НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

«На правах рукопису»
УДК 004.94, 656.052

До захисту допущено:
В.о. завідувача кафедри
_____ Ірина ДЖИГИРЕЙ
«___»_____2024 р.

Магістерська дисертація

на здобуття ступеня магістра

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

зі спеціальності 122 «Комп'ютерні науки»

на тему: «Інтелектуальна система керування автономним транспортним засобом»

Виконав:
студент ІІ курсу, групи КІ-21мп
Молнар Андрій Сергійович _____

Науковий керівник:
проф. кафедри ШІ, д.т.н., професор
Синєглазов Віктор Михайлович _____

Рецензент:

_____

Засвідчую, що у цій магістерській дисертації
немає запозичень з праць інших авторів
без відповідних посилань
Студент: _____

Київ – 2024

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – другий (магістерський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ

В.о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«___»_____2024 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Молнару Андрію Сергійовичу

1. Тема дисертації «Інтелектуальна система керування автономним транспортним засобом», науковий керівник дисертації д.т.н., професор Синєглазов Віктор Михайлович, затверджені наказом по університету від «08» листопада 2023 р. №5200-с

2. Термін подання студентом дисертації 07.01.2024.

3. Об'єкт дослідження: автономні транспортні засоби

4. Предмет дослідження: моделі прогнозування траєкторії для автономних транспортних засобів

5. Перелік завдань, які потрібно розробити:

Затвердження теми МД

1) Ознайомлення зі структурою МД згідно з Положенням про державну атестацію студентів НТУУ «КПІ ім. І. Сікорського».
2) Ознайомлення з ДСТУ 3008-2015 та стандартами ЄСПД.
3) Проведення дослідження за темою МД під керівництвом керівника.
4) Завершення роботи над першою і другою частинами МД.
5) Підготовка інших розділів роботи.
6) Оформлення дипломної роботи.

7) Підготовка презентації доповіді.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: презентація

7. Дата видачі завдання 30.08.2023

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Затвердження теми МД	01.09.2023-10.09.2023	Виконано
2	Ознайомлення зі структурою МД згідно з Положенням про державну атестацію студентів НТУУ «КПІ ім. І. Сікорського»	11.09.2023-15.09.2023	Виконано
3	Ознайомлення з ДСТУ 3008-2015 та стандартами ЄСПД	16.09.2023-20.09.2023	Виконано
4	Проведення дослідження за темою МД під керівництвом керівника	21.09.2023-15.10.2023	Виконано
5	Завершення роботи над першою і другою частинами МД	16.10.2023-25.10.2023	Виконано
6	Підготовка інших розділів роботи	26.10.2023-10.12.2023	Виконано
7	Оформлення дипломної роботи	11.12.2023-01.01.2024	Виконано
8	Підготовка презентації доповіді	02.01.2024-05.01.2024	Виконано

Студент Андрій МОЛНАР

Науковий керівник Віктор СИНЄГЛАЗОВ

РЕФЕРАТ

Дипломна робота: 120 с., 37 рис., 13 табл., 1 додаток, 66 посилань.

У XX столітті автомобілі стали масовим явищем завдяки швидкій індустріалізації та урбанізації. Збільшення кількості транспортних засобів на дорогах призвело до значного збільшення дорожньо-транспортних пригод, які нерідко призводять до важких травм або смерті. За статистикою, переважна більшість ДТП відбувається через помилки водія. З цього випливає актуальність впровадження самокерованих автомобілів, як способу значного зменшення кількості ДТП. Останніми роками ця галузь зазнала безпрецедентного зростання. Незважаючи на прогрес, все ще існують проблеми, які необхідно вирішити. Прогнозування майбутніх траєкторій оточуючих об'єктів має важливе значення для прийняття безпечних рішень. Це складне завдання, оскільки агенти є різноманітними і їхня поведінка впливає один на одного. Більшість існуючих моделей або занадто неефективні для обчислень у реальному часі, або мають недостатню точність.

Об’єктом дослідження є автономні транспортні засоби. Предметом дослідження є моделі прогнозування траєкторії для автономних транспортних засобів.

Метою роботи є дослідження та порівняння методів та моделей прогнозування траєкторії учасників дорожнього руху для автономних транспортних засобів і створення нової ефективної моделі на основі дослідження.

Наукова новизна полягає у створенні нової моделі прогнозування траєкторії, що є точною та ефективною в обчисленнях одночасно.

НЕЙРОННІ МЕРЕЖІ, АВТОНОМНІ ТРАНСПОРТНІ ЗАСОБИ, САМОКЕРОВАНЕ АВТО, ПРОГНОЗУВАННЯ РУХУ, ПРОГНОЗУВАННЯ ТРАЄКТОРІЇ, ТРАНСФОРМЕР

ABSTRACT

Diploma thesis: 120 p., 37 figures, 13 tables, 1 appendix, 66 references.

In the XX century, cars became a mass phenomenon due to rapid industrialization and urbanization. The increase in the number of vehicles on the roads has led to a significant increase in road accidents, which often lead to serious injury or death. According to statistics, the vast majority of road accidents are caused by driver errors. This makes the introduction of self-driving cars highly relevant, since it's expected to significantly reduce the number of road accidents. In recent years, the field has witnessed unprecedented growth, fueled by rapid advancements in sensor technologies, computational power, and machine learning. Despite the progress, there are still challenges that need to be addressed. Predicting the future trajectories of surrounding objects is essential for making safe decisions. This is a challenging task because agents are diverse, and their behavior affects each other. Most of the existing models are either too inefficient for real-time computing or offer insufficient accuracy.

The object of research is autonomous vehicles. The subject of the study is trajectory prediction models for autonomous vehicles.

The purpose of the study is research and comparison of methods and models for predicting the trajectory of road users for autonomous vehicles and creation of a new effective model based on the research.

The scientific novelty lies in creating a new model for trajectory prediction that is accurate and computationally efficient at the same time.

NEURAL NETWORKS, AUTONOMOUS VEHICLES, SELF-DRIVING CARS, MOTION PREDICTION, TRAJECTORY PREDICTION, TRANSFORMER

ЗМІСТ

ВСТУП

Історія автомобіля бере свій початок ще у 18 столітті із зростанням популярності парової машини. Винахідником сучасного автомобіля прийнято вважати німецького інженера Карла Бенца, який створив перший серійний автомобіль із двигуном внутрішнього згорання у 1885 році [1].

У 20 столітті автомобілі стали масовим явищем завдяки розвитку промисловості, пов’язаній з ним урбанізації та зростанню міського населення, яке потребувало швидкий і зручний вид транспорту. Проте, зростання кількості транспортних засобів та інтенсивності дорожнього руху призвело і до збільшення порушень правил дорожнього руху, які призводять до дорожньо-транспортних пригод, травм та смерті.

Тоді ж (ще в 1920-х) почалися перші дослідження в області створення самокерованих автомобілів, які могли б вирішити цю та низку інших проблем. Недостатній для створення економічно доцільного повністю самокерованого автомобіля рівень технологій, кризи та війни зумовили зниження інтересу до самокерованих автомобілів на деякий час. Проте інтерес відновився завдяки розвитку мікропроцесорів та комп’ютерних технологій наприкінці 20-го століття. Зараз самокеровані автомобілі перебувають на стадії активних розробок одними із найбільших корпорацій у світі. Деякі з них вже проходять тести на дорогах загального користування.

