

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____ Комп'ютерних наук
(повна назва)
Кафедра _____ Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _____ перший (бакалаврський)

Мультиагентна система для оптимізації навантаження громадського транспорту
на основі підкріплювального навчання
(тема)

Виконав:
здобувач _____ четвертого _____ року навчання,
групи _____ ІТШ-22-2

_____ Єлисей ЗАВАДА
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми _____ освітньо-професійна

Освітня програма _____ Штучний інтелект
(повна назва освітньої програми)

Керівник _____ ас. Дмитро МАЛЄЄВ
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ

(підпис)

_____ Лариса ЧАЛА
(власне ім'я, прізвище)

2026 р.

Харківський національний університет радіоелектроніки

Факультет _____ Комп'ютерних наук _____

Кафедра _____ Штучного інтелекту _____

Рівень вищої освіти _____ перший (бакалаврський) _____

Спеціальність _____ 122 Комп'ютерні науки _____
(код і повна назва)

Тип програми _____ освітньо-професійна _____

Освітня програма _____ Штучний інтелект _____
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

« _____ » _____ 20 ____ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві _____ Заваді Єлисею Олександровичу _____
(прізвище, ім'я, по батькові)

1. Тема роботи Мультиагентна система для оптимізації навантаження громадського транспорту на основі підкріплювального навчання

затверджена наказом університету від 5 червня 2026 р. № 488Ст

2. Термін подання студентом роботи до екзаменаційної комісії 8 липня 2026 р.

3. Вихідні дані до роботи вихідні дані міської транспортної мережі Харкова; згенеровані дані пасажиропотоків; мова програмування Python; фреймворк FastAPI; бібліотеки PyTorch, NumPy та Gymnasium; клієнтська частина React; алгоритми Q-learning, DQN, Policy Gradient та Actor-Critic; технологія WebSocket; СУБД PostgreSQL

4. Перелік питань, що потрібно опрацювати в роботі _____

1) Аналіз предметної галузі _____

2) Технологічний та прикладний опис розробки _____

3) Реалізація та тестування системи _____

КАЛЕНДАРНИЙ ПЛАН

[illegible]

Дата видачі завдання 8 червня 20 26 р.

Здобувач _____
(підпис)

Керівник роботи _____
(підпис)

ас. Дмитро МАЛЄСВ
(посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 84 с., 27 рис., 3 табл., 2 дод., 15 джерел.

ГРОМАДСЬКИЙ ТРАНСПОРТ, ІНТЕЛЕКТУАЛЬНІ ТРАНСПОРТНІ СИСТЕМИ, МУЛЬТИАГЕНТНА СИСТЕМА, ОПТИМІЗАЦІЯ ТРАНСПОРТНИХ ПОТОКІВ, ПІДКРІПЛЮВАЛЬНЕ НАВЧАННЯ, ШТУЧНИЙ ІНТЕЛЕКТ, DQN, FASTAPI, PYTORCH, Q-LEARNING, REACT.

Об'єкт дослідження – процес управління навантаженням громадського транспорту.

Предмет дослідження – методи мультиагентного підкріплювального навчання для оптимізації транспортних потоків.

Мета роботи – розробка мультиагентної системи оптимізації навантаження громадського транспорту на основі підкріплювального навчання.

Методи дослідження – машинне навчання, підкріплювальне навчання, мультиагентне моделювання, аналіз транспортних потоків.

У роботі розроблено мультиагентну систему для оптимізації роботи громадського транспорту. Реалізовано середовище симуляції транспортної мережі та алгоритми Q-learning, DQN, Policy Gradient і Actor-Critic для навчання агентів прийняттю ефективних рішень у динамічному середовищі. Серверну частину системи реалізовано на FastAPI, а клієнтський інтерфейс – на React. Проведено тестування системи та оцінку ефективності роботи агентів в умовах змінного транспортного навантаження.

ABSTRACT

Bachelor's thesis contains: 84 pp., 27 fig., 3 tabl., 2 ann., 15 references.

ARTIFICIAL INTELLIGENCE, DQN, FASTAPI, INTELLIGENT TRANSPORT SYSTEMS, MULTI-AGENT SYSTEM, OPTIMIZATION OF TRANSPORT FLOWS, PUBLIC TRANSPORT, PYTORCH, Q-LEARNING, REACT, REINFORCEMENT LEARNING.

Object of research – the process of managing public transport load.

Subject of research – methods of multi-agent reinforcement learning for transport flow optimization.

Purpose of the work – development of a multi-agent system for optimizing public transport load based on reinforcement learning.

Research methods – machine learning, reinforcement learning, multi-agent modeling, transport flow analysis.

The paper presents a multi-agent system for optimizing public transport operation. A transport network simulation environment and Q-learning, DQN, Policy Gradient, and Actor-Critic algorithms were implemented for training agents to make efficient decisions in a dynamic environment. The server side of the system was implemented using FastAPI, while the client interface was developed with React. System testing and evaluation of agent performance under changing transport load conditions were carried out.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів	8
Вступ.....	9
1 Аналіз предметної галузі	11
1.1 Система громадського транспорту	11
1.2 Нерівномірне навантаження транспортних мереж.....	13
1.3 Методи оптимізації транспортних потоків	16
1.4 Штучний інтелект у транспортних системах	19
1.5 Мультиагентні системи в задачах транспортного управління	21
1.6 Підкріплювальне навчання в задачах транспортного управління	23
2 Технологічний та прикладний опис розробки	26
2.1 Архітектура мультиагентних систем	26
2.2 Алгоритми машинного навчання для задач оптимізації.....	28
2.2.1 Q-learning	28
2.2.2 Deep Q-Network (DQN).....	31
2.2.3 Policy Gradient	33
2.2.4 Actor-Critic	35
2.2.5 Multi-Agent Reinforcement Learning (MARL).....	37
2.3 Методи моделювання транспортних систем.....	39
2.3.1 Агентне моделювання	39
2.3.2 Моделювання пасажиропотоку	41
2.3.3 Імітаційне моделювання транспортної мережі	42
2.4 Вибір технологій для реалізації системи	45
2.4.1 Мови програмування та бібліотеки.....	45
2.4.2 Фреймворки для підкріплювального навчання	45
2.4.3 Інструменти моделювання транспортних систем	46
3 Реалізація та тестування системи	48
3.1 Реалізація середовища та агентів	48
3.1.1 Граф транспортної мережі та пасажиропотоки	48

3.1.2 Простір станів та дій агента	51
3.1.3 Реалізація агентів системи	54
3.2 Реалізація алгоритмів навчання.....	59
3.2.1 Реалізація Q-learning та DQN.....	59
3.2.2 Реалізація Policy Gradient та Actor-Critic.....	61
3.2.3 MARL-тренер та процес навчання моделей.....	63
3.2.4 Порівняння реалізованих алгоритмів	65
3.3 Реалізація серверної та клієнтської частини	66
3.3.1 REST API та WebSocket-сервер.....	66
3.3.2 Клієнтська частина на React	69
3.4 Реалізація серверної та клієнтської частини	72
Висновки	74
Перелік джерел посилання	76
Додаток А Користувацький інтерфейс та аналіз	78
Додаток Б Відомість кваліфікаційної роботи.....	84

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних;

МАС – мультиагентна система;

ПН – підкріплювальне навчання;

ШІ – штучний інтелект;

АС – Actor-Critic – алгоритм Actor-Critic;

API – Application Programming Interface – інтерфейс програмування застосунків;

DQN – Deep Q-Network – глибока Q-мережа;

FPS – Frames Per Second – кількість кадрів за секунду;

GPU – Graphics Processing Unit – графічний процесор;

HTTP – Hypertext Transfer Protocol – протокол передачі гіпертексту;

JSON – JavaScript Object Notation – формат обміну даними;

ML – Machine Learning – машинне навчання;

PG – Policy Gradient – градієнт політики;

Q-learning – алгоритм Q-навчання;

REST – Representational State Transfer – архітектурний стиль побудови API;

RL – Reinforcement Learning – підкріплювальне навчання;

SQL – Structured Query Language – мова структурованих запитів;

UI – User Interface – користувацький інтерфейс;

URL – Uniform Resource Locator – уніфікований локатор ресурсів;

UX – User Experience – користувацький досвід.

ВСТУП

Громадський транспорт є невід’ємною складовою міської інфраструктури, що забезпечує щоденну мобільність мільйонів людей. Однак нерівномірний розподіл пасажиропотоків, явище зграювання транспортних засобів та неефективне використання рухомого складу залишаються актуальними проблемами більшості міських транспортних систем. Традиційні методи управління на основі фіксованих розкладів не здатні адаптуватися до динамічних змін попиту в режимі реального часу, що зумовлює необхідність пошуку нових підходів до оптимізації транспортних систем.

Перспективним напрямом вирішення цих проблем є застосування мультиагентних систем і алгоритмів підкріплювального навчання, які дозволяють автономним агентам самостійно виробляти оптимальні стратегії поведінки шляхом взаємодії із середовищем. На відміну від класичних методів оптимізації, підкріплювальне навчання не вимагає явного задання правил поведінки і здатне адаптуватися до нових умов без повторного програмування системи.

Метою кваліфікаційної роботи є розробка мультиагентної системи оптимізації навантаження громадського транспорту на основі алгоритмів підкріплювального навчання. Для досягнення поставленої мети вирішено такі завдання:

- аналіз існуючих підходів до оптимізації транспортних систем та методів підкріплювального навчання;
- проєктування архітектури мультиагентної системи з визначенням простору станів, дій та функції винагороди;
- реалізація середовища симуляції транспортної мережі на основі реальних даних міста Харкова;
- реалізація алгоритмів Q-learning, DQN, Policy Gradient та Actor-Critic; розробка серверної частини на базі FastAPI та клієнтського

інтерфейсу на React; тестування системи та оцінка ефективності навчених агентів.

Об'єктом дослідження є процес управління навантаженням громадського транспорту в умовах динамічного пасажиропотоку. Предметом дослідження є методи мультиагентного підкріплювального навчання для оптимізації розподілу транспортних ресурсів. Практична цінність роботи полягає у розробці програмного комплексу що дозволяє навчати агентів управляти маршрутами громадського транспорту та візуалізувати результати симуляції в режимі реального часу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Система громадського транспорту

Громадський транспорт є одним із ключових елементів міської інфраструктури, що забезпечує мобільність населення та підтримує функціонування економічної й соціальної системи міста. У великих урбанізованих центрах він виконує не лише транспортну функцію, але й виступає важливим фактором формування міського середовища. Завдяки громадському транспорту мешканці отримують можливість переміщатися між різними районами міста без необхідності використання приватного автомобіля, що сприяє зменшенню транспортних заторів та оптимізації використання дорожньої мережі [1].

До складу систем громадського транспорту входять різні види перевезень, серед яких автобуси, трамваї, тролейбуси, метрополітен та приміські електропоїзди. Кожен із цих видів транспорту має власні технічні характеристики, рівень місткості та специфіку експлуатації.

Наприклад, автобуси відзначаються високою гнучкістю маршрутної мережі, оскільки можуть легко змінювати маршрути відповідно до потреб міста. Метрополітен, навпаки, характеризується високою пропускною здатністю та стабільністю руху, проте потребує значних витрат на будівництво та обслуговування інфраструктури.

Організація роботи громадського транспорту базується на поєднанні маршрутної мережі, розкладу руху та системи управління транспортними потоками. Маршрутна мережа визначає зв'язки між різними районами міста і формує основу транспортної доступності для населення. Від того, наскільки раціонально побудована така мережа, залежить ефективність транспортних перевезень та рівень задоволеності пасажирів. Непродумана структура маршрутів може призводити до перевантаження окремих

напрямків або, навпаки, до недостатнього використання транспортних ресурсів.

Важливим фактором функціонування транспортної системи є інтенсивність пасажиропотоку. Пасажиропотік відображає кількість людей, які користуються транспортом на певному маршруті або у визначений проміжок часу.

У більшості міст пасажиропотоки мають виражену часову нерівномірність. У ранкові години велика кількість людей переміщується до місць роботи та навчання, а ввечері відбувається зворотний рух до житлових районів. Коливання створюють суттєві труднощі для організації транспортного обслуговування, адже система повинна забезпечувати достатню пропускну здатність у години пікового навантаження.

Окрім часових коливань, пасажиропотоки мають і просторову нерівномірність. У центральних частинах міст транспортне навантаження значно вище, ніж у периферійних районах. Це пояснюється концентрацією ділової активності, торговельних центрів та адміністративних установ у центральних районах. Внаслідок цього транспортні маршрути, що проходять через центр міста, часто працюють із перевантаженням, тоді як інші напрямки залишаються недостатньо завантаженими [2].

Ефективність громадського транспорту значною мірою залежить від частоти руху транспортних засобів. Частота руху визначає інтервал часу між прибуттям транспортних засобів на зупинку. Занадто великі інтервали призводять до накопичення пасажирів на зупинках, що створює дискомфорт та збільшує час очікування. У той самий час надмірне збільшення кількості транспортних засобів може спричинити нераціональне використання ресурсів транспортної системи.

Ще одним важливим аспектом є координація між різними видами транспорту. У сучасних містах дедалі частіше застосовується принцип інтегрованих транспортних систем, коли автобусні, трамвайні та метрополітенні маршрути працюють у взаємодії між собою, дозволяючи

організовувати зручні пересадки та скорочувати загальний час подорожі пасажирів. Добре організована система пересадок підвищує ефективність транспортної мережі і робить її більш зручною для користувачів.

З розвитком цифрових технологій у транспортній сфері почали активно впроваджуватися інтелектуальні транспортні системи. Системи використовують дані з датчиків, систем позиціонування та мобільних додатків для аналізу транспортних потоків.

Завдяки цьому стає можливим отримувати актуальну інформацію про завантаженість маршрутів, час прибуття транспорту та зміни у пасажиропотоках. Отримані дані можуть застосовуватися для вдосконалення планування маршрутів і підвищення ефективності управління транспортною мережею [3].

Зростання міст і збільшення кількості пасажирів створюють нові виклики для транспортних систем. Статичні моделі планування маршрутів, які використовувалися раніше, не завжди здатні ефективно реагувати на динамічні зміни попиту на перевезення. Зміни у транспортній інфраструктурі, дорожні інциденти або масові події можуть суттєво впливати на роботу транспорту.

Ці фактори стимулюють пошук нових підходів до управління транспортними системами. Одним із перспективних напрямів досліджень є використання методів штучного інтелекту, які здатні адаптуватися до змін середовища та оптимізувати роботу транспортної мережі на основі аналізу великих обсягів даних.

1.2 Нерівномірне навантаження транспортних мереж

Однією з найбільш поширених проблем функціонування систем громадського транспорту є нерівномірний розподіл транспортного навантаження. У межах міської транспортної мережі різні маршрути та напрямки можуть мати суттєво відмінні рівні пасажиропотоку.

У результаті на одних маршрутах спостерігається перевантаження транспортних засобів, тоді як на інших транспорт працює з низьким рівнем заповнення, знижуючи ефективність використання транспортних ресурсів і негативно впливає на якість обслуговування пасажирів [1].

