Формалізація задачі:

- Markov Decision Process (MDP), стани, дії, винагороди
- Диференціальні ігри для взаємодії між агентами

Методи ШІ:

- Reinforcement Learning: PPO, HAPPO
- Meta-RL: адаптація до нових умов у реальному часі
- Curriculum Learning: поступове ускладнення
- Behavior Cloning + RL: демонстрації → RL-політика
- Self-supervised learning: покращення станів через pretraining
- Generative Models: VAE/GAN для критичних сценаріїв
- CTDE: централізоване тренування, децентралізоване виконання

Інженерні особливості:

- Об'єднання сенсорних даних (LiDAR, камери, IMU)
- Масштабоване тренування в Isaac Gym
- Комбінована функція винагороди

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Методи штучного інтелекту для керування транспортним засобом в симуляції складного середовища

Формалізація задачі як MDP

MDP визначається як п'ятірка:

$$\mathcal{M} = (S, A, P, R, \gamma)$$

де S — множина можливих **станів** агента (наприклад, положення, швидкість, дані з LiDAR, тощо),

A — множина **дій** (керування — рульове колесо, газ, гальмо),

$P(s_{t+1}|s_t,a_t)$ — **ймовірність переходу** до наступного стану s_{t+1} при виконанні дії a_t в стані s_t,

$R(s_t,a_t)\in R$ — **функція винагороди**, що повертає числову оцінку якості дії,

$\gamma\in[0,1]$ — **коефіцієнт дисконтування**, який визначає вплив майбутніх винагород.

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Методи штучного інтелекту для керування транспортним засобом в симуляції складного середовища

Формалізація задачі як MDP

У MDP агент формує стратегію (політику) – правило, яке визначає, яку дію обрати в кожному стані. Політика (policy) π(s) – функція, яка зіставляє кожному стану s певну дію a або розподіл імовірностей дій

Метою агента є знайти таку ***стохастичну політику*** π(at|st), яка максимізує очікувану сумарну винагороду:

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \right]$$

Тобто оптимальна політика:

$$\pi^* = \arg\max_\pi J(\pi)$$

Процес є марківським, якщо перехід у наступний стан st+1 залежить лише від поточного стану st і дії at, але не від попередніх станів:

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Методи штучного інтелекту для керування транспортним засобом в симуляції складного середовища

Конфігурація тренування в простому середовищі

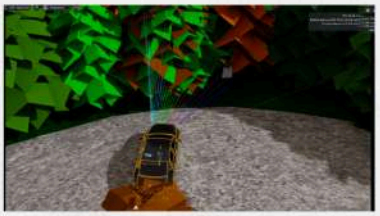
Стан (R^7):

Позиція (x,y) — глобальні координати
Швидкість v — скаляр
Орієнтація θ — кут повороту
Відхилення від траєкторії e_y
Кутова помилка e_θ
Відстань до цілі d_t

Дії (R^2):

steering ∈ [−1, 1] — кут повороту керма
throttle ∈ [−1, 1] — газ / гальмо

$$\text{reward} = \left(-w_d \cdot d_t - w_{e_y} \cdot |e_y| - w_{e_\theta} \cdot |e_\theta| + w_v \cdot v - w_{oob} \cdot \text{is_out_of_bounds} + w_{\text{goal}} \cdot \text{goal_reached}\right)$$

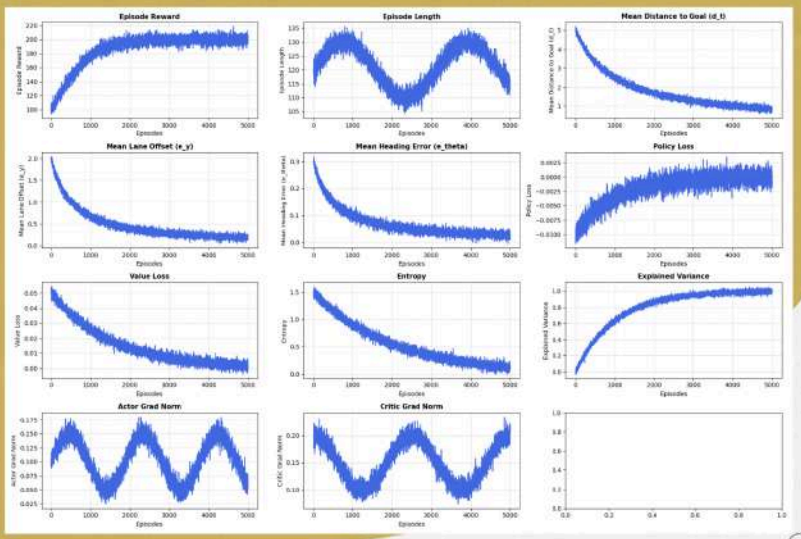


Умови дострокового завершення епізоду:

Виїзд за межі поля
Агент зупинився / перевернувся
Занадто довге проходження епізоду
Повне відхилення від смуги