Інтерес до самокерованих автомобілів обумовлений кількома факторами. По-перше, самокеровані автомобілі можуть значно підвищити рівень безпеки на дорогах та зменшити кількість жертв ДТП завдяки більш швидкому реагуванню на загрози. По-друге, вони можуть підвищити ефективність дорожнього руху. Самокеровані автомобілі можуть координувати свій рух з іншими автомобілями, що може призвести до

зменшення кількості заторів [2]. По-третє, вони можуть зробити транспорт більш доступним для людей із обмеженими можливостями.

Головною характеристикою самокерованих автомобілів є їх здатність до самостійного прийняття рішень. Одним із найважливіших етапів у цьому є здатність передбачувати майбутню траєкторію учасників дорожнього руху за допомогою аналізу навколишнього середовища. Це складне завдання, оскільки агенти (наприклад, транспортні засоби та пішоходи) є різноманітними і їхня поведінка впливає один на одного. Останніми роками ця галузь зазнала безпрецедентного зростання, спричиненого швидким розвитком сенсорних технологій, обчислювальних потужностей та машинного навчання. Незважаючи на прогрес, досягнутий за останні роки, все ще існують проблеми, які необхідно вирішити. Більшість існуючих моделей або занадто неефективні для обчислень у реальному часі, або мають недостатню точність. Ця робота дає загальний огляд існуючих підходів, аналізує їх переваги та недоліки, а також пропонує модель, яка може поєднати високу швидкість прогнозувань із високою точністю у тестах.

РОЗДІЛ 1 АНАЛІЗ СФЕРИ ПРОГНОЗУВАННЯ ТРАЄКТОРІЇ

1.1 Огляд літератури у сфері прогнозування траєкторії

Розвиток технологій автономних (також відомих як самокерованих) автомобілів починається у 20-х роках минулого століття. У 1924 році американська компанія Houdina Radio Control показала радіокерований автомобіль, який пересувався вулицями Нью-Йорка. Цей же винахід під іншою назвою демонстрували у 1926 у Мілвокі та у 1932 на вулицях Фредеріксбургу [3]. Перші спроби не були самокерованими у строгому сенсі слова, оскільки автомобілем все одно керував оператор за допомогою радіоімпульсів.

У 1950-х було створено перші справжні автономні автомобілі. Оскільки комп'ютери того часу були занадто великі та повільні для обробки великої кількості інформації у режимі реального часу, ці автомобілі керувалися сигналами від спеціальних сенсорів та маячків, що були вбудовані в саму дорогу. Американські RCA Labs у співпраці з General Motors у 1953 році створили систему із мініатюрним автомобілем, який керувався сигналами від підлоги лабораторії. У 1957 вони успішно продемонстрували повномасштабний варіант системи на відрізку шосе довжиною 122 м. Проте такий підхід був занадто непрактичним, оскільки вимагав оснащення кожної дороги у країні дороговартісними електронними компонентами [4, 5].

У 1980-1990-х було розроблено системи контролю, які є більш схожими на сучасні. Роботизований фургон компанії Mercedes-Benz розвинув швидкість 96 км/год на вулиці без дорожнього руху. Керувався він із використанням технологій комп'ютерного зору [6]. У цьому ж десятилітті DARPA профінансувала проект «Автономного наземного транспортного засобу» (англ. Autonomous Land driven Vehicle) або ALV. В рамках проекту

була проведена перша демонстрація руху по дорозі із використанням LIDAR, комп'ютерного зору і автономного роботизованого управління для керування транспортним засобом. До 1989 року Університет Карнегі-Мелон став піонером у використанні нейронних мереж для контролю автономних транспортних засобів, що є основою більшості сучасних стратегій керування [7].

На сьогоднішній день дослідження в галузі самокерованих транспортних засобів активно продовжуються. Було створено численну кількість робіт, заснованих на різних принципах. Для ефективного огляду має сенс дати класифікацію цих робіт. Існує декілька способів класифікації моделей. Розглянемо класифікації запропоновані Лефвре та ін. [8] і Мозаффарі та ін. [9].

За типом гіпотез про агентів, траєкторія яких моделюється, можна поділити методи на 3 типи із зростаючим рівнем складності та абстракції:

1. Фізичні моделі руху (англ. Physics-based motion models) – найпростіші моделі. Вони припускають, що рух транспортних засобів залежить лише від законів фізики. Існують різні види фізичних моделей: від простих моделей постійної швидкості (англ. Constant Velocity) або прискорення (англ. Constant Acceleration) [10, 11], які передбачають прямолінійний рух транспортних засобів, до більш складних динамічних моделей, що описують рух за допомогою рівнянь Лагранжа та враховують різні сили, що впливають на рух автомобіля [13, 14]. Оскільки фізичні моделі покладаються лише на низькорівневі властивості руху, то вони обмежені лише короткостроковим (менше секунди) прогнозуванням руху. Вони не здатні передбачити зміни в русі автомобіля, спричинені виконанням певного маневру (наприклад уповільнення, поворот із постійною швидкістю і прискорення для повороту на перехресті) або зміни, спричиненні зовнішніми факторами (наприклад, уповільнення через

транспортний засіб попереду). До їх переваг можна віднести простоту та ефективність обчислень.

2. Моделі руху на основі маневрів (англ. Maneuver-based motion models) – більш досконалі моделі, оскільки вони враховують, що рух автомобіля може залежати від маневру, який водій намагається виконати. Такі модель засновані або на використанні прототипних траєкторій або оцінці намірів маневру. Ідея прототипних траєкторій полягає в тому, що набори траєкторій руху можна згрупувати у скінченну кількість кластерів, де кожен кластер відповідає типовому маневру. Під час прогнозування можна використовувати шляхом пошуку найбільш схожого кластеру з тренувальних даних [14, 15]. Альтернативним підходом є оцінка наміру водія і прогнозування послідовних фізичних станів, які відповідають можливому виконанню маневру [16, 17]. Перевагою над використанням прототипів є відсутність необхідності співставляти часткову траєкторію із тренувальними даними. Моделі руху на основі маневрів більш надійні ніж фізичні у довгостроковому прогнозуванні руху, проте не враховують вплив інших транспортних засобів на рух один одного.

3. Моделі руху з урахування взаємодії (англ. Interaction-aware models) – найскладніші моделі, які враховують взаємозалежність між маневрами різних агентів. До 2014 р. таких моделей було небагато і більшість з них були засновані на використанні динамічних баєсових мереж (англ. Dynamic Bayesian Network) [17, 18]. Проте останні досягнення в методах машинного навчання (насамперед глибокого навчання) надають нові потужні інструменти для створення таких моделей і саме моделі з урахування взаємодії є найбільш перспективними.

Різницю у поведінці моделей різного типу можна побачити на рис. 1.1 Фізична модель прогнозує постійну швидкість та орієнтацію, модель на основі маневрів прогнозує, що чорний автомобіль їде прямо, а синій

повертає ліворуч, а модель із урахуванням взаємодії «розуміє», що правила дорожнього руху вимагають, щоб синій автомобіль пропустив чорний.