Нерівномірність навантаження пов'язана передусім із особливостями просторової структури міста. У більшості міст ділова активність, адміністративні установи та великі торговельні центри зосереджені у центральних районах. Водночас значна частина населення проживає на периферії міста або в спальних районах. У ранкові години велика кількість людей прямує до центру міста, а у вечірній час спостерігається зворотний рух. Подібні коливання пасажиропотоку створюють значне навантаження на транспортну систему, особливо на основні магістральні маршрути.

Навантаження на транспортні засоби також залежить від часу доби. У транспортній системі більшості міст можна виділити кілька характерних періодів: ранкову годину пік, денний період відносно стабільного руху, вечірню годину пік та нічний період з мінімальним попитом на перевезення. Під час пікових навантажень пасажирів часто стикаються з переповненими автобусами, трамваями або вагонами метро, що знижує комфорт перевезень і може призводити до затримок у роботі транспортної мережі.

Нерівномірність навантаження проявляється не лише у часовому, але й у просторовому вимірі. Деякі транспортні маршрути проходять через райони з високою щільністю населення або великою кількістю робочих місць.

Такі маршрути мають стабільно високий рівень завантаження. Інші маршрути можуть обслуговувати менш населені райони або мати обмежену кількість пасажирів. У результаті транспортні засоби на різних маршрутах використовуються з різною ефективністю.

Додатковим фактором, що впливає на нерівномірність транспортного навантаження, є особливості планування маршрутної мережі. У багатьох містах маршрути громадського транспорту формувалися поступово

протягом тривалого часу. Часто вони створювалися для вирішення локальних транспортних задач без урахування комплексної оптимізації всієї транспортної системи, що призвело до того, що окремі ділянки транспортної мережі мають надмірну кількість маршрутів, тоді як інші райони залишаються недостатньо забезпеченими транспортним сполученням.

Значний вплив на розподіл транспортного навантаження має і поведінка пасажирів. Люди прагнуть обирати найшвидший або найзручніший спосіб пересування, що призводить до концентрації пасажиропотоку на окремих маршрутах.

Наприклад, швидкісні трамвайні лінії або маршрути метрополітену часто мають значно більший попит, ніж альтернативні види транспорту. Подібні дисбаланси у виборі маршрутів можуть створювати додаткові труднощі для організації транспортного обслуговування [2].

Традиційні підходи до управління транспортними системами базуються на фіксованих розкладах руху та статистичних даних про пасажиропотоки, дозволяючи планувати роботу транспорту на основі середніх значень попиту. Однак вони не враховують динамічні зміни транспортної ситуації, які можуть виникати протягом дня. Наприклад, дорожні затори, аварії або масові заходи можуть суттєво змінювати структуру транспортних потоків.

Складність транспортної системи міста значною мірою ускладнює процес її оптимізації. Кількість маршрутів, транспортних засобів та пасажирів створює велику кількість взаємопов'язаних факторів, які впливають на загальну ефективність системи. Навіть незначні зміни в одному елементі транспортної мережі можуть призводити до суттєвих змін у роботі інших її частин.

Зростання міст і збільшення пасажиропотоків посилюють проблему нерівномірного навантаження транспортних систем. У сучасних умовах виникає потреба у більш гнучких методах управління транспортною

мережею, які здатні адаптуватися до змін попиту на перевезення та ефективно розподіляти транспортні ресурси.

Серед перспективних підходів до вирішення цієї проблеми розглядається застосування інтелектуальних методів управління транспортом, зокрема мультиагентних систем і алгоритмів підкріплювального навчання, дозволяючи моделювати поведінку окремих елементів транспортної системи та знаходити оптимальні стратегії управління транспортними потоками в умовах динамічного середовища [3].

1.3 Методи оптимізації транспортних потоків

Оптимізація транспортних потоків є важливою складовою ефективного функціонування міських транспортних систем. У сучасних містах зростання кількості пасажирів і транспортних засобів призводить до збільшення навантаження на транспортну інфраструктуру. Це створює необхідність застосування різних методів планування та оптимізації транспортних потоків, спрямованих на підвищення ефективності перевезень і зменшення перевантаження транспортної мережі [4].

У традиційних транспортних дослідженнях транспортна система часто розглядається як мережа, що складається з вузлів і зв'язків між ними. Вузли відповідають транспортним зупинкам, пересадковим пунктам або станціям, тоді як зв'язки відображають маршрути транспортних засобів. Подібне представлення дозволяє застосовувати математичні методи аналізу мереж для дослідження транспортних потоків.

Серед класичних методів оптимізації транспортних систем можна виділити кілька основних підходів:

- математичні моделі транспортних мереж, що дозволяють оцінювати навантаження на різні ділянки мережі;
- методи транспортного планування, які використовують статистичні дані про пасажиропотоки;

- алгоритми пошуку оптимальних маршрутів, що визначають найкоротші або найшвидші шляхи переміщення;

- методи оптимального розподілу транспортних ресурсів, які враховують кількість транспортних засобів та пасажирів.

Одним із найбільш поширених напрямів є використання алгоритмів пошуку найкоротших шляхів у транспортній мережі, дозволяючи визначати оптимальні маршрути для пасажирів або транспортних засобів. Найбільш відомими серед них є:

- алгоритм Дейкстри;
- алгоритм Беллмана-Форда;
- алгоритм A^* .

Вони широко застосовуються в навігаційних системах і транспортному плануванні, оскільки дозволяють швидко знаходити оптимальні маршрути в складних транспортних мережах.

Іншим важливим підходом є використання методів математичної оптимізації. У межах таких методів задача управління транспортною системою формулюється у вигляді математичної моделі, що містить цільову функцію та систему обмежень. Наприклад, цільова функція може полягати у:

- мінімізації сумарного часу перевезення пасажирів;
- зменшенні експлуатаційних витрат транспортної системи;
- рівномірному розподілі транспортного навантаження між маршрутами.

Подібні моделі часто розв'язуються за допомогою методів лінійного або нелінійного програмування.

З розвитком комп'ютерного моделювання значного поширення набули імітаційні моделі транспортних систем. Імітаційне моделювання дозволяє створювати комп'ютерні моделі міських транспортних мереж і досліджувати їхню поведінку в різних умовах.

Наприклад, можна моделювати зміну пасажиропотоків, вплив дорожніх заторів або зміну розкладу руху транспорту. Це дає змогу оцінити ефективність різних варіантів організації транспортної системи без необхідності змінювати реальну інфраструктуру [5].

Окремим напрямом є використання агентного моделювання. У таких моделях транспортна система розглядається як сукупність взаємодіючих агентів. До агентів можуть належати:

- транспортні засоби;
- пасажирів;
- елементи транспортної інфраструктури;
- системи управління транспортом.

Кожен агент має власні правила поведінки та може приймати рішення залежно від умов середовища, дозволяючи моделювати складні транспортні процеси та досліджувати взаємодію між різними компонентами транспортної системи.

Попри значну кількість існуючих методів оптимізації, більшість класичних підходів мають певні обмеження. Багато з них базуються на статичних моделях і не враховують динамічний характер транспортних потоків. У реальних умовах транспортна ситуація може швидко змінюватися через дорожні затори, погодні умови або інші фактори.

У зв'язку з цим сучасні дослідження дедалі частіше спрямовані на використання методів штучного інтелекту для управління транспортними системами. Підходи дозволяють аналізувати великі обсяги даних, адаптуватися до змін середовища та знаходити ефективні стратегії управління транспортними потоками. Особливий інтерес викликає застосування мультиагентних систем та алгоритмів підкріплювального навчання, які здатні самостійно знаходити оптимальні рішення у складних динамічних системах [6].

1.4 Штучний інтелект у транспортних системах

Упродовж останніх десятиліть транспортні системи міст стають дедалі складнішими. Зростання населення, збільшення кількості поїздок і розширення міських територій призводять до значного підвищення навантаження на транспортну інфраструктуру.

Традиційні методи управління транспортними системами, що базуються на статичних розкладах руху та попередньо встановлених правилах, часто не здатні ефективно реагувати на динамічні зміни транспортної ситуації. У зв'язку з цим у транспортних дослідженнях все активніше застосовуються методи штучного інтелекту, які дозволяють автоматизувати процес аналізу даних і прийняття рішень [7].

Штучний інтелект охоплює широкий спектр підходів і алгоритмів, що дозволяють комп'ютерним системам виконувати задачі, які традиційно потребують людського інтелекту. У контексті транспортних систем ці технології використовуються для аналізу транспортних потоків, прогнозування пасажиропотоків, оптимізації маршрутів і управління дорожнім рухом.

Завдяки розвитку цифрових технологій транспортні системи отримують доступ до великої кількості даних, які можуть бути використані для побудови інтелектуальних моделей управління.

Важливим джерелом інформації для таких систем є дані, що надходять із різних технологічних платформ. До них належать GPS-дані транспортних засобів, інформація з мобільних додатків пасажирів, дані систем електронної оплати проїзду, а також інформація з дорожніх датчиків та камер спостереження. Їх аналіз дозволяє отримувати детальну картину функціонування транспортної системи та виявляти закономірності у поведінці пасажирів і транспортних потоків.

У сучасних дослідженнях штучний інтелект використовується для вирішення низки задач у транспортній сфері. Найбільш поширеними напрямками застосування є:

- прогнозування пасажиропотоків на транспортних маршрутах;
- аналіз завантаженості транспортних засобів;
- визначення оптимальних маршрутів руху;
- управління світлофорними системами;
- прогнозування дорожніх заторів.

Подібні задачі часто розв'язуються за допомогою методів машинного навчання, які дозволяють будувати моделі на основі великих наборів даних. Машинне навчання використовується для прогнозування транспортних потоків, аналізу поведінки пасажирів та оцінки ефективності транспортної мережі.

Водночас класичні алгоритми машинного навчання мають певні обмеження при застосуванні до транспортних систем. Більшість таких алгоритмів працює з історичними даними та формує прогнози на їх основі. У реальних умовах транспортна ситуація може змінюватися дуже швидко, що вимагає від систем управління здатності адаптуватися до нових умов у режимі реального часу.

З цієї причини у сучасних дослідженнях все більшої популярності набувають методи підкріплювального навчання. У межах цього підходу система навчається приймати рішення шляхом взаємодії із середовищем. Алгоритм поступово формує оптимальну стратегію поведінки, отримуючи винагороду за правильні рішення та штраф за неефективні дії, добре пасуючи для задач управління складними системами, до яких належать і транспортні мережі [8].

У транспортних дослідженнях підкріплювальне навчання використовується для оптимізації різних процесів, зокрема для управління світлофорами, розподілу транспортних потоків та оптимізації руху громадського транспорту. Подібні моделі дозволяють враховувати

динамічний характер транспортних систем і знаходити ефективні стратегії управління в умовах змінного середовища.

Ще одним важливим напрямом розвитку інтелектуальних транспортних систем є використання мультиагентних підходів. У таких моделях транспортна система розглядається як сукупність окремих агентів, які взаємодіють між собою. Агентами можуть виступати транспортні засоби, пасажери, транспортні вузли або елементи системи управління. Кожен агент має власну поведінку та приймає рішення залежно від умов середовища.

Поєднання мультиагентних систем із алгоритмами підкріплювального навчання дозволяє створювати більш гнучкі моделі управління транспортними системами. Подібні підходи дають можливість досліджувати складні транспортні процеси та знаходити ефективні стратегії оптимізації навантаження транспортної мережі..

1.5 Мультиагентні системи в задачах транспортного управління

Зростання складності міських транспортних систем стимулює пошук нових підходів до управління транспортними потоками. Традиційні централізовані моделі управління не завжди здатні ефективно реагувати на динамічні зміни транспортної ситуації, оскільки рішення приймаються на основі обмеженої інформації або з певною затримкою. У таких умовах усе більшої популярності набуває концепція мультиагентних систем, яка дозволяє розглядати транспортну мережу як сукупність взаємодіючих автономних елементів [9].

Мультиагентна система являє собою обчислювальну модель, у межах якої кілька автономних агентів взаємодіють між собою та з навколишнім середовищем. Кожен агент має власні правила поведінки, здатний приймати рішення та реагувати на зміни в системі. Взаємодія агентів дозволяє

формувати складну поведінку всієї системи без необхідності централізованого управління.

У транспортних задачах агенти можуть представляти різні компоненти транспортної системи. Найчастіше такими агентами виступають:

- транспортні засоби (автобуси, трамваї, тролейбуси);
- пасажирів;
- транспортні вузли або зупинки;
- елементи системи управління транспортом.

Подібне представлення транспортної системи дозволяє моделювати поведінку її окремих компонентів і досліджувати взаємодію між ними.

Однією з важливих переваг мультиагентних систем є їхня здатність працювати в умовах децентралізованого управління. У таких системах кожен агент приймає рішення на основі локальної інформації про стан середовища. Наприклад, транспортний засіб може змінювати інтервал руху залежно від завантаженості маршруту або кількості пасажирів на зупинках.

Мультиагентні моделі дозволяють досліджувати складні транспортні процеси, які важко описати за допомогою традиційних математичних моделей. Поведінка транспортної системи формується в результаті взаємодії великої кількості агентів, кожен з яких діє відповідно до власних правил. У результаті можна отримати більш реалістичне відображення процесів, що відбуваються у транспортній мережі.

У наукових дослідженнях мультиагентні системи активно використовуються для вирішення різних задач транспортного управління. Зокрема, вони застосовуються для:

- оптимізації руху громадського транспорту;
- управління світлофорними системами;
- моделювання поведінки пасажирів;
- дослідження транспортних заторів.

Завдяки можливості моделювати взаємодію великої кількості агентів мультиагентні системи дозволяють досліджувати різні сценарії розвитку транспортної ситуації. Наприклад, можна аналізувати, як зміна кількості транспортних засобів на маршруті впливає на загальну ефективність транспортної мережі.

Ще однією важливою особливістю мультиагентних моделей є їхня здатність до масштабування. Навіть у великих транспортних системах, що містять тисячі транспортних засобів і пасажирів, мультиагентні моделі дозволяють ефективно відтворювати поведінку системи. Це робить їх особливо корисними для дослідження складних міських транспортних мереж.

У сучасних дослідженнях мультиагентні системи дедалі частіше поєднуються з методами машинного навчання. Подібний підхід дозволяє агентам адаптувати свою поведінку до змін середовища та поступово покращувати ефективність прийнятих рішень. Особливо перспективним напрямом є використання алгоритмів підкріплювального навчання, які дозволяють агентам навчатися оптимальної поведінки шляхом взаємодії з середовищем [10].

1.6 Підкріплювальне навчання в задачах транспортного управління

Підкріплювальне навчання (Reinforcement Learning, RL) є одним із напрямів машинного навчання, що орієнтований на розв'язання задач прийняття рішень у складних динамічних середовищах. На відміну від класичних методів машинного навчання, де модель навчається на основі заздалегідь підготовлених даних, підкріплювальне навчання передбачає активну взаємодію алгоритму із середовищем. У процесі такої взаємодії система поступово формує стратегію поведінки, яка дозволяє максимізувати отриману винагороду [8].