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Результати тренування попередньо описаної примітивної конфігурації: 1 середовище 5000 епох



КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Методи штучного інтелекту для керування транспортним засобом в симуляції складного середовища

Конфігурація тренування в складному середовищі

Multi-Agent Training з HAPPO

HAPPO (Heterogeneous-Agent Proximal Policy Optimization):

Централізований критик, локальні політики
Підтримка різних агентів (гетерогенність)
Масштабованість до складного трафіку

Стан (наведено в таблиці)

Сумарна розмірність стану ≈N_lidar + 1 (v) + 1 (θ) + 1 (d_target) + 2 (v_target) + 1 + 4*K = 578

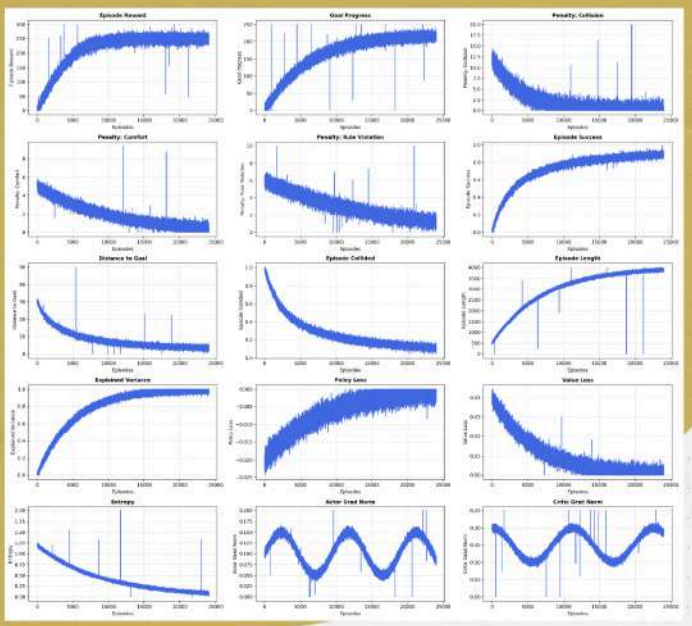
Під час навчання критик може бачити додаткову інформацію про всіх агентів: стани всіх агентів, дії всіх агентів, маски видимості (чи доступна інформація про агента j для агента i) Це дозволяє використовувати centralized value function без порушення децентралізованого виконання.

Компонент	Опис	Розмірність
LiDAR/дальність до об'єктів	Радарні або лідарні дані (кутовий сектор з дискретизацією)	N_lidar (64)
Власна швидкість	Швидкість руху вперед/назад	1
Орієнтація	Кутова помилка відносно бажаного курсу	1
Відстань до цілі	Евклідова відстань до цільової точки	1
Вектор до цілі	Вектор напрямку до таргету у локальній системі координат	2
Прогрес по маршруту	Пропорційне значення прогресу по заданій траєкторії	1
Наявність інших агентів	Відносна позиція та швидкість сусідніх агентів (в радіусі R)	K (127) × (x, y, vx, vy)

11

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Результати тренування попередньо описаної складної конфігурації: 128 середовищ 25000 епох



12

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Методи штучного інтелекту для керування транспортним засобом в симуляції складного середовища

Висновки

У процесі роботи було розроблено модульне симуляційне середовище для тренування автономних транспортних засобів в умовах низького зчеплення, складного рельєфу та динамічних загроз. Середовище підтримує мультисенсорну обробку та достовірну фізику, а задача керування формалізована як марковський процес прийняття рішень. Запропоновано комбінований підхід, що поєднує reinforcement learning, imitation learning, meta-RL та self-supervised learning, що дало змогу досягти гнучкої, адаптивної та стійкої поведінки агентів у складних сценаріях. Проведено навчання мультиагентних політик з використанням PPO, HAPPO та hybrid imitation + RL підходів із централізованим тренуванням і децентралізованим виконанням (CTDE). Для підвищення стійкості політик реалізовано генерацію складних сценаріїв за допомогою генеративних моделей (VAE, GAN), а також впроваджено self-supervised методи для покращення адаптації агентів. Розроблена функція винагороди враховує фактори безпеки, комфорту та ефективності руху. Створена архітектура є масштабованою, придатною до узагальнення на нові умови та може бути використана як у дослідницьких задачах, так і в реальних застосуваннях автономного транспорту.

13

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

КПІ ІМ. ІГОРЯ СІКОРСЬКОГО
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Методи штучного інтелекту для керування транспортним засобом в симуляції складного середовища

ДЯКУЮ ЗА УВАГУ!

Кваліфікаційна робота на здобуття ступеня бакалавра
за освітньо-професійною програмою
«Системи і методи штучного інтелекту»
спеціальності 122 «Комп'ютерні науки»

КІ-12
Лоєнко Вікторія Григорівна
Керівник: к.т.н. в області ШІ Осауленко В. М.
Рецензент: к.т.н. Хайдуров В. В.

2025