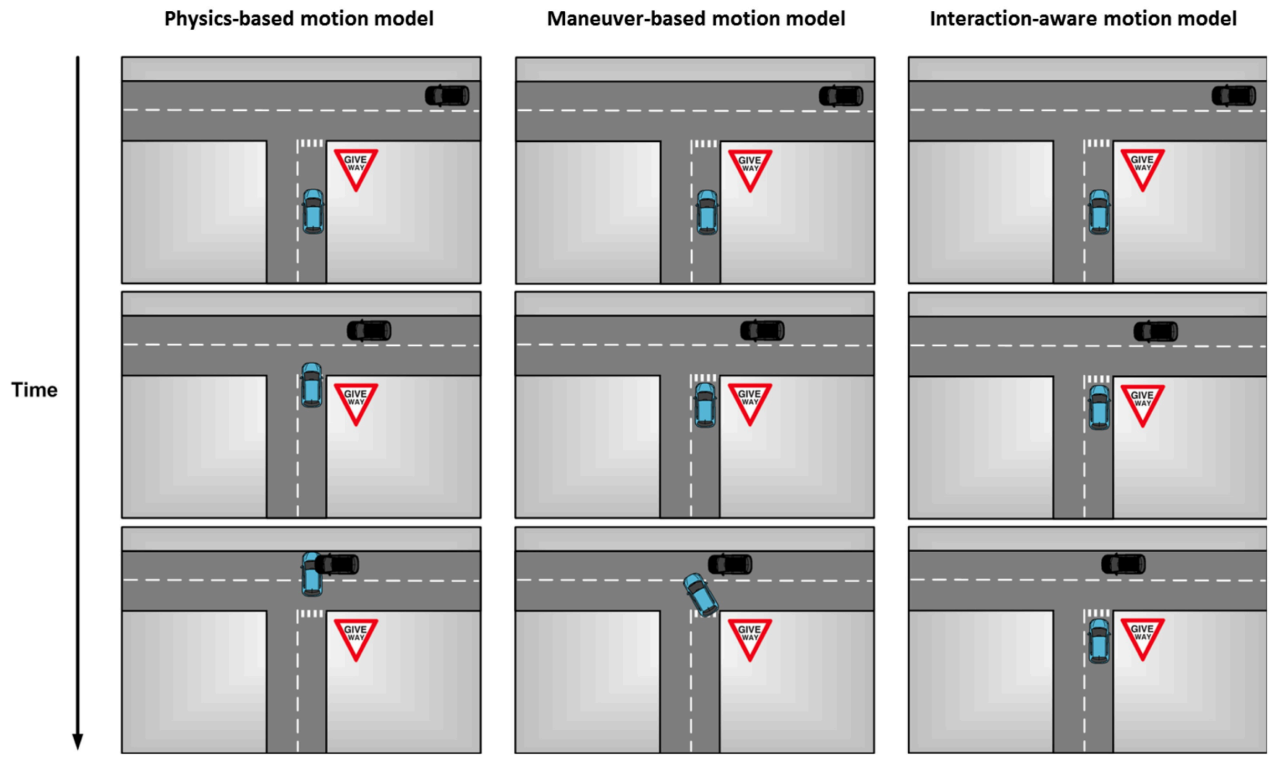

Рисунок 1.1 – Приклади прогнозування руху різними типами моделей [8]

Оскільки моделі глибокого навчання з урахування взаємодії набули широкого розповсюдження, існує необхідність в їх подальшій класифікації. Її можна провести за трьома критеріями: представлення вхідних даних, тип вихідних даних і метод прогнозування.

Ці класифікації представлено на рис. 1.2.

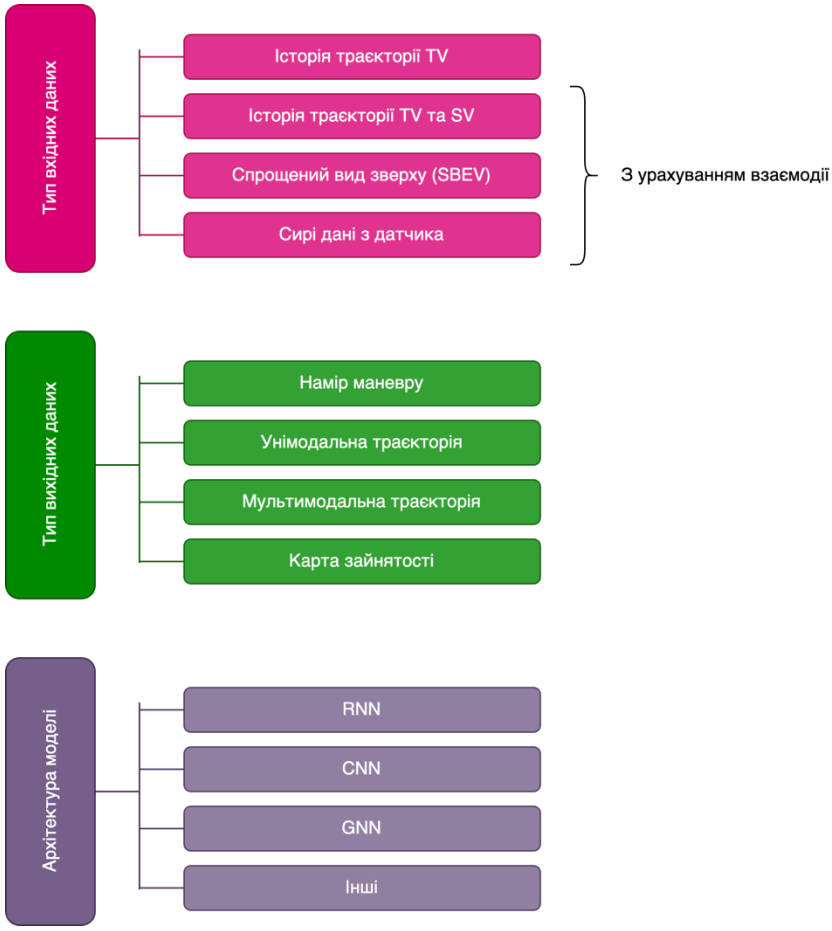

Рисунок 1.2 – Класифікації моделей прогнозування траєкторії

Представлення вхідних даних:

1. Історія траєкторії цільових транспортних засобів або TV (англ. Target Vehicles). Найпростіший підхід, який використовує минулі стани (наприклад позиція, швидкість, прискорення та напрямок) для прогнозування майбутніх. Не відноситься до моделей з урахування взаємодії.

2. Історія траєкторії цільових (TV) та навколишніх транспортних засобів (SV) (англ. Surrounding Vehicles). Один із способів врахувати взаємодію – у явному вигляді подавати історію минулих станів SV і TV моделі. Одним із недоліків такого підходу є припущення, що стани всіх SV завжди можна спостерігати, що не завжди реалістично. Крім того, на рух

можуть впливати і інші фактори – такі як умови навколишнього середовища і правила дорожнього руху.

3. Спрощений вид зверху (англ. Simplified Bird’s Eye View). При цьому підході статичні та динамічні об’єкти, дорожні смуги та інші елементи навколишнього середовища зображуються набором полігонів та ліній на зображенні. Результатом є зображення, схоже на мапу, яке зберігає розмір та положення об’єктів та геометрію дороги. При такому підході не втрачається інформація про навколишнє середовище. Проте одним із можливих недоліків залишається те, що він повністю залежить від точності від модуля сприйняття (англ. perception module), який використовувався для оцінки станів статичних і динамічних об’єктів.