У межах підкріплювального навчання система складається з кількох основних компонентів. Ключовими елементами є агент, середовище, стан системи та функція винагороди. Агент виконує певні дії, взаємодіючи із середовищем, після чого отримує зворотний зв'язок у вигляді винагороди або штрафу. На основі цієї інформації агент поступово коригує свою поведінку.

Процес навчання в таких системах зазвичай відбувається у вигляді послідовності кроків. На кожному кроці агент:

- отримує інформацію про поточний стан середовища;
- обирає одну з можливих дій;
- виконує цю дію;
- отримує винагороду та новий стан системи.

Повторюючи подібний процес багато разів, агент поступово формує політику поведінки, яка дозволяє приймати ефективні рішення у різних ситуаціях.

Підкріплювальне навчання широко застосовується для оптимізації складних систем, де традиційні методи управління не забезпечують достатньої ефективності. До таких систем належать робототехнічні комплекси, енергетичні мережі, логістичні системи та транспортні мережі. Основною перевагою цього підходу є здатність алгоритму адаптуватися до змін середовища та знаходити ефективні стратегії управління навіть у випадках, коли точна модель системи невідома.

У транспортних системах підкріплювальне навчання може використовуватися для вирішення різноманітних задач. Наприклад, воно застосовується для оптимізації роботи світлофорів, управління потоками транспортних засобів або оптимізації роботи громадського транспорту. У подібних задачах агентом може виступати система управління транспортом, а середовищем – транспортна мережа міста.

У більш складних моделях підкріплювальне навчання поєднується з мультиагентними підходами. У такому випадку в системі одночасно

функціонує кілька агентів, які взаємодіють між собою. Наприклад, окремими агентами можуть бути транспортні засоби, що працюють на одному маршруті, або різні елементи транспортної інфраструктури.

Подібні моделі дозволяють досліджувати поведінку транспортної системи у більш реалістичних умовах. Кожен агент приймає рішення на основі власної інформації про стан системи, а загальна поведінка транспортної мережі формується в результаті взаємодії між агентами. У поєднанні з мультиагентними системами підкріплювальне навчання створює ефективний інструмент для оптимізації транспортних процесів у складних міських умовах [10].

2 ТЕХНОЛОГІЧНИЙ ТА ПРИКЛАДНИЙ ОПИС РОЗРОБКИ

2.1 Архітектура мультиагентних систем

Мультиагентні системи є одним із підходів до моделювання та управління складними динамічними середовищами. У таких системах загальна поведінка формується не централізованим керуванням, а взаємодією окремих автономних компонентів – агентів. Кожен агент здатний аналізувати стан середовища, приймати рішення та виконувати певні дії. Подібний підхід добре підходить для моделювання транспортних систем, де існує велика кількість незалежних елементів, які взаємодіють між собою [9].

У загальному вигляді мультиагентна система складається з трьох основних компонентів: агентів, середовища та механізму взаємодії між ними. Агенти можуть представляти різні об'єкти транспортної системи, тоді як середовище відображає загальний стан транспортної мережі. Взаємодія між агентами може здійснюватися як безпосередньо, так і через зміни стану середовища.

У задачах транспортного управління агентами можуть виступати різні елементи транспортної системи. Найчастіше використовуються наступні типи агентів:

- транспортні засоби (автобуси, трамваї, тролейбуси);
- пасажери або групи пасажирів;
- транспортні вузли або зупинки;
- системи управління рухом.

Кожен із таких агентів може мати власні правила поведінки та цілі. Наприклад, транспортний засіб може прагнути мінімізувати час простою на зупинках або уникати перевантажених маршрутів.

У мультиагентній системі важливу роль відіграє середовище, у якому функціонують агенти. У контексті транспортної системи середовище

включає дорожню мережу, транспортні маршрути, пасажиропотоки та інші фактори, що впливають на рух транспорту. Стан середовища постійно змінюється під впливом дій агентів і зовнішніх факторів.

Архітектура мультиагентної системи зазвичай передбачає наявність кількох рівнів взаємодії. На локальному рівні агенти приймають рішення на основі інформації про власний стан та найближче оточення. На глобальному рівні формується загальна поведінка системи, що виникає в результаті взаємодії великої кількості агентів.

Як показано на рисунку 2.1, агенти взаємодіють із середовищем та отримують інформацію про його поточний стан. На основі цієї інформації вони приймають рішення та виконують певні дії, що впливають на стан системи. Подібний механізм дозволяє моделювати складні процеси, які виникають у транспортних мережах.

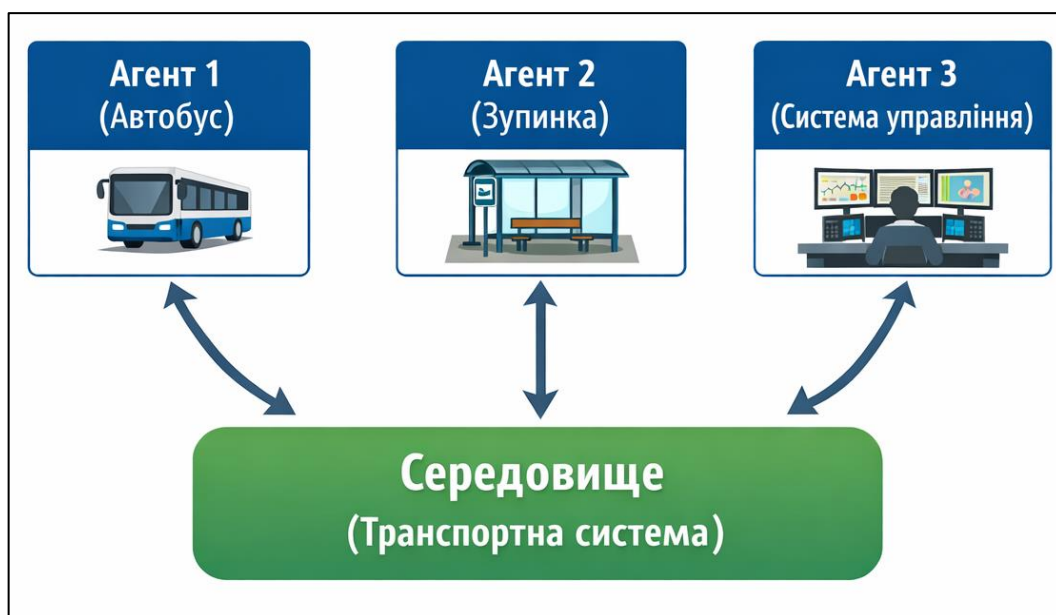


Рисунок 2.1 – Загальна структура мультиагентної системи

Залежно від способу організації мультиагентні системи можуть мати різну архітектуру. Найбільш поширеними є нижче зазначені типи архітектур:

- централізована, у якій рішення координуються центральним контролером;
- децентралізована, де кожен агент приймає рішення самостійно;
- гібридна, що поєднує елементи централізованого та децентралізованого управління.

У задачах оптимізації транспортних систем часто застосовується саме децентралізований підхід. Це пояснюється тим, що транспортна мережа складається з великої кількості незалежних об'єктів, і централізоване управління може бути недостатньо ефективним або вимагати значних обчислювальних ресурсів.

Важливою особливістю мультиагентних систем є здатність до адаптації. Агенти можуть змінювати свою поведінку залежно від стану середовища або взаємодії з іншими агентами. Це робить мультиагентні системи придатними для моделювання складних динамічних процесів, таких як рух транспорту у великих містах.

У сучасних дослідженнях мультиагентні системи часто поєднуються з алгоритмами машинного навчання. Особливо ефективним виявляється використання підкріплювального навчання, яке дозволяє агентам самостійно знаходити оптимальні стратегії поведінки в процесі взаємодії із середовищем, отримавши назву multi-agent reinforcement learning і активно застосовується для оптимізації транспортних систем [10].

2.2 Алгоритми машинного навчання для задач оптимізації

2.2.1 Q-learning

Алгоритм Q-learning є одним із базових методів підкріплювального навчання, що використовується для знаходження оптимальної стратегії прийняття рішень у середовищах з дискретними станами та діями. Його основна ідея полягає у поступовому навчанні функції цінності дій, яка

дозволяє оцінити, наскільки вигідно виконати певну дію у конкретному стані системи. У транспортних системах це може застосовуватися для оптимізації рішень щодо управління рухом транспортних засобів, вибору маршрутів або регулювання інтервалів руху.

У Q-learning агент поступово формує таблицю значень Q , яка відображає очікувану винагороду для кожної пари стан–дія. У процесі навчання агент взаємодіє із середовищем: спостерігає стан системи, обирає дію, отримує винагороду та переходить у новий стан. Отримана інформація використовується для оновлення значення Q -функції. Основна мета алгоритму полягає у знаходженні такої політики поведінки, яка максимізує сумарну винагороду у довгостроковій перспективі [8].

Формально Q -функція визначає очікувану винагороду для пари стан–дія і записується у вигляді формули 2.1:

$$Q(s, a) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right], \quad (2.1)$$

де s – поточний стан середовища;

a – дія агента;

r_{t+1} – отримана винагорода;

γ – коефіцієнт дисконтування.

У процесі навчання значення функції Q оновлюється за допомогою ітераційного правила, зазначеної у формулі 2.2:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right], \quad (2.2)$$

де α – коефіцієнт навчання;

s' – новий стан середовища;

a' – можливі дії у новому стані.

Алгоритм навчання можна представити у вигляді послідовності кроків:

- ініціалізація таблиці $Q(s, a)$;
- спостереження поточного стану s ;
- вибір дії відповідно до стратегії дослідження;
- виконання дії та отримання винагороди r ;
- оновлення значення $Q(s, a)$ за формулою 2.2.

Для задач управління транспортними системами Q-learning дозволяє агентам поступово навчатися оптимальній поведінці. Наприклад, транспортний агент може визначати оптимальні інтервали руху або змінювати маршрути залежно від завантаженості транспортної мережі.

Як показано на рисунку 2.2, процес навчання агента відбувається циклічно. Агент постійно взаємодіє із середовищем, отримуючи інформацію про його поточний стан та результати виконаних дій.

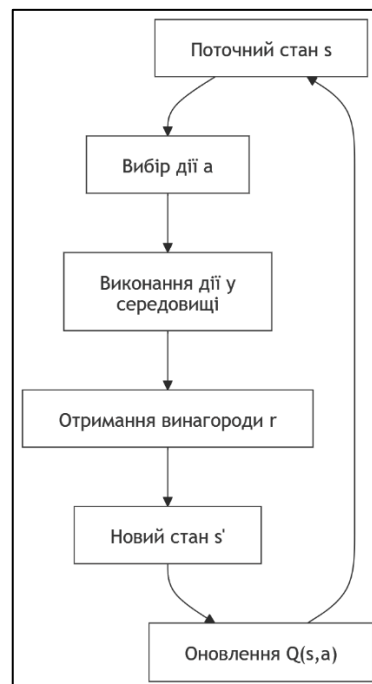


Рисунок 2.2 – Схема роботи алгоритму Q-learning

На основі отриманої винагороди оновлюються значення функції $Q(s, a)$, що дозволяє поступово покращувати стратегію поведінки агента. У задачах управління транспортними системами подібний підхід дає можливість агентам адаптуватися до змін у транспортному середовищі та знаходити ефективні рішення щодо оптимізації транспортних потоків.

2.2.2 Deep Q-Network (DQN)

Алгоритм Deep Q-Network (DQN) є розвитком класичного методу Q-learning, у якому замість таблиці значень використовується нейронна мережа, дозволяючи працювати з набагато складнішими середовищами, де кількість можливих станів є дуже великою. У транспортних системах це особливо важливо, оскільки стан мережі може залежати від великої кількості факторів: завантаженості маршрутів, кількості пасажирів, інтервалів руху транспорту або дорожньої ситуації.

У класичному Q-learning кожна пара стан-дія має відповідне значення у таблиці Q. Однак у великих системах така таблиця може містити мільйони можливих комбінацій, що робить її практично непридатною для використання. У DQN цю проблему вирішують шляхом використання нейронної мережі, яка апроксимує функцію $Q(s, a)$ і дозволяє оцінювати цінність дій навіть для станів, які раніше не зустрічалися під час навчання [8].

Архітектурно модель працює таким чином: на вхід нейронної мережі подається опис поточного стану середовища, після чого мережа обчислює оцінку функції Q для всіх можливих дій. Агент обирає одну з цих дій і виконує її у середовищі, отримуючи винагороду та новий стан системи. На основі цієї інформації параметри мережі поступово коригуються.

У загальному вигляді цільове значення Q-функції визначається за формулою 2.3:

$$y = r + \gamma \max_{a'} Q(s', a'; \theta^-), \quad (2.3)$$

де r – отримана винагорода;

s' – новий стан середовища;

γ – коефіцієнт дисконтування;

θ^- – параметри цільової мережі.

Під час навчання нейронна мережа намагається мінімізувати різницю між поточним значенням функції Q та її цільовим значенням. Для цього використовується функція втрат, висвітлена у формулі 2.4:

$$L(\theta) = \mathbb{E}[(y - Q(s, a; \theta))^2], \quad (2.4)$$

де θ – параметри нейронної мережі.

На практиці реалізація алгоритму DQN включає кілька важливих механізмів стабілізації навчання. Найбільш поширеними є:

- Experience Replay: збереження попереднього досвіду агента у спеціальному буфері пам'яті;
- Target Network: використання окремої мережі для обчислення цільових значень Q ;
- ϵ -greedy стратегія: баланс між дослідженням нових дій та використанням уже знайдених рішень.

Вони дозволяють уникнути нестабільності навчання та покращують якість отриманої політики поведінки.

Як показано на рисунку 2.3, алгоритм Deep Q-Network використовує нейронну мережу для оцінювання значень функції $Q(s, a)$ для всіх можливих дій у поточному стані середовища. На основі цих оцінок агент обирає дію, яка виконується у середовищі. Після виконання дії система отримує новий стан середовища та відповідну винагороду, що використовується для подальшого навчання моделі.

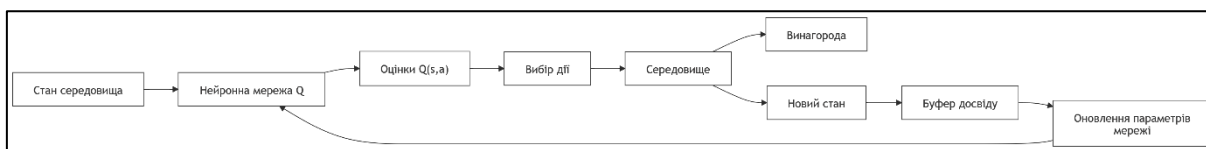


Рисунок 2.3 – Архітектура алгоритму Deep Q-Network

Важливою особливістю DQN є використання буфера досвіду та окремої цільової мережі, що дозволяє стабілізувати процес навчання. Буфер досвіду зберігає попередні взаємодії агента із середовищем, які випадковим чином використовуються під час навчання нейронної мережі. Це зменшує кореляцію між навчальними прикладами та підвищує ефективність алгоритму при роботі зі складними динамічними системами, такими як транспортні мережі.

2.2.3 Policy Gradient

На відміну від алгоритмів типу Q-learning, де агент навчається оцінювати корисність окремих дій, підхід Policy Gradient працює інакше. У цих методах агент безпосередньо навчається формувати політику поведінки – тобто правило, яке визначає, яку дію потрібно обрати в певному стані системи, що дозволяє уникнути побудови таблиць значень Q і працювати з більш складними або безперервними просторами станів і дій.

Політика у таких алгоритмах зазвичай задається параметризованою функцією. Вона може бути представлена у вигляді нейронної мережі, яка отримує на вході стан середовища та повертає розподіл ймовірностей можливих дій. Це означає, що агент не просто обирає дію з найбільшим значенням, а приймає рішення відповідно до певної ймовірнісної моделі.

Формально політика описується функцією у формулі 2.5:

$$\pi_{\theta}(a | s), \quad (2.5)$$

де s – стан середовища;

a – можлива дія;

θ – параметри політики.

Метою навчання є підбір таких параметрів θ , які максимізують сумарну очікувану винагороду агента під час взаємодії із середовищем. Цільову функцію можна записати у формулу 2.6:

$$J(\theta) = \mathbb{E}_{\pi_{\theta}} \left[\sum_{t=0}^T \gamma^t r_t \right], \quad (2.6)$$

де γ – коефіцієнтом дисконтування майбутніх винагород.

Основна ідея методу полягає у використанні градієнтного підйому для покращення політики. Параметри політики змінюються у напрямку, який збільшує очікувану винагороду (формула 2.7):

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a | s) Q^{\pi}(s, a)]. \quad (2.7)$$

Практично це означає, що дії, які призводять до більшої винагороди, отримують більшу ймовірність вибору в майбутньому.

Якщо коротко узагальнити роботу алгоритмів Policy Gradient, то процес навчання можна описати так:

- агент спостерігає поточний стан середовища;
- на основі політики обирається дія;
- дія виконується у середовищі;
- агент отримує винагороду;
- параметри політики оновлюються відповідно до отриманого результату.

Як показано на рисунку 2.4, у центрі алгоритму знаходиться політика агента, яка визначає вибір дій у кожному стані середовища. Отримана після виконання дії винагорода використовується для коригування параметрів

політики, що дозволяє механізму агента поступово покращувати свою поведінку.



Рисунок 2.4 – Принцип роботи алгоритмів Policy Gradient

Методи Policy Gradient широко застосовуються у задачах управління складними системами, де класичні алгоритми навчання можуть бути недостатньо ефективними. У транспортних системах подібні підходи можуть використовуватися для адаптивного управління транспортними потоками або оптимізації роботи громадського транспорту.

2.2.4 Actor-Critic

Алгоритми Actor-Critic поєднують ідеї двох підходів підкріплювального навчання: методів оцінювання значення дій та методів оптимізації політики. У таких моделях процес навчання розділяється між двома компонентами – актором та критиком. Актор відповідає за вибір дій відповідно до поточної політики, тоді як критик оцінює якість цих дій і допомагає коригувати політику агента.

Основна ідея Actor-Critic полягає в тому, що замість безпосереднього використання функції $Q(s, a)$ для прийняття рішень використовується оцінка функції цінності стану. Критик аналізує стан середовища та визначає, наскільки вигідним є перебування системи в цьому стані. Актор, у свою чергу, використовує цю інформацію для коригування політики та вибору більш ефективних дій.

Функція цінності стану визначається у формулі 2.8:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right], \quad (2.8)$$

де s – стан середовища;

γ – коефіцієнт дисконтування;

r_t – винагорода на кроці t .

Під час навчання використовується так званий помилковий сигнал тимчасової різниці (Temporal Difference error), який визначає різницю між очікуваною та фактично отриманою винагородою (формула 2.9):

$$\delta = r + \gamma V(s') - V(s), \quad (2.9)$$

де s' – новий стан середовища.

Це значення використовується критиком для оновлення оцінки функції цінності, а актором – для коригування параметрів політики.

У загальному вигляді взаємодію між компонентами алгоритму можна описати так:

- актор генерує дію відповідно до поточної політики;
- середовище повертає новий стан та винагороду;
- критик оцінює якість отриманого результату;
- політика актора коригується на основі оцінки критика.

На рисунку 2.5 показано взаємодію між основними компонентами алгоритму.

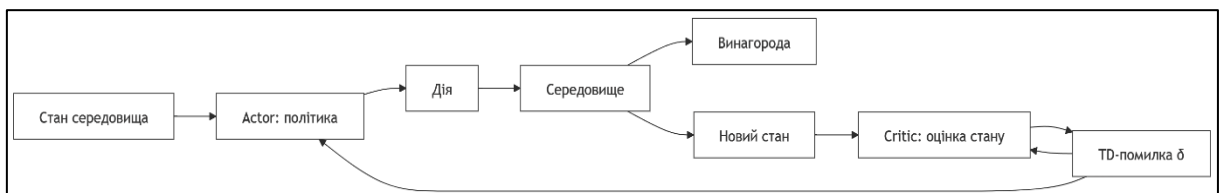


Рисунок 2.5 – Архітектура алгоритму Actor-Critic

Актор приймає рішення щодо вибору дії, тоді як критик аналізує результати цієї дії та оцінює її ефективність. Отримана оцінка використовується для коригування політики агента. Подібна структура дозволяє поєднати переваги методів оцінювання функцій цінності та методів оптимізації політики. У результаті алгоритми Actor-Critic демонструють високу ефективність у задачах управління складними системами, включаючи транспортні мережі, де необхідно приймати рішення в умовах невизначеності та змінного середовища.

2.2.5 Multi-Agent Reinforcement Learning (MARL)

У багатьох реальних системах одночасно функціонує кілька агентів, які впливають один на одного та на спільне середовище. У таких випадках використання класичних алгоритмів підкріплювального навчання для одного агента є недостатнім. Для моделювання подібних ситуацій застосовується підхід Multi-Agent Reinforcement Learning (MARL), у якому навчання відбувається одночасно для кількох взаємодіючих агентів.

У мультиагентному середовищі кожен агент спостерігає стан системи, виконує певні дії та отримує винагороду. Проте на відміну від одноагентних моделей, результат дії одного агента може впливати на стан середовища та можливості інших агентів. Через це процес навчання стає складнішим, оскільки середовище для кожного агента змінюється внаслідок поведінки інших агентів.

Формально у мультиагентній системі кожен агент має власну політику, зазначену у формулі 2.10:

$$\pi_i(a_i | s), \quad (2.10)$$

де i – індекс агента;

s – стан середовища;

a_i – дія агента i .

Загальна винагорода системи може визначатися як сума винагород усіх агентів, висвітлена у формулі 2.11:

$$R = \sum_{i=1}^N r_i, \quad (2.11)$$

де N – кількість агентів у системі.

У мультиагентному підкріплювальному навчанні застосовуються різні підходи до організації навчання. Найбільш поширеними є:

- незалежне навчання агентів, коли кожен агент навчається окремо;
- кооперативне навчання, коли агенти працюють для досягнення спільної мети;
- змагальне навчання, коли агенти мають протилежні цілі.

У задачах оптимізації транспортних систем зазвичай застосовується кооперативний підхід. У такому випадку агенти взаємодіють між собою для покращення загальної ефективності транспортної мережі. Наприклад, кілька транспортних засобів можуть координувати свої дії для рівномірного розподілу пасажиропотоку на маршрутах.

На рисунку 2.6 показано принцип взаємодії кількох агентів у спільному середовищі. Кожен агент виконує власні дії, які змінюють стан системи.

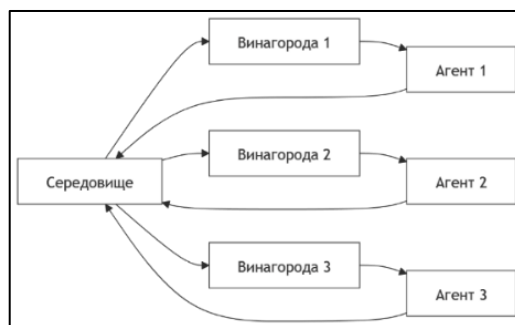


Рисунок 2.6 – Взаємодія агентів у мультиагентному підкріплювальному навчанні

У відповідь середовище формує новий стан та винагороду для кожного агента. Подібна структура дозволяє моделювати складні системи, у яких рішення приймаються одночасно кількома незалежними учасниками. Навчання таких агентів дозволяє знаходити ефективні стратегії координації, що сприяє більш рівномірному розподілу навантаження в транспортній системі [10].

2.3 Методи моделювання транспортних систем

2.3.1 Агентне моделювання

Агентне моделювання (Agent-Based Modeling, ABM) є одним із підходів до дослідження складних систем, у яких поведінка всієї системи формується через взаємодію окремих автономних компонентів. У межах такого підходу система розглядається як сукупність незалежних агентів, кожен з яких має власний стан, правила поведінки та можливість взаємодії з іншими агентами або середовищем. Подібна модель дозволяє описувати складні процеси, що виникають у системах з великою кількістю учасників [9].

У транспортних системах агентами можуть виступати різні об'єкти: транспортні засоби, пасажирів, зупинки або елементи транспортної інфраструктури. Кожен агент здатний реагувати на зміну стану середовища та приймати рішення на основі доступної інформації. Наприклад, пасажир може обирати найближчий маршрут або той, який забезпечує найменший час поїздки, тоді як транспортний засіб може змінювати свою поведінку залежно від кількості пасажирів на зупинках.

З формальної точки зору агент у моделі можна представити у вигляді набору характеристик (формула 2.12):

$$Agent_i = (S_i, A_i, T_i), \quad (2.12)$$

де S_i – множина можливих станів агента;

A_i – множина можливих дій;

T_i – функція переходу між станами.

У процесі моделювання система розвивається у часі. На кожному кроці агенти спостерігають стан середовища, обирають відповідну дію та змінюють власний стан або стан системи. Формально перехід системи між станами можна описати у формулі 2.13:

$$S_{t+1} = f(S_t, A_t), \quad (2.13)$$

де S_t – стан системи у момент часу t ;

A_t – набір дій усіх агентів.

Особливістю агентного моделювання є те, що глобальна поведінка системи виникає як результат локальних рішень агентів. Іншими словами, система не потребує централізованого управління. Це дозволяє більш реалістично моделювати транспортні процеси, оскільки кожен учасник системи діє відповідно до власних цілей та доступної інформації.

У транспортних дослідженнях агентне моделювання застосовується для аналізу поведінки пасажирів, моделювання транспортних потоків та дослідження взаємодії між транспортними засобами. Подібні моделі дозволяють оцінювати ефективність різних стратегій управління транспортною мережею та прогнозувати можливі зміни у транспортній системі.

Застосування агентного підходу також створює основу для використання алгоритмів підкріплювального навчання. У такому випадку агенти можуть навчатися на основі власного досвіду та поступово покращувати свою поведінку, що відкриває можливості для оптимізації роботи транспортних систем у складних динамічних умовах.

2.3.2 Моделювання пасажиропотоку

Моделювання пасажиропотоків є важливим етапом дослідження транспортних систем, оскільки дозволяє оцінити розподіл пасажирів у транспортній мережі та прогнозувати навантаження на маршрути громадського транспорту. Пасажиропотік характеризує кількість пасажирів, які переміщуються між різними точками транспортної мережі протягом певного періоду часу. Аналіз таких потоків дозволяє виявити перевантажені ділянки мережі та визначити напрями для оптимізації транспортного обслуговування.

У більшості транспортних моделей пасажиропотік описується як функція часу та просторового розташування. Для певного маршруту інтенсивність потоку можна записати у вигляді формули 2.14:

$$q = \frac{N}{T}, \quad (2.14)$$

де q – інтенсивність пасажиропотоку;

N – кількість пасажирів, що пройшли через певну точку мережі;

T – проміжок часу спостереження.

Подібні моделі дозволяють визначити середню завантаженість транспортних засобів і оцінити пропускну здатність транспортної інфраструктури.

Для більш детального аналізу використовується матриця походження–призначення (Origin–Destination Matrix), яка відображає кількість пасажирів, що переміщуються між різними транспортними вузлами. Така матриця має вигляд, зазначений у формулі 2.15:

$$OD = \begin{bmatrix} d_{11} & d_{12} & \dots & d_{1n} \\ d_{21} & d_{22} & \dots & d_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{n1} & d_{n2} & \dots & d_{nn} \end{bmatrix}, \quad (2.15)$$

де d_{ij} показує кількість пасажирів, які переміщуються з вузла i до вузла j .

Як показано на рисунку 2.7, пасажирів формують потік між зонами відправлення та призначення, використовуючи певні транспортні маршрути. У межах транспортної мережі потоки можуть розподілятися між кількома альтернативними маршрутами залежно від часу поїздки, інтервалів руху транспорту та інших факторів.



Рисунок 2.7 – Формування пасажиропотоку між транспортними зонами

У сучасних дослідженнях для моделювання пасажиропотоків активно використовуються дані з електронних квитків, систем GPS-моніторингу та мобільних додатків. Аналіз таких даних дозволяє отримувати детальну інформацію про переміщення пасажирів у реальному часі та більш точно прогнозувати навантаження на транспортну мережу.

Моделі пасажиропотоку відіграють важливу роль у задачах оптимізації транспортних систем. Вони дозволяють оцінити ефективність транспортної мережі, визначити необхідну кількість транспортних засобів та дослідити вплив різних стратегій управління на розподіл пасажирів у системі. У поєднанні з агентним моделюванням і методами підкріплювального навчання моделі можуть використовуватися для створення адаптивних систем управління громадським транспортом.

2.3.3 Імітаційне моделювання транспортної мережі

Імітаційне моделювання є одним із найбільш поширених інструментів дослідження складних транспортних систем. Його основна ідея полягає у створенні комп'ютерної моделі, яка відтворює поведінку реальної

транспортної мережі у часі. Подібні моделі дозволяють аналізувати роботу системи без необхідності втручання у реальну інфраструктуру та досліджувати різні сценарії функціонування транспортної мережі.

У транспортних дослідженнях імітаційні моделі використовуються для аналізу руху транспортних засобів, дослідження поведінки пасажирів та оцінки ефективності різних стратегій управління транспортною системою. Основною перевагою такого підходу є можливість врахування великої кількості факторів, які впливають на роботу транспортної мережі.

Типова імітаційна модель транспортної системи включає кілька ключових компонентів:

- транспортну мережу (маршрути, зупинки, вузли);
- транспортні засоби;
- пасажиропотоки;
- правила руху та управління системою.

Взаємодія між цими компонентами формує динамічну модель транспортної системи, у якій стан системи змінюється з часом.