4. Сирі дані з датчика. Вхідні дані містять в собі усю доступну інформацію про навколишнє середовище. Це дозволяє моделі навчитися виділяти корисні ознаки із усіх доступних сенсорних даних. Очевидним недоліком є значна потреба в обчислювальних потужностях, що може зробити такий підхід непрактичним для вбудованих у автономні транспортні засоби комп’ютери.

5. Векторизоване представлення. Моделі напряму використовують семантичні карти високої роздільної здатності (англ. HD) в якості вхідних даних, уникаючи таким чином втрати інформації при їх растеризації у вигляді BEV зображень. Такий підхід слугує проміжною ланкою між BEV та використанням сирих даних датчиків – модель отримає велику кількість даних у компактному форматі, який не вимагає надмірної обчислювальної потужності.

Класифікацію деяких існуючих робіт за даним принципом наведено у табл. 1.1.

Таблиця 1.1 – Класифікація робіт за вхідними даними

Клас	*Роботи*	*Опис*
Історія траєкторії TV	[20, 21, 22, 23, 24]	Використовується лише історія станів TV (координати, швидкість, напрям тощо)
Історія траєкторії SV	[19, 25, 26]	Історія станів TV та шести SV
	[27, 28]	Історія станів TV, 3-х найближчих SV та 4-х SV суміжних до них
	[29]	Історія станів TV та дев’яти SV
	[30]	Використовується механізм уваги (англ. attention) для зважування внеску кожного спостережуваного транспортного засобу
Спрощений вид зверху (BEV)	[31]	Послідовність 2-канальних зображень з видом зверху
	[32, 34]	BEV, елементи дороги та транспортні засоби представлені як многокутники різних кольорів
	[35, 36]	Вид зверху, розбитий на комірки. Кожна комірка містить прихований стан LSTM для відповідного транспортного засобу.
	[37, 38]	Вид зверху, розбитий на комірки. Кожна комірка містить ймовірність бути зайнятою та її швидкість.
Сирі дані з датчика	[39]	3D хмари точок за певний часовий інтервал
	[40]	Дані LIDAR та растеризована мапа
Векторизоване представлення	[63]	HD карти сцени та історія станів транспортних засобів

Тип вихідних даних:

1. Намір маневру. Прогнозування намірів маневру – задача оцінки маневру, який має здійснити TV у найближчий час. Наприклад, можливі маневри можуть включати зміну смуги, повороти ліворуч та праворуч тощо. Такі моделі зазвичай обчислювально ефективні, проте забезпечують лише поверхневий рівень розуміння поведінки автомобіля та обмежені наперед заданим набором можливих маневрів.

2. Унімодальна траєкторія. Моделі описують майбутню поведінку автомобіля шляхом прогнозування часового ряду із координатами TV. Таким чином можна отримати більш точну інформацію про траєкторію руху. Недоліком є те, що прогнозуються лише одна траєкторія, хоча у багатьох ситуаціях існує декілька ймовірних можливих траєкторій (тобто розподіл можливих траєкторій має декілька мод).

3. Мультимодальна траєкторія. На відміну від унімодальних траєкторій, цей підхід передбачає прогнозування траєкторії для кожної моди, разом із її ймовірністю. Мультимодальне прогнозування можна поділити на 2 підкласи: із статичними модами та із динамічними модами. При першому підході, множина типів поведінки задається явно і траєкторії прогнозуються для кожної із них. Недоліком є труднощі у визначенні повної множини намірів та у ручній розмітці даних. При другому підході, модель може динамічно навчатися типам поведінки, що усуває ці недоліки. Проте такі моделі складно навчати, оскільки вони часто збігаються до однієї моди.

4. Карта зайнятості. Замість прогнозування траєкторії здійснюється оцінка зайнятості кожної комірки на карті виду зверху для майбутніх часових інтервалів. Такий підхід дозволяє краще прогнозувати декілька мод, проте точність обмежується кількістю комірок на карті, а зростання кількості комірок призводить до збільшення обчислювальних затрат.

Класифікацію деяких існуючих робіт за типом вихідних даних представлено у табл. 1.2.

Таблиця 1.2 – Класифікація робіт за вихідними даними

Клас	*Роботи*	*Опис*
Намір маневру	[19]	Ймовірності лівого/правого повороту або руху прямо
	[28]	Поведінка зміни смуги руху
Унімодальна траєкторія	[42, 41, 36]	Позиції TV у часі
	[30]	Середнє та дисперсія нормального розподілу, що відповідає x та y координатам
	[39]	Положення та кут напряму TV
Мультимодальна траєкторія	[35, 25]	Розподіл траєкторії для шести наперед заданих маневрів та їх ймовірності
	[43, 22, 24]	Фіксована кількість вибірок із оціненого розподілу траєкторії
	[32]	Фіксована кількість детерміністичних траєкторій та їх ймовірності
Карта зайнятості	[37, 38]	Ймовірність зайнятості кожного пікселя BEV зображення

Архітектура моделі:

1. Рекурентні нейронні мережі або RNN (англ. Recurrent Neural Networks). Клас штучних мереж, створений для обробки послідовностей, який може використовувати внутрішню пам'ять. Найбільш поширеними типами є мережі довгої короткочасної пам’яті LSTM (англ. long short-term memory) та вентильні рекурентні вузли GRU (англ. gated recurrent unit). Ці мережі є одними із найбільш популярних у задачах пов’язаних із аналізом послідовностей даних або часових рядів, та, зокрема, у задачі

прогнозування траєкторії. Недоліком RNN є їх обмеженість у моделюванні просторових взаємозв’язків між транспортними засобами та контексту. Тому досить часто RNN використовуються разом із іншими методами для розв'язання цієї проблеми.

2. Згорткові нейронні мережі або CNN (англ. Convolutional Neural Networks). Клас глибоких штучних мереж, який успішно застосовується для аналізу зображень. В контексті задачі прогнозування траєкторії, CNN цінуються за їх здатність приймати дані у вигляді зображень, генерувати результати, схожі на зображення та зберігати просторові зв’язки при цьому. Ці можливості дозволяють моделювати взаємодію транспортних засобів і загальний контекст, а також створювати карту зайнятості. Однак недоліком 2D CNN є відсутність механізму для моделювання часових залежностей між станами транспортних засобів у різні моменти руху.

3. Графові нейронні мережі або GNN (англ. Graph Neural Networks). Клас штучних мереж, що спеціалізується на обробці даних, які можна представити у вигляді графів. CNN можна розглядати як окремий варіант графових мереж, де вузли – пікселі і тільки сусідні пікселі з’єднані ребрами графу. У контексті прогнозування траєкторій, транспортні засоби можна розглядати як вузли, а взаємодію між ними – як ребра графу.

4. Інші методи. Більшість сучасних робіт використовує певну комбінацію попередніх методів – наприклад комбінувати RNN та CNN, що дозволяє нівелювати недоліки кожного з методів, та використовувати їх сильні сторони. Також набувають популярності методи, що використовують архітектуру Трансформер.