Формально динаміку транспортної системи можна описати рівнянням переходу у формулі 2.16:

$$S_{t+1} = F(S_t, U_t, W_t), \quad (2.16)$$

де S_t – стан транспортної системи у момент часу t ;

U_t – керуючі дії системи управління;

W_t – зовнішні фактори (пасажиропотік, дорожня ситуація).

У межах імітаційного моделювання система розвивається дискретними кроками часу. На кожному кроці відбувається оновлення стану транспортної мережі, переміщення транспортних засобів та зміна пасажиропотоків, що дозволяє відтворювати поведінку транспортної системи у довгостроковій перспективі.

На рисунку 2.8 видно, що імітаційна модель працює у вигляді циклічного процесу. Після ініціалізації системи відбувається послідовне оновлення стану транспортної мережі, виконання керуючих дій та аналіз результатів моделювання.



Рисунок 2.8 – Загальна логіка імітаційного моделювання транспортної системи

Практичне застосування імітаційного моделювання дозволяє досліджувати ефективність різних стратегій управління транспортною системою. Зокрема, можна оцінювати вплив зміни кількості транспортних засобів на маршруті, моделювати зміну пасажиропотоків або аналізувати наслідки введення нових транспортних маршрутів. У поєднанні з алгоритмами підкріплювального навчання подібні моделі можуть використовуватися як середовище для навчання агентів, що дозволяє оптимізувати роботу транспортної системи у складних умовах.

2.4 Вибір технологій для реалізації системи

2.4.1 Мови програмування та бібліотеки

Одним із ключових рішень при розробці програмної системи є вибір мови програмування. Для реалізації транспортних моделей, алгоритмів навчання та серверної частини системи обрано мову Python, яка є стандартом де-факто у сфері машинного навчання та наукових обчислень. Python забезпечує широку екосистему бібліотек, просту інтеграцію між компонентами та зручність прототипування складних алгоритмів.

Для роботи з числовими даними, векторами спостережень і матрицями пасажиропотоків використовується бібліотека NumPy, яка забезпечує ефективні операції над масивами без накладних витрат інтерпретатора. Побудова та навчання нейронних мереж реалізовано засобами PyTorch – фреймворку, що підтримує автоматичне диференціювання, GPU-прискорення та гнучку побудову архітектур нейронних мереж. Граф транспортної мережі реалізовано через бібліотеку NetworkX, яка надає зручний інтерфейс для роботи з орієнтованими графами та алгоритмами пошуку шляхів.

Серверна частина системи реалізована на фреймворку FastAPI, який забезпечує асинхронну обробку HTTP-запитів і WebSocket-з'єднань, автоматичну генерацію документації та валідацію даних через Pydantic. Для клієнтського інтерфейсу обрано бібліотеку React у поєднанні з інструментом збірки Vite та CSS-фреймворком Tailwind, що дозволяє будувати інтерфейс із мінімальними витратами на конфігурацію.

2.4.2 Фреймворки для підкріплювального навчання

Для реалізації алгоритмів підкріплювального навчання необхідно використовувати спеціалізовані бібліотеки машинного навчання. Однією з

найбільш поширених мов для реалізації таких алгоритмів є Python, оскільки вона має велику кількість бібліотек для роботи з нейронними мережами та алгоритмами навчання.

Серед найбільш поширених фреймворків для реалізації алгоритмів підкріплювального навчання можна виділити:

- TensorFlow: бібліотеку для створення та навчання нейронних мереж;
- PyTorch: фреймворк для побудови моделей глибокого навчання;
- Stable-Baselines3: бібліотеку, що містить реалізації популярних алгоритмів RL.

TensorFlow та PyTorch використовуються для побудови нейронних мереж, які можуть апроксимувати функції цінності або політики поведінки агента. Stable-Baselines3, у свою чергу, містить готові реалізації алгоритмів підкріплювального навчання, таких як DQN, PPO або A2C, що значно спрощує процес розробки моделей [11].

Використання подібних бібліотек дозволяє швидко реалізувати алгоритми навчання та експериментувати з різними підходами до оптимізації поведінки агентів.

2.4.3 Інструменти моделювання транспортних систем

Для навчання агентів підкріплювального навчання необхідно створити середовище симуляції, у якому агенти можуть взаємодіяти з транспортною системою. Серед спеціалізованих інструментів моделювання транспортних мереж найбільш поширеним є SUMO (Simulation of Urban Mobility), що дозволяє моделювати рух транспортних засобів у міській мережі [13]. Іншим підходом є використання AnyLogic, який підтримує агентне моделювання та імітаційні транспортні моделі.

Проте у даній роботі прийнято рішення реалізувати власне середовище симуляції замість використання SUMO або AnyLogic. Основними причинами є такі.

По-перше, SUMO орієнтований на мікроскопічне моделювання руху окремих автомобілів, тоді як у задачі оптимізації громадського транспорту ключовим є управління на рівні маршрутів і зупинок.

По-друге, інтеграція SUMO з Python через TraCI API вносить суттєві накладні витрати часу на кожен крок симуляції, що критично при навчанні протягом сотень епізодів.

По-третє, власне середовище дозволило безпосередньо завантажувати реальні GTFS-дані харківської мережі та OD-матрицю пасажиропотоків без проміжних конвертацій формату.

Перелічені фактори зумовили вибір власної реалізації середовища симуляції сумісного з інтерфейсом Gymnasium, що дозволило повністю контролювати логіку генерації пасажиропотоків, дискретизацію часу та інтеграцію з GTFS-даними харківської транспортної мережі.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Реалізація середовища та агентів

3.1.1 Граф транспортної мережі та пасажиропотоки

Реалізація транспортного середовища розпочинається з побудови графу маршрутної мережі, який є основою для моделювання руху транспортних засобів та взаємодії агентів. Граф реалізовано у класі `NetworkGraph` з використанням бібліотеки `NetworkX` у вигляді орієнтованого графу (`DiGraph`), де вузли відповідають транспортним зупинкам, а ребра – відрізкам маршрутів між ними.

Вхідними даними для побудови графу слугують два джерела: оброблений файл `routes.json` із описом маршрутів та зупинок, а також GTFS-файли з реальними даними харківської транспортної мережі. Мережа містить 30 зупинок (S01-S30), розташованих у різних районах міста – від центральних вузлів (Майдан Конституції, Держпром, Площа Свободи) до периферійних районів (Салтівка, ХТЗ, П'ятихатки, Рогань). Координати кожної зупинки задані у форматі WGS-84 (широта/довгота), що дозволяє використовувати їх для відображення на карті та обчислення реальних відстаней.

Транспортна мережа включає 10 маршрутів (R01–R10), серед яких чотири тролейбусні (маршрути 5т, 3т, 7т, 11т) та шість автобусних (маршрути 14, 47, 271, 89, 55, 23). Кількість зупинок на маршруті варіюється від 7 до 11, середня розрахункова швидкість – від 20 км/год для тролейбусів до 30 км/год для автобусів. Характеристики маршрутів наведено у таблиці 3.1.

Таблиця 3.1 – Характеристики транспортних маршрутів

Маршрут	Тип	Кількість зупинок	Середня швидкість, км/год
R01 (5т)	Тролейбус	11	22
R02 (14)	Автобус	9	25
R03 (47)	Автобус	9	28
R04 (3т)	Тролейбус	8	20
R05 (271)	Автобус	8	30
R06 (7т)	Тролейбус	7	21
R07 (89)	Автобус	8	26
R08 (55)	Автобус	9	29
R09 (23)	Автобус	7	27
R10 (11т)	Тролейбус	8	20

Під час завантаження кожне ребро графу отримує два атрибути: відстань між зупинками у метрах (`distance_m`) та розрахунковий час руху у секундах (`travel_time_s`), що обчислюється на основі відстані та середньої швидкості маршруту. Якщо пряме ребро між зупинками відсутнє у даних, відстань обчислюється за формулою Гаверсина на основі географічних координат. Для пошуку найкоротших шляхів між зупинками використовується алгоритм Дейкстри з вагою `travel_time_s`, результати якого кешуються для підвищення продуктивності під час симуляції.

Окрім основного графу, клас завантажує геометрію маршрутів із файлу `shapes.txt` у форматі GTFS – набір координатних точок, що описують фактичну траєкторію руху транспорту по вулицях. Це дозволяє формувати GeoJSON-представлення як зупинок, так і маршрутів для подальшого відображення у клієнтському інтерфейсі.

Моделювання пасажиропотоків реалізовано у класі `PassengerFlow`, який відповідає за генерацію пасажирів на зупинках, управління чергами очікування та посадки і висадку пасажирів. Основою для генерації є матриця походження-призначення (OD-матриця), завантажена з файлу `od_matrix.csv`. Вона задає середню годинну інтенсивність потоку між кожною парою зупинок мережі.

Генерація пасажирів на кожному кроці симуляції відбувається за розподілом Пуасона, що відповідає стохастичній природі появи пасажирів

на зупинці. Для кожної пари зупинок із OD-матриці береться годинна інтенсивність потоку, масштабується на тривалість одного кроку симуляції та множиться на часовий мультиплікатор, після чого з отриманого середнього значення семплюється випадкова кількість пасажирів.

Використання розподілу Пуасона обґрунтовано тим, що поява пасажирів на зупинці є випадковим процесом із відомою середньою інтенсивністю, що широко застосовується у транспортному моделюванні, оскільки дозволяє реалістично відтворювати варіативність пасажиропотоку без необхідності точно знати момент появи кожного окремого пасажирів.

Часовий мультиплікатор $M(t)$ моделює дві пікові години та нічний період зниженого попиту. У ранкову пікову годину (задану параметрами конфігурації) значення мультиплікатора зростає до максимального рівня за синусоїдним законом, що забезпечує плавний наростаючий і спадаючий характер навантаження. Вечірній пік моделюється аналогічно, але з коефіцієнтом 0,85 від ранкового максимуму. У нічний час застосовується знижений множник відповідно до параметра `night_multiplier` із конфігурації.

Черги пасажирів на кожній зупинці зберігаються у вигляді списку пар (`destination`, `wait_steps`), де `wait_steps` – кількість кроків симуляції, протягом яких пасажир перебуває в очікуванні. Якщо час очікування перевищує граничне значення `MAX_WAIT_STEPS = 60` кроків, пасажир автоматично залишає зупинку – це дозволяє моделювати відмову від поїздки за умов надмірного очікування.

Посадка пасажирів у транспортний засіб реалізована з урахуванням пріоритету: першочергово обслуговуються пасажирів, що очікують найдовше. Крім того, пасажир може сісти лише в той транспортний засіб, маршрут якого проходить через його зупинку призначення.

Клас також надає статистику обслуговування: загальна кількість згенерованих пасажирів, обслугованих, тих що не змогли сісти через переповненість, та тих що пішли через тривале очікування. На основі цих даних обчислюється ключова метрика – коефіцієнт

обслуговування (*service_rate*), що використовується для оцінки ефективності навченої стратегії управління.

3.1.2 Простір станів та дій агента

Для того щоб агент-автобус міг приймати рішення, йому необхідно отримувати структуровану інформацію про поточний стан середовища. У реалізованій системі спостереження агента формується класом *StateSpace* у вигляді нормалізованого вектора з 11 компонентів, де всі значення приведені до діапазону $[0, 1]$ для коректної роботи нейронної мережі.

Вектор спостереження включає такі компоненти: коефіцієнт заповненості автобуса відносно його місткості;

- нормалізований індекс поточної зупинки на маршруті;
- поточний крок симуляції відносно загальної тривалості епізоду;
- кількість пасажирів у черзі на поточній зупинці;
- середній час їх очікування;
- кількість пасажирів на наступній зупинці;
- відстань до наступної зупинки;
- інтервал до попереднього автобуса на маршруті;
- інтервал до наступного автобуса;
- ознака кінцевої зупинки;
- ознака пікового часу доби.

Набір ознак дозволяє агенту враховувати як власний стан, так і ситуацію на сусідніх зупинках та відносно розташування інших транспортних засобів на маршруті. Схему графу транспортної мережі з позначенням вузлів-зупинок та ребер-маршрутів наведено на рисунку 3.1. Слід зазначити, що координати зупинок є наближеними та використовуються для візуалізації структури мережі, а не для точного географічного відображення.

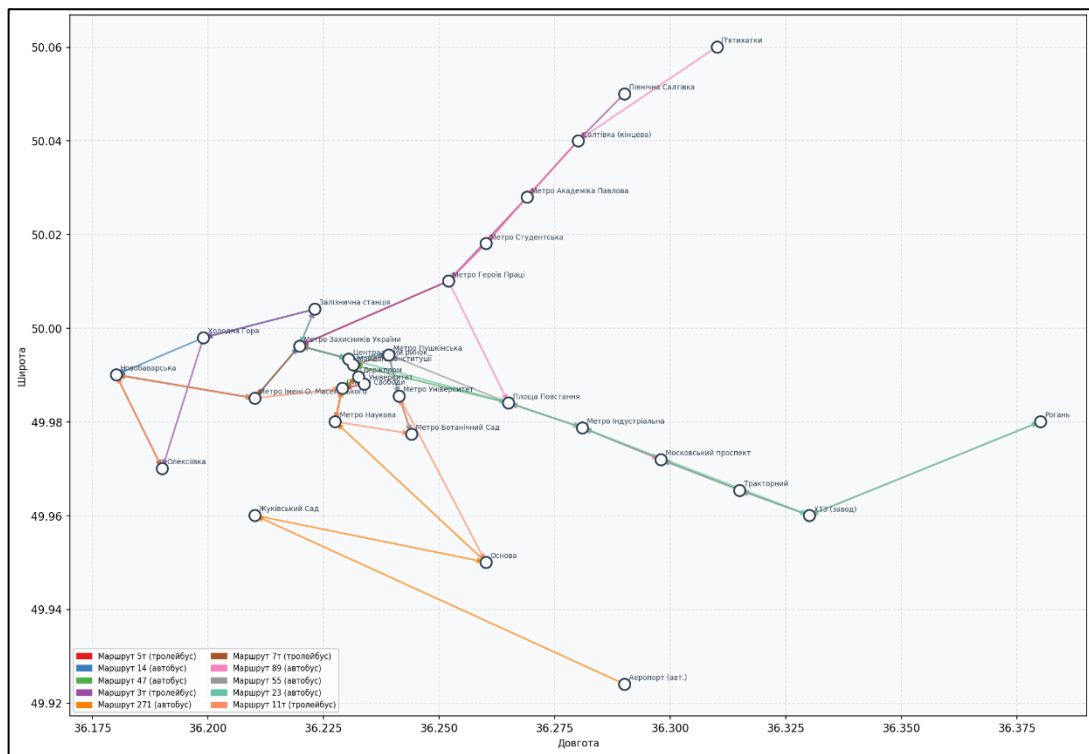


Рисунок 3.1 – Граф транспортної мережі

Простір дій агента є дискретним і містить чотири можливі дії. Перша – рухатись до наступної зупинки (MOVE_NEXT) – базова дія, яка завжди доступна. Друга – чекати на поточній зупинці (WAIT) – дозволяє прийняти додаткових пасажирів, але недоступна якщо автобус переповнений.

Третя – пропустити наступну зупинку (SKIP_NEXT) – застосовується для прискорення руху на перевантажених ділянках, але заблокована якщо заповненість нижче 30% або транспортний засіб знаходиться на кінцевій зупинці.

Четверта – затриматись для регулювання інтервалу (HOLD_AT_STOP). Вона активується лише якщо попередній автобус знаходиться на відстані не більше двох зупинок, щоб запобігти утворенню колон. Перевірка допустимості кожної дії виконується методом `is_valid()` класу `ActionSpace` перед кожним кроком симуляції.

Функція винагороди реалізована у класі `RewardFunction` і складається з кількох компонентів, що штрафують небажану поведінку та заохочують

ефективну роботу. Штраф за переповненість нараховується якщо кількість пасажирів перевищує місткість транспортного засобу, причому інтенсивність штрафу зростає пропорційно до перевищення.

Штраф за очікування залежить від кількості пасажирів у черзі на поточній зупинці та від середнього часу їх очікування – якщо він перевищує 10 кроків, штраф збільшується додатково. Штраф за пасажирів, що не змогли сісти через переповненість автобуса, нараховується окремо. Штраф за convoу застосовується коли два автобуси одного маршруту рухаються впритул – за відстані в одну зупинку штраф максимальний, за дві – частковий.

Поряд зі штрафами функція містить два бонуси. Бонус за рівномірне завантаження нараховується коли стандартне відхилення коефіцієнтів заповненості між усіма автобусами маршруту є малим: при відхиленні менше 0,1 бонус подвоюється, при менше 0,2 – повний, при менше 0,3 – половина. Бонус за точність розкладу нараховується якщо транспортний засіб прибуває на зупинку вчасно. Динаміку часового мультиплікатора $M(t)$ протягом доби, що відображає ранковий та вечірній піки пасажиропотоку, наведено на рисунку 3.2.

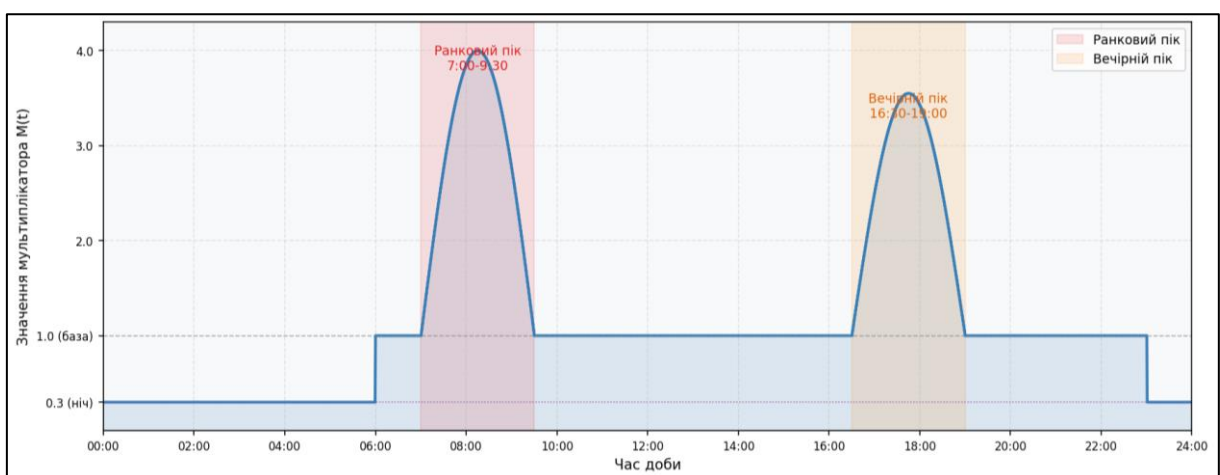


Рисунок 3.2 – Динаміка часового мультиплікатора $M(t)$ протягом доби

Для навчання в режимі MARL додатково використовується системна винагорода `compute_system_reward()`, яка обчислюється на рівні всього маршруту: враховує сумарну чергу на всіх зупинках, загальну кількість пасажирів що не сіли, та рівномірність завантаження між усіма автобусами. Крім того, реалізовано reward shaping за методом Ng: до сирої винагороди додається різниця потенціалів стану до і після дії, де потенціал визначається як від’ємна сума черг по всіх зупинках.

3.1.3 Реалізація агентів системи

Усі агенти системи реалізують спільний інтерфейс, визначений в абстрактному класі `BaseAgent`. Він задає чотири обов’язкові методи: `observe()` для отримання спостереження зі середовища, `act()` для вибору дії, `reset()` для скидання стану на початок епізоду та `learn()` для оновлення внутрішньої моделі, який за замовчуванням лише накопичує статистику і перевизначається у навчаючих агентів. Кожен агент також зберігає базову статистику: загальну та останню винагороду, кількість виконаних кроків і ознаку активності. UML-діаграму базового класу наведено на рисунку 3.3.

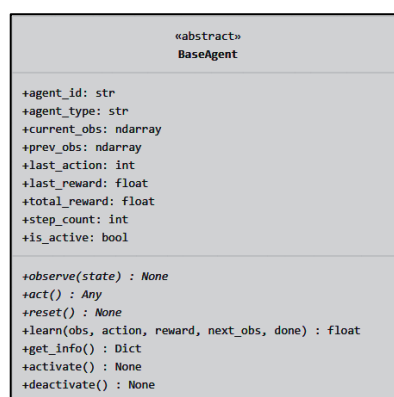


Рисунок 3.3 – UML-діаграма базового класу `BaseAgent`

Центральним агентом системи є `BusAgent`, що реалізує логіку прийняття рішень транспортного засобу. Агент підтримує epsilon-greedy

стратегію дослідження: з ймовірністю ϵ обирається випадкова допустима дія, в іншому разі дія визначається підключеною policy-функцією. Така архітектура дозволяє використовувати будь-який навчений алгоритм (Q-таблиця, DQN, Actor-Critic) без зміни коду агента – достатньо передати відповідну функцію через `set_policy()`. Агент веде локальний буфер досвіду, який після кожного кроку передається до глобального replay buffer тренера. Окрім основного режиму навчання, реалізовано метод `act_greedy()` для оцінювання навченої моделі без дослідження середовища. UML-діаграму класу наведено на рисунку 3.4.

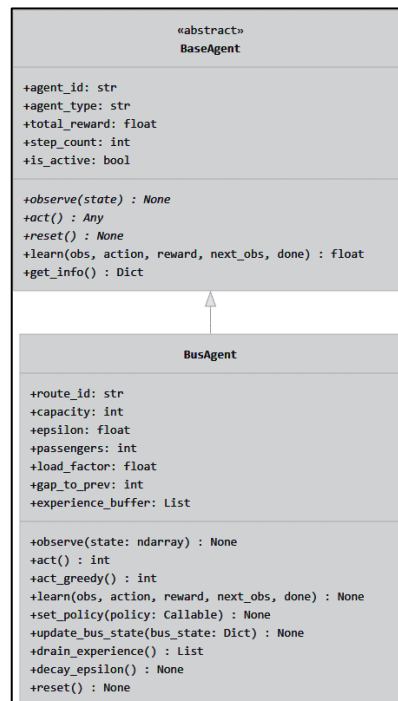


Рисунок 3.4 – UML-діаграма класу BusAgent

Агент-зупинка StopAgent є пасивним учасником системи - він не навчається і не приймає активних рішень, але виконує ключову роль у керуванні чергами пасажирів. На кожному кроці симуляції метод `tick()` збільшує час очікування кожного пасажирів в черзі та видаляє тих, хто перевищив граничний час очікування.

При прибутті автобуса метод `board_bus()` визначає яких пасажирів можна посадити: враховуються лише ті, чия зупинка призначення знаходиться попереду на маршруті, причому пріоритет мають пасажирів з найбільшим часом очікування. Агент також надає нормалізований вектор стану з чотирьох компонентів - розмір черги, середній та максимальний час очікування і коефіцієнт обслуговування – який використовується іншими агентами як частина спостереження. UML-діаграму класу наведено на рисунку 3.5.

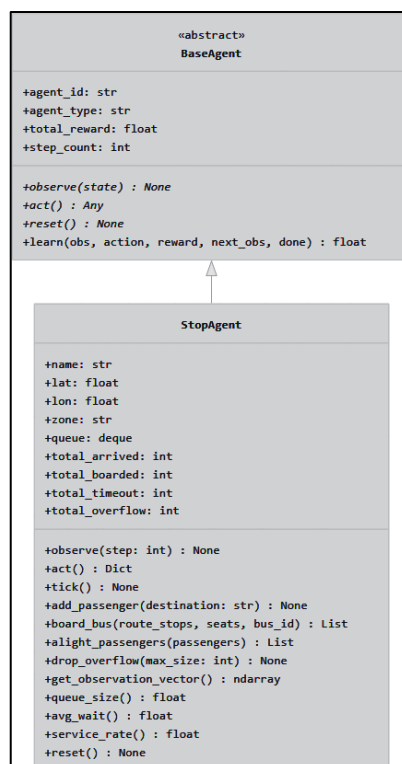


Рисунок 3.5 – UML-діаграма класу StopAgent

Агент-пасажир `PassengerAgent` моделює поведінку окремого пасажирів через чотири стани, визначені переліком `PassengerStatus`: очікування на зупинці, поїздка в автобусі, прибуття до призначення та відмова від поїздки. Вибір автобуса реалізовано у методі `_choose_bus()`: з доступних транспортних засобів відбираються ті, що прямують до зупинки

призначення і мають вільні місця, після чого для кожного обчислюється оцінка на основі завантаженості та належності до пріоритетних маршрутів.

З двох найкращих кандидатів один обирається випадково з ваговою вибіркою, що забезпечує реалістичність поведінки. Пасажир також адаптує свою стратегію: при негативному досвіді він поступово збільшує вагу часу очікування і зменшує вагу завантаженості при виборі автобуса. UML-діаграму класу наведено на рисунку 3.6.

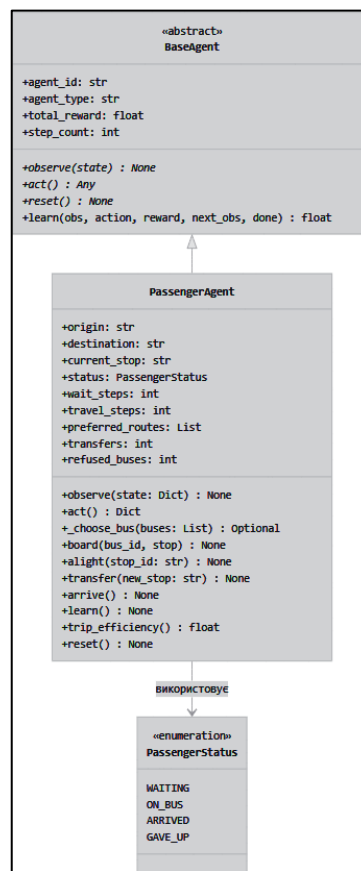


Рисунок 3.6 – UML-діаграма класу PassengerAgent

Агент-координатор ControlAgent відповідає за запобігання convoy effect – ситуації коли кілька автобусів одного маршруту збиваються в групу, внаслідок чого перший переповнений, а наступні порожні. На кожному кроці агент отримує позиції та завантаженість всіх автобусів маршруту, виявляє пари що рухаються впритул методом detect_convoys(), і формує

рекомендації щодо дій: якщо задній автобус майже порожній – йому рекомендується затриматись (HOLD_AT_STOP), якщо передній переповнений – задній пропускає наступну зупинку (SKIP_NEXT).

Рекомендації передаються до BusAgent через метод `override_action()`, який заміщує запропоновану агентом дію на дію координатора якщо така існує. Для оцінки якості розподілу транспортних засобів координатор обчислює метрику рівномірності `evenness_score` від 0 до 1, де 1 відповідає ідеально рівномірному розподілу між зупинками. UML-діаграму класу наведено на рисунку 3.7.

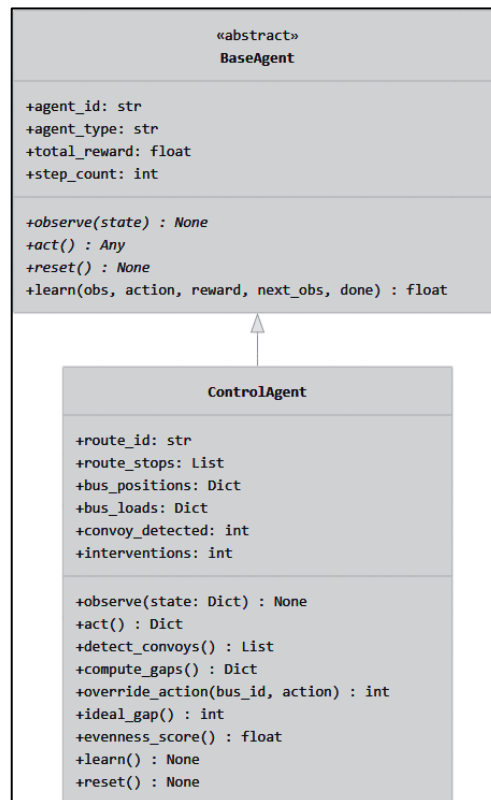


Рисунок 3.7 – UML-діаграма класу ControlAgent

Взаємодія між агентами організована таким чином, що на кожному кроці симуляції ControlAgent першим аналізує позиції транспортних засобів і формує рекомендації, після чого BusAgent отримує спостереження, обирає дію і звіряє її з рекомендацією координатора через `override_action()`.

Паралельно StopAgent оновлює черги і передає інформацію про завантаженість зупинок до вектора спостереження автобусного агента, а PassengerAgent ухвалює рішення про посадку на основі доступних транспортних засобів на поточній зупинці.

3.2 Реалізація алгоритмів навчання

3.2.1 Реалізація Q-learning та DQN

Алгоритм Q-learning реалізовано у класі QLearning як базовий метод для перевірки середовища та порівняння з більш складними підходами. Оскільки вектор спостереження агента є неперервним, перед збереженням у Q-таблицю він дискретизується: кожен з 11 компонентів ділиться на рівні інтервали і перетворюється на цілочисельний індекс, як показано у лістингу 3.1.

Лістинг 3.1 – Дискретизація вектора спостереження

```
def discretize_obs(obs: np.ndarray, bins: int = 10) -> Tuple:
    return tuple((obs * bins).astype(int).clip(0, bins -
1).tolist())
```

Q-таблиця реалізована як словник, де ключем є дискретизований стан, а значенням - масив Q-значень для кожної з чотирьох дій. Нові стани ініціалізуються нулями при першому зверненні. Оновлення Q-значень виконується за правилом Беллмана, як наведено у лістингу 3.2.