У табл. 1.3 наведено класифікацію деяких існуючих робіт за архітектурами моделі:

Таблиця 1.3 – Класифікація робіт за архітектурами моделі

Клас	*Роботи*	*Опис*
RNN	[29]	Глибока LSTM використовується для прогнозування x,y позицій TV
	[24]	LSTM із структурою кодувальник-декодувальник
	[25]	Декілька RNN: одна група LSTM моделює індивідуальні траєкторія, інша – попарну взаємодію автомобілів
	[19]	Декілька RNN: LSTM кодує вхідну послідовність. Прихований стан кодувальника подається 6 LSTM декодувальникам (по одному на можливий маневр). Окремий LSTM кодувальник прогнозує ймовірність кожного маневру
CNN	[31]	Класична CNN із 6 шарами використовується для прогнозування намірів SV
	[37]	Архітектура згортки-зворотньої згортки (англ. convolution-deconvolution)
	[39]	3D згортка застосовується до часового виміру вхідних даних. Після цього, 2D згортки використовуються для аналізу просторових особливостей. Потім 2 гілки шарів згортки використовуються для знаходження ймовірності транспортно засобу та прогнозування його положення на поточному і майбутньому кадрах
	[40]	2 CNN окремо вилучають дані LIDAR та растрової карти. Потім 3 різні мережі застосовуються до конкатенації даних для виявлення транспортних засобів та прогнозування їх траєкторії

Закінчення табл. 1.3

Клас	*Роботи*	*Опис*
GNN	[33]	Згорткові графові мережі (англ. Graph Convolutional Network, GCN) і графові мережі з увагою (англ. Graph Attention Network, GAT)
Інші	[35]	LSTM застосовується до кожної траєкторії. Результат у вигляді BEV зображення подається до CNN. Потім 6 LSTM декодерів (по одному на маневр) застосовуються до результату минулого кроку.
	[38]	CNN вилучає просторові ознаки із вхідного зображення. Ці ознаки оброблюються LSTM з архітектурою кодувальник-декодувальник. Результат подається до мережі зворотної згортки для отримання зображення того ж розміру, що і вхідне.
	[43]	Варіаційний автокодувальник на основі GRU генерує розподіл траєкторій

1.2 Постановка задачі прогнозування траєкторії

У найбільш загальному вигляді, задачу прогнозування траєкторії руху агентів можна сформулювати таким чином:

Нехай $X_{TV_s} = \{x_t^i, x_{t+1}^i, \dots, x_{t+m}^i\}_{i=1}^N$, де x_t^i – стан агенту i в час t, N – кількість TV, а m – довжина вікна прогнозування; O_{EV} – доступні спостереження. Задача полягає в обчисленні умовного розподілу $P(X_{TV_s}|O_{EV})$.

Для зменшення обчислювальних вимог, у багатьох існуючих роботах не враховується взаємозалежність між майбутніми станами агентів. Тоді поведінку кожного TV можна прогнозувати окремо. На кожному кроці обирається один агент T і для нього обчислюється $P(X_{TV}|O_{EV})$ [9].

Конкретні формулювання задачі відрізняються в доступних спостереженнях O_{EV} та X_{TV_s}. Наприклад, у Social GAN [44] використовуються набори даних ETH [47] та UCY [48], які містять лише історію координат для кожного агенту. Тому постановка задачі набуває вигляду:

Маючи траєкторії всіх агентів $X = X_1, \dots, X_n$, спрогнозувати траєкторії $\hat{Y} = \hat{Y}_1, \dots, \hat{Y}_n$ одночасно. $X_i = (x_i^t, y_i^t), t = 1 \dots t_{obs}$ – історія координат агенту i, а $\hat{Y} = (x_i^t, y_i^t), t = t_{obs} + 1, \dots, t_{pred}$ – його майбутня траєкторія.

В QCNet [45] та багатьох інших роботах використовується один із популярних наборів даних Argoverse/Argoverse 2 [46], який, окрім координат, також містить кут повороту та швидкість кожного агенту. Також він містить семантичну мапу навколишнього середовища. Візуалізації деяких прикладів із Argoverse 2 зображено на рис. 1.2.

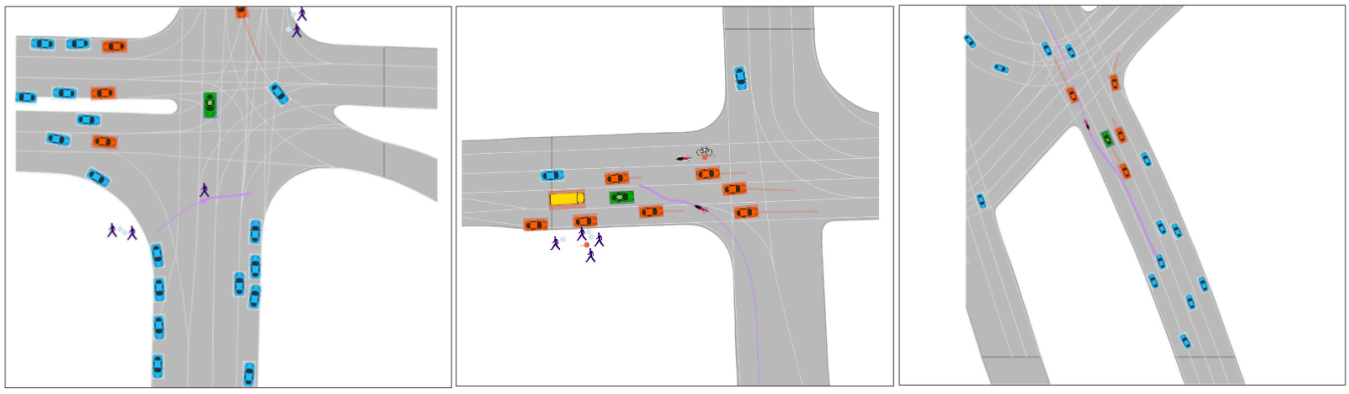

Рисунок 1.2 – Приклади сцен із Argoverse 2.

Деякі роботи [49] напряму використовують записи відеокамер без додаткових сенсорів як у Intersection Drone Dataset [50]. Приклад із Intersection Drone Dataset зображено на рис. 1.3.

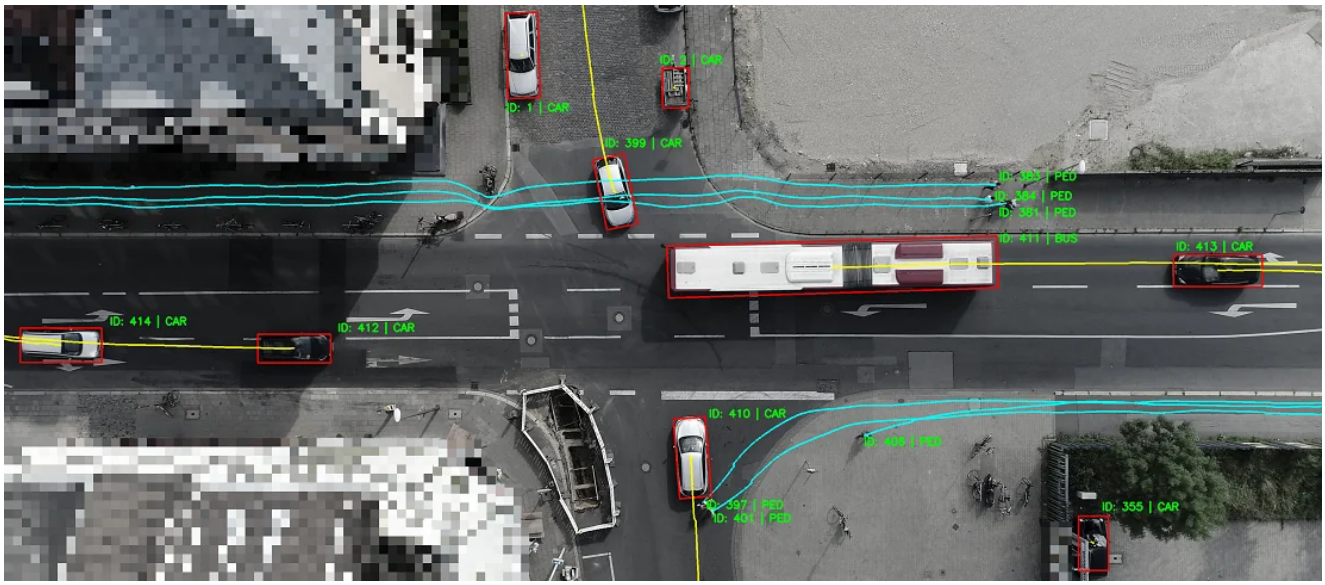

Рисунок 1.3 – Приклад сцени із Intersection Drone Dataset.