Лістинг 3.2 – Оновлення Q-таблиці

```
target = reward + (0.0 if done else self.gamma *
np.max(q_next))
td_error = target - q_current[action]
q_current[action] += self.alpha * td_error
```

Після навчання клас повертає функцію `get_policy()`, яку можна безпосередньо підключити до `BusAgent` через `set_policy()` без зміни коду агента.

Алгоритм DQN реалізовано у класі `DQNAgent` з використанням `PyTorch`. Нейронна мережа `QNetwork` являє собою повнозв'язну архітектуру зі змінною кількістю прихованих шарів, де після кожного лінійного шару застосовується активація `ReLU` та нормалізація шару `LayerNorm`. За замовчуванням використовується конфігурація з двох прихованих шарів по 128 нейронів. Ваги ініціалізуються методом `Xavier` для стабільного старту навчання, як показано у лістингу 3.3.

Лістинг 3.3 – Архітектура `QNetwork`

```
for layer in self.net:
    if isinstance(layer, nn.Linear):
        nn.init.xavier_uniform_(layer.weight)
        nn.init.zeros_(layer.bias)
```

У `DQNAgent` реалізовано дві окремі мережі: основна `q_net` для вибору дій та цільова `target_net` для обчислення цільових `Q`-значень. Ваги цільової мережі оновлюються копіюванням з основної кожні `target_update_freq` кроків, що стабілізує навчання. Підтримується режим `Double DQN`: дія для наступного стану обирається основною мережею, а її оцінка виконується цільовою, що зменшує систематичне завищення `Q`-значень. Крок навчання наведено у лістингу 3.4.

Лістинг 3.4 – Крок навчання DQN з `Double DQN`

```
if self.use_double: next_actions =
self.q_net(next_obs_t).argmax(1)
    q_next = self.target_net(next_obs_t).gather(1,
next_actions.unsqueeze(1)).squeeze(1)
else: q_next = self.target_net(next_obs_t).max(1)[0]
q_target = rewards_t + self.gamma * q_next * (1.0 - dones_t)
loss = F.huber_loss(q_current, q_target)
```

Як функція втрат використовується Huber loss, яка менш чутлива до викидів порівняно з MSE. Градієнти обрізаються до норми 10.0 для запобігання вибуху градієнтів. Окрім стандартного буфера, реалізовано PrioritizedReplayBuffer на основі структури даних SumTree, що забезпечує семпсування за $O(\log n)$. У PER досвід з більшою TD-помилкою отримує вищий пріоритет і семплується частіше, а importance sampling weights коригують зміщення розподілу, як показано у лістингу 3.5.

Лістинг 3.5 – Семпсування з PrioritizedReplayBuffer

```
probs = np.array(priorities) / (total + 1e-8)
weights = (len(self.tree.data) * probs) ** (-self.beta)
weights = weights / (weights.max() + 1e-8)
```

Після кожного епізоду планувальник швидкості навчання StepLR знижує її на 10% кожні 500 епізодів, що сприяє стабілізації навчання на пізніх стадіях.

3.2.2 Реалізація Policy Gradient та Actor-Critic

Алгоритм REINFORCE реалізовано у класі PolicyGradient. Мережа PolicyNetwork приймає вектор спостереження і повертає розподіл ймовірностей над чотирма діями через softmax. При виборі дії невалідні дії маскуються нулями, після чого розподіл перенормовується, як показано у лістингу 3.6.

Лістинг 3.6 – Маскування невалідних дій у PolicyGradient

```
mask = torch.zeros(self.n_actions, device=self.device)
for a in valid_actions:
    mask[a] = 1.0
probs = probs * mask
probs = probs / (probs.sum() + 1e-8)
```

На відміну від DQN, алгоритми Policy Gradient оптимізують безпосередньо параметри policy-функції, а не апроксимують Q-значення. У реалізованій системі представлено два таких алгоритми: REINFORCE та Advantage Actor-Critic (A2C).

Оновлення policy відбувається лише після завершення повного епізоду за методом Монте-Карло. Для кожного кроку обчислюється дисконтована кумулятивна винагорода G_t , після чого повернення нормалізуються і центруються відносно baseline - ковзного середнього попередніх епізодів. Функція втрат включає два компоненти, як наведено у лістингу 3.7.

Лістинг 3.7 – Обчислення функції втрат REINFORCE

```
pg_loss = -(log_probs_t * advantages.detach()).mean()
entropy_loss = -entropy_coef * entropies_t.mean()
loss = pg_loss + entropy_loss
```

Перший компонент - стандартний policy gradient loss, другий - entropy regularization з коефіцієнтом 0.01, що штрафує надто детерміновану policy і стимулює дослідження середовища. Baseline оновлюється після кожного епізоду за формулою ковзного середнього з коефіцієнтом 0.95.

Клас ActorCritic реалізує дві окремі нейронні мережі з однаковою архітектурою прихованих шарів. ActorNetwork повертає розподіл ймовірностей над діями, CriticNetwork повертає скалярну оцінку стану $V(s)$. Обидві мережі використовують ортогональну ініціалізацію ваг, яка забезпечує кращу збіжність порівняно з Xavier у policy-based методах.

На відміну від REINFORCE, A2C оновлюється не раз на епізод, а кожні n кроків. Після накопичення n -step буфера обчислюються n -step повернення з bootstrap-оцінкою останнього стану через critic, як показано у лістингу 3.8.

Лістинг 3.8 – Обчислення n-step повернень з bootstrap

```
with torch.no_grad():
    next_obs_t =
torch.FloatTensor(next_obs).unsqueeze(0).to(self.device)
    bootstrap_val = self.critic(next_obs_t).item()
G = bootstrap_val
for r, done in zip(reversed(self._reward_buf),
reversed(self._done_buf)):
    G = r + self.gamma * G * (1.0 - float(done))
    returns.insert(0, G)
```

Advantage обчислюється як різниця між поверненнями і поточними оцінками critic, після чого нормалізується. Actor і Critic оновлюються окремими оптимізаторами Adam з різними швидкостями навчання, що дозволяє незалежно налаштовувати кожну мережу. Critic мінімізує MSE між оцінкою $V(s)$ і цільовим поверненням, actor - policy gradient loss з урахуванням entropy regularization, як наведено у лістингу 3.9.

Лістинг 3.9 – Оновлення Actor і Critic

```
critic_loss = self.value_coef * F.mse_loss(values_t, returns_t)
actor_loss = -(log_probs_t * advantages).mean()
actor_loss = actor_loss - self.entropy_coef * entropy
```

Градiente обох мереж обрізаються до норми 5.0. Після оновлення буфери кроків очищаються, і цикл збору досвіду починається заново. Обидва алгоритми реалізують метод `get_policy()`, що повертає детерміновану функцію вибору дій для підключення до BusAgent під час оцінювання навченої моделі.

3.2.3 MARL-тренер та процес навчання моделей

Клас MARLTrainer координує одночасне навчання всіх агентів системи і підтримує три режими взаємодії між ними. У режимі independent кожен автобус має власну DQN модель і навчається незалежно від інших. У

режимі shared всі автобуси одного маршруту використовують спільну модель, що зменшує кількість параметрів і прискорює навчання. У режимі cooperative існує єдина модель для всіх агентів і спільний replay buffer, а винагорода кожного агента є зваженою сумою індивідуальної та системної винагороди, як показано у лістингу 3.10.

Лістинг 3.10 – Обчислення кооперативної винагороди

```
system_reward = self.env.reward_fn.compute_system_reward(
    [b.to_dict() for b in self.env.buses.values()],
    self.env.flow.get_all_queue_sizes(),
    info.get("passengers_left", 0))
rewards = {bid: r * 0.5 + system_reward * 0.5
            for bid, r in rewards.items() }
```

Кожен крок симуляції в методі `train_episode()` виконується у чіткій послідовності з чотирьох етапів. Спочатку `ControlAgent` кожного маршруту аналізує поточні позиції автобусів і формує рекомендації. Потім кожен `BusAgent` отримує спостереження, запитує допустимі дії у `ActionSpace` і обирає дію через DQN. Якщо `ControlAgent` виявив `convoy` і сформував рекомендацію для цього автобуса, вона замінює дію моделі. Послідовність кроку наведено у лістингу 3.11.

Лістинг 3.11 – Послідовність дій на кожному кроці

```
ctrl.observe({"bus_states": route_buses})
recs = ctrl.act()
raw_action = model.act(obs, valid)
final_action = ctrl_recommendations.get(bus_id, raw_action)
```

Після виконання дій середовище повертає нові спостереження і винагороди. Кожен перехід (`obs`, `action`, `reward`, `next_obs`, `done`) зберігається у відповідний replay buffer, і одразу викликається `model.update()` для кроку навчання. Наприкінці епізоду збираються агреговані метрики: сумарна і середня винагороди, коефіцієнт обслуговування пасажирів, середній час

очікування, кількість convoу-подій та метрика рівномірності розподілу автобусів.

Запуск навчання виконується через CLI-скрипт `train.py`, який приймає параметри алгоритму, кількість епізодів, сценарій симуляції та режим MARL. Основні параметри запуску наведено у лістингу 3.12.

Лістинг 3.12 – Запуск MARL навчання через CLI

```
python train.py --algo marl --episodes 300 --mode shared --save-  
every 100
```

У проєкті зберігаються чекпоінти кожні 100 епізодів у директорії `models/`: `marl_ep100/`, `marl_ep200/`, `marl_ep300/` і фінальна модель `marl_final/`. Кожна директорія містить окремі файли `dqn_R01.pt` - `dqn_R10.pt` для кожного маршруту. Таке збереження дозволяє порівнювати якість навчання на різних стадіях та відновлювати навчання з будь-якого чекпоінту.

Після завершення навчання метод `evaluate()` запускає задану кількість епізодів у повністю жадібному режимі без дослідження середовища, усереднює сумарну винагороду і коефіцієнт обслуговування та повертає підсумкову статистику у вигляді словника з полями `avg_reward`, `std_reward` та `avg_service_rate`.

3.2.4 Порівняння реалізованих алгоритмів

Для оцінки ефективності реалізованих алгоритмів проведено порівняльний експеримент на основі 10 епізодів навчання в однакових умовах симуляції. Одноагентні алгоритми (Q-learning, Policy Gradient, Actor-Critic) тестувались у режимі одного керованого агента з 30 автобусів, тоді як MARL/DQN навчав усі 30 агентів одночасно, що відповідає основній постановці задачі роботи (таблиця 3.2).

Таблиця 3.2 – Порівняння алгоритмів за ключовими метриками (10 епізодів)

Алгоритм	Середня винагорода	Service Rate	Сер. час очікування, кроків
Q-learning	9483	0.367	30.0
Policy Gradient	9678	0.373	29.8
Actor-Critic	9677	0.373	29.7
MARL/DQN (shared)	6782	0.024	26.6

Нижча абсолютна винагорода MARL/DQN пояснюється складністю задачі: замість одного агента система навчає 30 агентів одночасно в конкурентному середовищі, що суттєво ускладнює процес навчання на перших епізодах. Водночас лише MARL/DQN забезпечує скоординоване управління всіма маршрутами і саме для цього алгоритму проведено повноцінне навчання протягом 300 епізодів.

3.3 Реалізація серверної та клієнтської частини

3.3.1 REST API та WebSocket-сервер

Серверна частина системи реалізована на базі фреймворку FastAPI і запускається через ASGI-сервер Uvicorn. Точка входу – функція `create_app()` у `app.py`, яка конфігурує додаток, реєструє `middleware`, підключає REST-роутер і WebSocket-ендпоінти. Ініціалізація і завершення роботи сервера керуються через `lifespan`-менеджер контексту, що замінює застарілий механізм `on_event`. При старті ініціалізується сесія симуляції і асинціо-замок, при зупинці – автоматично зупиняються потоки симуляції та навчання, як показано у лістингу 3.13.

Відповідно до технічного завдання до роботи передбачалось використання СУБД PostgreSQL. У процесі реалізації прийнято рішення зберігати стан симуляції та результати навчання у форматах JSON і CSV замість реляційної бази даних. Це рішення обґрунтоване тим, що дані

симуляції є тимчасовими і переважно передаються клієнту через WebSocket у реальному часі, а не зберігаються для подальшого аналізу.

Лістинг 3.13 – Lifespan-менеджер серверного додатку

```
@asynccontextmanager
async def lifespan(app: FastAPI):
    session = get_session()
    session.init_async_lock()
    yield
    session.stop_simulation()
    session.stop_training()
```

Файли JSON і CSV забезпечують достатню швидкість читання/запису без накладних витрат на підтримку з'єднання з БД та міграції схеми. Інтеграція PostgreSQL залишається актуальним напрямом розвитку системи для зберігання історії навчання та агрегованої статистики у виробничому середовищі.

Усі REST-ендпоінти зареєстровані у routes.py під префіксом /api і поділяються на п'ять функціональних груп. Група мережі повертає GeoJSON-представлення зупинок і маршрутів для відображення на карті. Група симуляції керує запуском, зупинкою, паузою і відновленням симуляційного циклу. Група навчання запускає і зупиняє процес навчання агентів та повертає його поточний стан і історію. Група моделей забезпечує збереження і завантаження навчених ваг. Група метрик надає поточний знімок стану (/api/snapshot), дані для графіків (/api/charts) і агреговану статистику.

Усі вхідні і вихідні дані описані Pydantic-схемами у schemas.py. Запити на запуск симуляції та навчання валідуються через StartSimRequest і StartTrainingRequest, де перераховані допустимі значення алгоритму (qllearning, dqn, pg, ac, marl), сценарію і MARL-режиму. Відповідь на більшість POST-запитів повертає StatusResponse з полями ok, message і необов'язковим data. Snapshot-відповідь містить списки об'єктів BusData і

StopData та об'єкт MetricsData з поточними показниками симуляції. Повний перелік ендпоінтів наведено у таблиці 3.3.

Таблиця 3.3 – REST-ендпоінти API

Метод	Ендпоінт	Призначення
GET	/api/health	Перевірка доступності сервера
GET	/api/network	GeoJSON зупинок і маршрутів
GET	/api/snapshot	Поточний стан симуляції
GET	/api/charts	Дані для графіків
GET	/api/metrics/summary	Агрегована статистика
GET	/api/training/status	Стан процесу навчання
GET	/api/training/history	Історія епізодів навчання
POST	/api/simulation/start	Запуск симуляції
POST	/api/simulation/pause	Пауза симуляції
POST	/api/simulation/stop	Зупинка симуляції
POST	/api/training/start	Запуск навчання
POST	/api/training/stop	Зупинка навчання
POST	/api/scenario/change	Зміна сценарію
POST	/api/model/save	Збереження моделі
POST	/api/model/load	Завантаження моделі

Для передачі даних у реальному часі реалізовано два WebSocket-ендпоінти: /ws/simulation (стан системи та команди керування кожні 0.5 с) і /ws/training (прогрес навчання та історія щосекунди).

Управління підключеннями реалізовано у класі ConnectionManager через множину активних WebSocket-з'єднань і asyncio-замок для потокобезпечного доступу, як показано у лістингу 3.14.