1.3 Постановка задачі структурно-параметричного синтезу

Нехай $D = X_{TV_s} \cup O_{EV}$ – тренувальний набір, а $N = \{n_1, \dots, n_m\}$ – множина можливих топологій нейронних мереж.

Метою роботи є синтезувати оптимальну нейронну мережу на основі навчальної вибірки D, яка б забезпечувала ефективне та високоточне розв'язання прикладної задачі прогнозування траєкторії, поставленої вище. Векторний критерій оптимальності можна визначити як:

$$I = \{I_1(N, D), I_2(N), I_3(N)\} \rightarrow min,$$

де $I_1(N,D) = E_{valid}(N,D)$ – похибка узагальнення (тобто помилка на тестовій вибірці), $I_2(N)$ – час прогнозування, $I_3(N)$ – складність нейронної мережі.

Тобто необхідно визначити оптимальну структуру і топологію нейронної мережі, яка б мінімізувала помилку і час на прогнозування, оскільки це критично важливо для застосування у режимі реального часу на дорогах. Деталі про способи оцінювання якості прогнозованих траєкторій наведені у розділі 4.2.

Висновки до розділу 1

У розділі виконано аналіз актуальності та проведено дослідження предметної області в галузі самокерованих автомобілів загалом та прогнозування траєкторії зокрема.

Огляд існуючої літератури від зародження ідеї самокерованих автомобілів до сучасності проведено в п. 1.1. Наведено декілька видів класифікації робіт – за складністю припущень (на фізичні, моделі з урахування маневрів і з урахуванням взаємодії) та за типом вхідних, вихідних представлень і архітектурою моделі.

Більшість сучасних робіт стосуються моделей із урахування взаємодії, проте вони мають різноманітні архітектури та типи представлень. Недоліком існуючих моделей є те, що вони або точні проте занадто складні обчислювально для використання в режимі реального часу, або ефективні, але недостатньо точні.

В останніх пунктах наведено формальну постановку задачі прогнозування траєкторії та постановку задачі структурно-параметричного синтезу.

РОЗДІЛ 2 АНАЛІЗ ОСНОВНИХ СКЛАДОВИХ МОДЕЛЕЙ ПРОГНОЗУВАННЯ ТРАЄКТОРІЇ

В цьому розділі розглядаються теоретичні відомості про основні типи нейронних мереж, які можуть застосовуватись для розв'язання задачі прогнозування траєкторії учасників дорожнього руху.

2.1 Згорткові нейронні мережі

Згорткові нейронні мережі (англ. Convolutional Neural Network) або CNN – клас нейронних мереж, покликаний імітувати особливості зорової області головного мозку [64]. Першу сучасну згорткову мережу було застосовано для задачі розпізнавання поштових індексів США ще у 1989 році ЛеКуном та ін. [65]. Проте справжньої популярності згорткові мережі досягли значно пізніше – у 2010-х роках, коли збільшення обчислювальних можливостей комп’ютерів та підтримка навчання з використання відеокарти дозволило створювати глибокі згорткові мережі та навчати їх на великих зображеннях.

CNN є важливою складовою багатьох сучасних систем машинного навчання, зокрема в області зображень та відео. Вони також успішно застосовуються для обробки послідовностей – часових рядів та природної мови.

Типова архітектура згорткової нейронної мережі зображена на рис. 2.1. Вона складається із шарів згортки та агрегування або об’єднання (англ. pooling), що чергуються між собою.

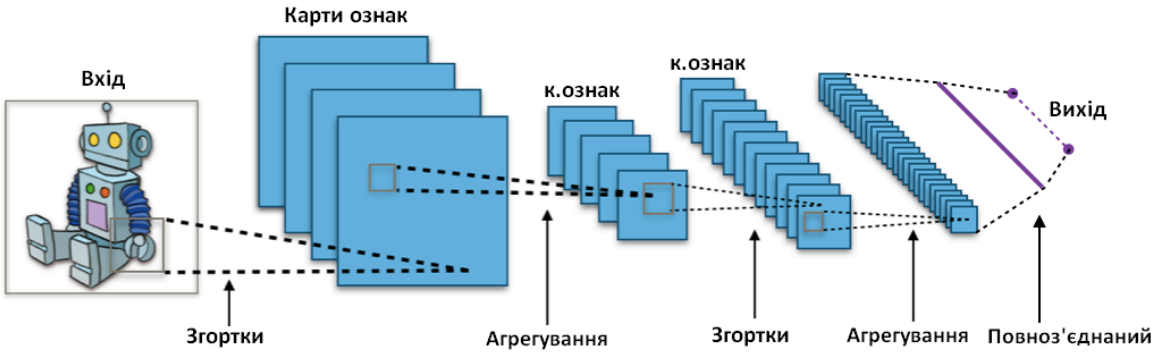

Рисунок 2.1 – Типова згорткова нейронна мережа[1]

2.1.1 Шар згортки

Основним будівельним блоком згорткової мережі є шар згортки, який складається із ядер (англ. kernel) або фільтрів (англ. filter). Ядра представляють собою матриці (зазвичай квадратні) із вагами. Типові розміри ядер згортки – 3×3, 5×5, 1×1. У згортковому шарі використовується ковзне вікно, де фільтр «ковзає» по вхідному тензору, виконуючи скалярний добуток:

$$O(i,j) = \sum_{m,n} I(i \times stride + m, j \times stride + n) * K(m,n),$$

де $I(i,j)$ – вхідна матриця, $stride$ – крок ковзання, K – ядро згортки

Приклад роботи ядра згортки зображено на рис. 2.2.

[1] Джерело зображення: https://upload.wikimedia.org/wikipedia/commons/a/ae/Typical_cnn_uk.png

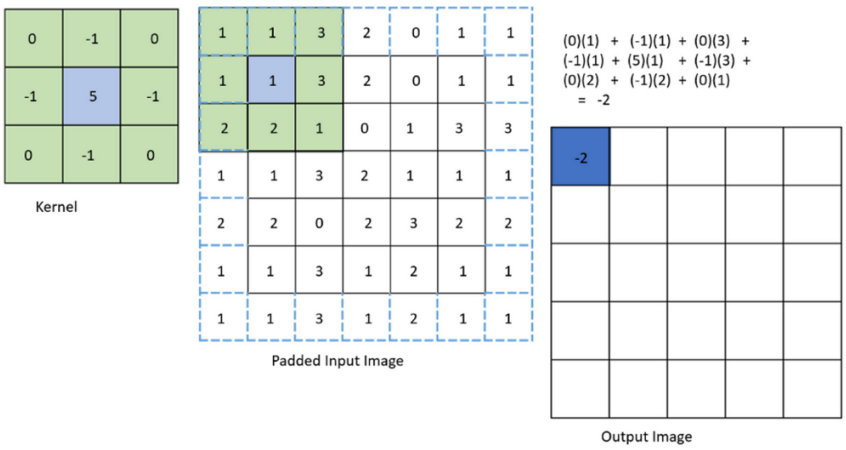

Рисунок 2.2 – Приклад роботи фільтру згортки[2]

Фільтр рухається вправо із кроком $stride$, поки не обробить всю ширину зображення. Після цього фільтр переміщується до початку (зліва) вхідного зображення і повторює процес, поки все зображення не буде оброблене.

У випадку вхідних даних із декількома каналами (наприклад RGB зображення), ядро має глибину рівну глибині вхідного тензору. На кожному рівні глибини виконується скалярний добуток і результати додаються. На виході маємо стиснутий результат глибини один. Приклад роботи фільтру для багатоканального входу зображено на рис. 2.3.

[2] Джерело зображення: https://medium.com/v2 /format:webp/1*cpOsBrBoV_XIPIkPtuiNvQ.png

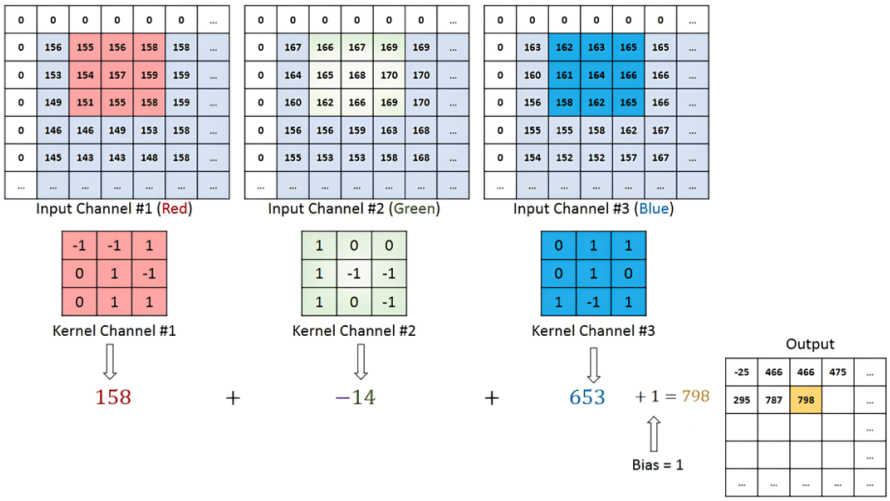

Рисунок 2.3 – Приклад роботи фільтру згортки на багатоканальному зображені[3]

Ідея згортки полягає в вилученні важливих ознак (таких як лінії, ребра) із вхідного зображення. Зазвичай, згорткові мережі складаються з багатьох шарів згорток. Тоді перший шар відповідає за виявлення простих ознак – ліній та кольорів, а кожен наступний за більш і більш абстрактні складні ознаки. Таким чином, досягається цілісний аналіз зображення, аналогічно до біологічних процесів.

Використання згорткових мереж дозволяє значно зменшити кількість параметрів, оскільки шари згортки використовують спільні ваги для різних областей вхідного зображення. Наприклад для обробки зображення 100×100 звичайній повнозв'язній мережі необхідно використати 10000 вагів у кожному нейроні. Натомість, використання згортки 5×5 дає змогу використати лише 25 нейрони, незалежно від розміру вхідного зображення. Іншою перевагою згорткових мереж є їхня просторова інваріантність, що дозволяє їм впізнавати ознаки, незалежно від їх положення.

---

[3] Джерело зображення: https:// medium.com/v2/format:webp/1*Ch23Xi-vJt_hDSBGDAx5PA.gif

2.1.2 Шар об’єднання

Операція об’єднання є іншою ключовою складовою більшості згорткових мереж. Вона відіграє важливу роль у зменшенні розмірності входу, що дозволяє зменшити обчислювальну складність та запобігати явищу перенавчання (англ. overfitting) завдяки вилученню домінантних ознак.

Ідея полягає в тому, що точне місцезнаходження ознаки не так важливе, як його приблизне положення відносно інших ознак. Зазвичай використовується максимальне (англ. max pooling) та середнє об’єднання (англ. average pooling), серед яких перевагу частіше віддають першому. Max pooling передбачає вибір максимального значення з локальної області вхідної карти ознак. При average pooling обчислюється середнє значення елементів локальної області. Приклади роботи обох варіантів операцій об’єднання зображено на рис. 2.4.

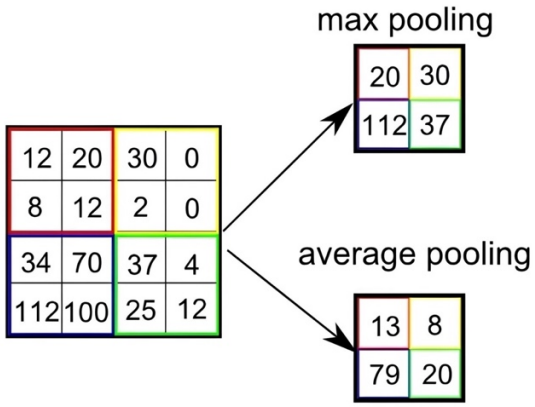

Рисунок 2.4 – Приклад роботи max та average pooling[4]

[4] Джерело зображення: https://medium.com/v2/format:webp/1*mOz3uklxC_VgQtA1p0UFtg.png

Середнє об’єднання згладжує зображення, тоді як max pooling виділяє найбільш важливі ознаки кожного регіону. На практиці, більшість згорткових мереж обирає max pooling, оскільки він демонструє кращі результати.

2.1.3 Згорткові нейронні мережі для обробки послідовностей

Згорткові мережі досягають високої точності у задачах комп’ютерного зору завдяки своєму вмінню ефективно виділяти локальні ознаки із вхідних даних. Ці ж властивості CNN можуть дозволити їм досягати успіхів у обробці послідовностей. В таких задачах час можна розглядати як просторовий вимір, аналогічно до висоти або ширини 2D зображення.

Для цього використовується одновимірна згортка або 1D convolution. Аналогічно до 2D згортки, в одновимірній згортці фільтр (ядро) ковзає вздовж вхідної послідовності, виявляючи локальні закономірності і створюючи послідовність карт ознак. Приклад роботи 1D згортки зображено на рис. 2.5.