Лістинг 3.14 – Широкомовне надсилання даних клієнтам

```

async def broadcast(self, data: dict):
    message = json.dumps(data, ensure_ascii=False)
    async with self._get_lock():
        clients = set(self._clients)
        dead = set()
        for ws in clients:
            try: await ws.send_text(message)
            except Exception: dead.add(ws)
        if dead: async with self._get_lock(): self._clients -= dead

```

Клієнти що відключились під час broadcast автоматично видаляються з множини активних з'єднань без переривання надсилання іншим. CORS-middleware налаштовано для дозволу запитів з портів 5173 і 3000, де запускається фронтенд-застосунок у режимі розробки. У production-режимі зібраний фронтенд монтується як статичні файли з підтримкою SPA-роутингу через fallback на index.html.

3.3.2 Клієнтська частина на React

Клієнтська частина системи реалізована як односторінковий застосунок на React з використанням Vite як інструменту збірки і Tailwind CSS для стилізації. Глобальний стан застосунку керується через Zustand-стор simulationStore, який зберігає поточні дані автобусів, зупинок, метрик, статус симуляції та навчання, GeoJSON мережі, повну історію навчання і параметри UI. Інтеграція з бекендом відбувається через два канали: REST API для одноразових запитів і WebSocket для потокових даних у реальному часі.

Підключення до WebSocket-сервера реалізовано у двох хуках. Хук useSimulationWS встановлює з'єднання з /ws/simulation, завантажує GeoJSON мережі при першому підключенні і оновлює стан автобусів, зупинок та метрик при кожному snapshot. При відключенні хук автоматично перепідключається через 3 секунди.

Через те саме з'єднання надсилаються команди керування симуляцією (pause, resume, stop і change_scenario) без додаткових HTTP-запитів. Хук useTrainingWS підключається до /ws/training і оновлює прогрес навчання при кожному повідомленні, дописуючи нові епізоди до історії з дедуплікацією по номеру епізоду. При завершенні навчання сервер надсилає подію training_complete і з'єднання закривається. Система сповіщень відображає до п'яти повідомлень одночасно і автоматично прибирає їх через 4 секунди.

Застосунок складається з чотирьох сторінок між якими перемикання відбувається через бічну навігаційну панель.

Сторінка Dashboard є головним екраном моніторингу (рисунок А.1). У лівій панелі розміщено панель керування симуляцією, блок деталей вибраного автобуса і теплокарта завантаженості маршрутів. Індикатор статусу у верхній частині панелі показує назву активного сценарію під час симуляції або номер поточного епізоду під час навчання. У центральній частині відображаються картки метрик і інтерактивна карта.

Картки метрик відображають шість показників у реальному часі: середній час очікування на зупинках, середню завантаженість автобусів, коефіцієнт обслуговування пасажирів, кількість людей у чергах, кількість пасажирів що не змогли сісти та кількість convoу-подій. Колір кожного показника динамічно змінюється відповідно до порогових значень – зелений при нормальному стані, жовтий при підвищеному навантаженні, червоний при критичному.

На інтерактивній карті відображаються маршрути, зупинки і автобуси. Тролейбусні маршрути відрізняються кольором від автобусних. Маркер зупинки збільшується якщо черга перевищує 30 осіб, а його колір відображає завантаженість за чотирирівневою шкалою від зеленого до червоного. Колір маркера автобуса відображає рівень його завантаженості за тією самою шкалою. При кліку на автобус у лівій панелі відображаються його деталі: маршрут, поточна зупинка, кількість пасажирів, відсоток завантаженості і поточна винагорода. При наведенні на будь-який маркер з'являється підказка з ключовими показниками.

Теплокарта маршрутів відображає горизонтальні індикатори завантаженості для кожного з 10 маршрутів з оновленням кожні 2 секунди. Колір індикатора відповідає тій самій шкалі що і маркери на карті.

Панель керування у режимі симуляції дозволяє вибрати сценарій і швидкість відтворення. Набір доступних кнопок адаптується до поточного стану: Старт при зупиненій симуляції, Стоп і Пауза при активній, Стоп і

Продовжити при паузі. Сценарій можна змінювати під час активної симуляції без перезапуску.

Система підтримує вісім сценаріїв симуляції. Сценарій *normal* моделює звичайний робочий день з базовим пасажиропотоком (рисунок А.3). Сценарій *morning_peak* підвищує інтенсивність пасажиропотоку у п'ять разів відносно базового у проміжку з 7:00 до 9:30 (рисунок А.4). Сценарій *evening_peak* аналогічно підвищує потік у 4.5 рази з 16:30 до 19:00 (рисунок А.5). Сценарій *night* знижує пасажиропотік до 15% від базового і скорочує кількість активних автобусів на маршруті до одного (рисунок А.6). Сценарій *accident* випадково блокує один із маршрутів, перерозподіляє пасажирів на інші маршрути і знижує швидкість руху через затори (рисунок А.7). Сценарій *mass_event* генерує додатковий потік пасажирів на випадково обраній центральній зупинці і підвищує загальний потік до 1.5 від базового в нічний час (рисунок А.8). Сценарій *rain* підвищує попит на транспорт і знижує швидкість руху (рисунок А.9). Сценарій *reduced* моделює вихідний день зі зниженим пасажиропотоком і скороченою кількістю транспортних засобів (рисунок А.10).

Сторінка *Training* призначена для керування процесом навчання агентів (рисунок А.11). У лівій панелі розміщено вибір алгоритму навчання з п'яти варіантів, вибір MARL-режиму при обраному алгоритмі MARL, вибір сценарію і налаштування кількості епізодів. Всі елементи керування блокуються під час активного навчання. У правій частині відображаються прогрес-бар з поточними показниками навчання, графіки динаміки *reward* і *service rate* по епізодах та таблиця останніх 20 епізодів з детальною статистикою.

Сторінка *Simulation* орієнтована на детальний аналіз поточного кроку симуляції (рисунок А.2). При кліку на зупинку на карті у лівій панелі відображаються її характеристики: назва, зона, поточний розмір черги і середній час очікування. Додатково у лівій панелі відображається графік динаміки винагороди поточної сесії.

Сторінка Analytics надає розгорнутий аналіз результатів навчання (рисунок А.12). При завантаженні вона запитує повну історію навчання з сервера і використовує серверні дані пріоритетно над локальними. Сторінка містить графік reward і service rate по епізодах; radar-діаграму профілю агента з п'яти нормалізованих метрик – ефективність винагороди, коефіцієнт обслуговування, мінімізація очікування, використання місткості і відсутність convoу-подій; графік convoу-подій по епізодах; графік таймсерій симуляції з часом очікування і service rate по кроках; графік пасажирів що не змогли сісти по епізодах.

3.4 Реалізація серверної та клієнтської частини

Тестування системи виконувалось на двох рівнях. Модульне тестування охоплює п'ять наборів тестів реалізованих засобами pytest:

- test_env.py перевіряє коректність ініціалізації середовища, виконання кроків симуляції та скидання стану;
- test_agents.py тестує поведінку агентів при різних вхідних спостереженнях;
- test_reward.py верифікує правильність обчислення компонентів функції винагороди;
- test_flow.py перевіряє генерацію пасажиропотоку і логіку посадки та висадки;
- test_gtfs_parser.py тестує коректність завантаження GTFS-даних.

Інтеграційне тестування виконувалось шляхом запуску повного циклу симуляції через REST API і WebSocket з перевіркою відповідності даних між серверною і клієнтською частинами.

Оцінка ефективності навченої MARL-моделі проводилась шляхом порівняння ключових метрик до і після навчання протягом 100 епізодів. Коефіцієнт обслуговування пасажирів зріс з 0.007 до 0.069 – в десять разів. Кількість convoу-подій знизилась з 3569 до 3428. Середній час очікування

пасажирів зменшився з 49.7 до 48.8 кроків. Radar-діаграма профілю агента на рисунку А.12 демонструє що система досягла найкращих показників за метриками reward і service rate, тоді як показники convoy і capacity залишаються низькими. Низький рівень convoy-оптимізації зумовлений недостатнім штрафом за convoy у функції винагороди, тому агенти зосередились на максимізації посадки пасажирів, а не на рівномірному розподілі по маршруту.

Зниження кількості convoy-подій виявилось незначним: з 3569 до 3428 (–4%). Це пояснюється тим, що штраф за convoy у функції винагороди виявився недостатнім сигналом порівняно з бонусом за обслуговування пасажирів – агенти навчилися максимізувати посадку пасажирів, ігноруючи рівномірний розподіл по маршруту. Для усунення цього дисбалансу в подальшому необхідно збільшити вагу штрафу за convoy або застосувати окрему функцію винагороди для координуючого агента ControlAgent.

Низький показник capacity пояснюється обмеженням пасажиропотоком у навчальному середовищі, де для стабільності навчання використовувався коефіцієнт масштабування 0.15 від реального ОД-потoku. Через це автобуси не могли досягти рівня заповненості, який система вважає ефективним. Подальше вдосконалення моделі можливе шляхом збільшення штрафу за convoy та поступового нарощування пасажиропотоку під час навчання за методом curriculum learning.

Щодо коефіцієнта обслуговування 6.9%, його невисоке абсолютне значення пояснюється масштабуванням пасажиропотоку до 15% від реального: у навчальному середовищі генерувалось близько 14 тис. пасажирів замість 330 тис. За таких умов service rate обмежувався тривалістю епізоду та максимальним часом очікування пасажирів. Водночас зростання показника з 0.007 до 0.069 свідчить про реальне покращення стратегії агентів у межах симуляції.

ВИСНОВКИ

У кваліфікаційній роботі розроблено мультиагентну систему оптимізації навантаження громадського транспорту на основі алгоритмів підкріплювального навчання. Проведений аналіз предметної галузі дозволив встановити що нерівномірний розподіл пасажиропотоків та явище зграювання транспортних засобів є ключовими проблемами міських транспортних систем, а мультиагентні системи з підкріплювальним навчанням є перспективним підходом до їх вирішення.

Спроековано архітектуру мультиагентної системи що включає агентів-автобусів, агентів-зупинок, агентів-пасажирів та центрального агента управління. Визначено простір станів з 11 ознаками, простір дій з 4 можливими рішеннями та функцію винагороди що враховує час очікування пасажирів, завантаженість автобусів та кількість convoу-подій.

Реалізовано середовище симуляції транспортної мережі міста Харкова що включає 10 маршрутів, 30 зупинок та 30 автобусів з генерацією пасажиропотоків на основі OD-матриці з урахуванням часових піків та різних сценаріїв роботи системи.

Реалізовано чотири алгоритми навчання агентів - Q-learning, DQN з Double DQN, Policy Gradient та Actor-Critic - а також MARL-тренер з підтримкою спільного буфера досвіду для мультиагентного навчання. Розроблено серверну частину на базі FastAPI з REST API та WebSocket і клієнтський інтерфейс на React з інтерактивною картою маршрутів, панеллю керування навчанням та сторінкою аналітики.

За результатами навчання MARL-моделі протягом 100 епізодів коефіцієнт обслуговування пасажирів зріс з 0.007 до 0.069 – в десять разів порівняно з базовим рівнем, кількість convoу-подій знизилась з 3569 до 3428, а середній час очікування пасажирів зменшився з 49.7 до 48.8 кроків симуляції.

Напрямами подальшого розвитку системи є впровадження механізму динамічного перерозподілу автобусів між маршрутами, застосування curriculum learning для поступового нарощування складності навчального середовища та інтеграція з реальними даними телематики транспортних засобів.

Низький абсолютний показник коефіцієнта обслуговування 6.9% зумовлений використанням масштабного коефіцієнта 0.15 від реального пасажиропотоку OD-матриці для забезпечення стабільності навчання. В умовах обмеженого потоку відносно зростання показника в десять разів відображає реальне покращення стратегії агентів, а не абсолютну пропускну здатність системи. Зниження кількості convoу-подій на 4% є незначним через дисбаланс у функції винагороди – агенти пріоритизували посадку пасажирів над рівномірним розподілом по маршруту.

Розроблена система пройшла модульне тестування засобами pytest, що охоплює перевірку середовища симуляції, логіки агентів, функції винагороди, генерації пасажиропотоку та завантаження GTFS-даних. Інтеграційне тестування підтвердило коректність взаємодії між серверною та клієнтською частинами через REST API і WebSocket. Реалізовані вісім сценаріїв симуляції – від звичайного робочого дня до аварійної ситуації з перекриттям маршруту – дозволяють оцінювати поведінку навчених агентів в різних умовах транспортного навантаження.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Urban transit systems and technology. Wiley & Sons, Incorporated, John, 2007.
2. Rodrigue J.-P., Comtois C., Slack B. The geography of transport systems. Routledge, 2016. URL: <https://doi.org/10.4324/9781315618159> (date of access: 19.06.2026).
3. Fundamentals of intelligent transportation systems planning (artech house its library). Artech House Publishers, 2003. 210 p.
4. Urban transportation networks: equilibrium analysis with mathematical programming methods. Englewood Cliffs, N.J : Prentice-Hall, 1984. 399 p.
5. Thalanany S. Mobile evolution. CRC Press, 2015. URL: <https://doi.org/10.1201/b18137> (date of access: 19.06.2026).
6. Papageorgiou M., Kotsialos A. Freeway ramp metering: an overview. *IEEE transactions on intelligent transportation systems*. 2002. Vol. 3, no. 4. P. 271–281. URL: <https://doi.org/10.1109/tits.2002.806803> (date of access: 19.06.2026).
7. Russell S. J., Norvig P. Artificial intelligence: a modern approach. Pearson Publishing, 2021. 1170 p.
8. Sutton R. S., Barto A. G. Reinforcement learning: an introduction. A Bradford Book, 2018. 552 p.
9. An introduction to multiagent systems. 2nd ed. Hoboken, NJ : John Wiley & Sons, 2009. 461 p.
10. Innovations in multi-agent systems and applications – 1 / ed. by D. Srinivasan, L. C. Jain. Berlin, Heidelberg : Springer Berlin Heidelberg, 2010. URL: <https://doi.org/10.1007/978-3-642-14435-6> (date of access: 19.06.2026).
11. Stable-Baselines3 Docs – Reliable Reinforcement Learning Implementations – Stable Baselines3 2.9.0 documentation. *Stable-Baselines3*. URL: <https://stable-baselines3.readthedocs.io/en/master/> (date of access: 19.06.2026).

12. SUMO documentation. *Eclipse SUMO – Simulation of Urban MObility*. URL: <https://sumo.dlr.de/docs/index.html> (date of access: 19.06.2026).

13. PostgreSQL: documentation. *PostgreSQL: The world's most advanced open source database*. URL: <https://www.postgresql.org/docs/> (date of access: 19.06.2026).

14. FastAPI framework, high performance, easy to learn, fast to code, ready for production. *FastAPI*. URL: <https://fastapi.tiangolo.com/> (date of access: 17.06.2026).

15. PyTorch Contributors. PyTorch documentation. *PyTorch documentation*. URL: <https://docs.pytorch.org/docs/2.12/index.html> (date of access: 17.06.2026).