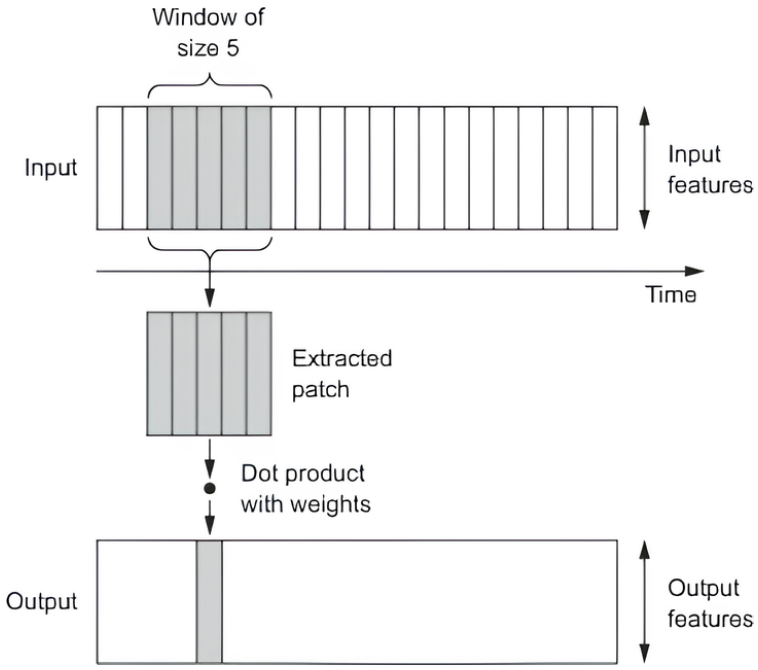

Рисунок 2.5 – Приклад роботи 1D згортки [67]

1D згортки демонструють високу точність, порівнянну із рекурентними нейронними мережами у багатьох задачах. Також до переваг одновимірної згортки відноситься значно вища обчислювальна ефективність та менша кількість параметрів. До недоліків можна віднести неможливість аналізувати довгострокові залежності та точне положення елемента послідовності.

2.2 Рекурентні нейронні мережі

Моделювання послідовностей (таких як природна мова, часові ряди, аудіо тощо) має велике значення у багатьох сферах. Один із найпопулярніших підходів до розв'язання цієї задачі полягає у використанні рекурентних нейронних мереж (англ. recurrent neural network) або RNN.

Головною особливістю цих моделей є використання зворотних зв’язків, що дозволяють враховувати часові особливості даних і, по суті, моделювати пам'ять.

Схема RNN розгорнута в часі зображена на рис. 2.6.

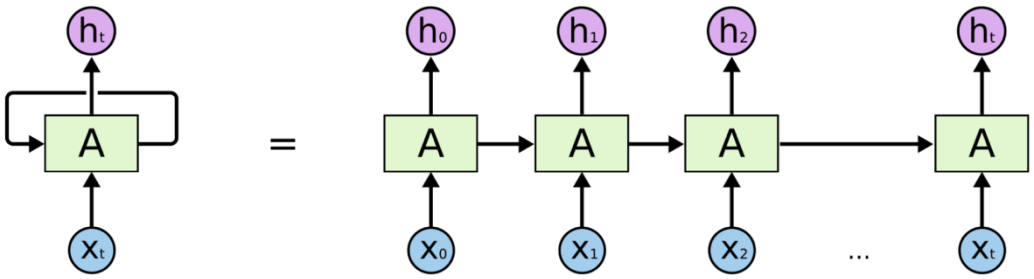

Рисунок 2.6 – RNN розгорнута в часі [24]

RNN опрацьовують послідовність крок за кроком, зберігаючи внутрішній прихований стан (англ. hidden state). Цей прихований стан оновлюється на кожному кроці і впливає на поточний прогноз. Математично це можна виразити таким чином:

1. Ініціалізація прихованих станів.
2. На часовому кроці t, на вхід подається вектор $x(t)$. RNN обчислює новий прихований стан.

$$h(t) = f(W_x x(t) + W_h h(t-1) + b_h),$$

де f – функція активації (зазвичай сигмоїда або гіперболічний тангенс), W_x, W_h, b_h – параметри, яким мережа навчається.

3. Генерується вихід

$$y(t) = g\big(W_y h(t) + b_y\big),$$

де g – функція активації, а W_y, b_y – параметри моделі.

Однією з основних проблем простих RNN моделей є обробка довгострокових залежностей у даних. Хоча параметри, які б дозволяли це зробити теоретично існують, на практиці навчити RNN цим параметрам майже неможливо через проблему зникнення градієнтів. Тому популярності набули більш складні архітектури рекурентних мереж, такі як довга короткострокова пам'ять (англ. long short-term memory) LSTM та вентильний рекурентний вузол (англ. gated recurrent unit) GRU.

2.2.1 Довга короткострокова пам'ять

LSTM – рекурентна нейронна мережа, яка створена для боротьби з проблемою зникання градієнтів у традиційних RNN [51]. Вона спрямована на забезпечення короткочасної пам’яті, яка б не зникала протягом тисяч часових кроків – звідси і назва.

Головна ідея в LSTM – створити додатковий модуль в мережі, який би вивчав коли запам’ятовувати, а коли забувати відповідну інформацію. Зазвичай, LSTM втілюють у блоках, кожен з яких містить вентилі (англ. gates), які керують потоком даних – вхідний вентиль (англ. input gate), вихідний вентиль (англ. output gate) та вентиль забування (англ. forget gate). Архітектуру типового блоку LSTM зображено на рис. 2.7.

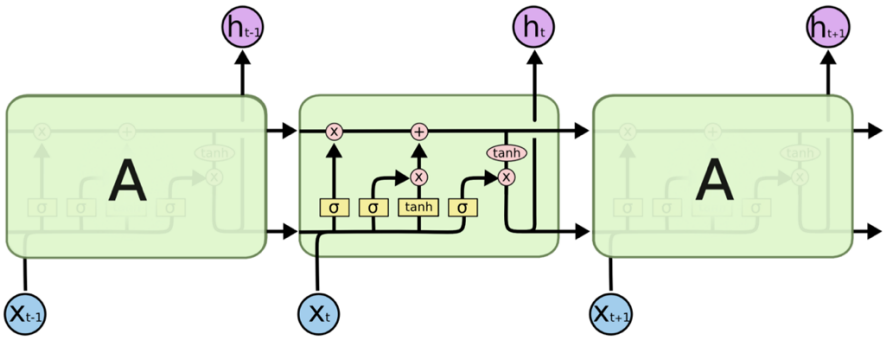

Рисунок 2.7 – Типовий блок LSTM [51]

Рівняння прямого поширення сигналу в блоці LSTM мають такий вигляд:

$$f_t = \sigma_t\big(W_f x_t + U_f h_{t-1} + b_f\big),$$
$$i_t = \sigma_t(W_i x_t + U_i h_{t-1} + b_i),$$
$$o_t = \sigma_t(W_o x_t + U_o h_{t-1} + b_o),$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c),$$
$$h_t = o_t \circ \sigma_h(c_t),$$

де x_t – вектор входу, h_t – вектор виходу, c_t – вектор стану комірки, W, U, b – ваги, f_t, i_t, o_t – forget gate, input gate і output gate відповідно, σ_g- сигмоїдна функція, а σ_c, σ_h – гіперболічний тангенс.

2.2.2 Вентильний рекурентний вузол

Вентильний рекурентний вузол – варіант рекурентної нейронної мережі, подібно до LSTM створений для розв'язання проблеми зникнення градієнтів [52]. Він має схожий підхід – використання механізму вентилей

для запам’ятовування або забування певних ознак. Проте на відміну від LSTM з його трьома вентилями, він має лише ці 2, що означає меншу кількість параметрів, ніж в LSTM.

Архітектуру блоку GRU зображено на рис. 2.8.

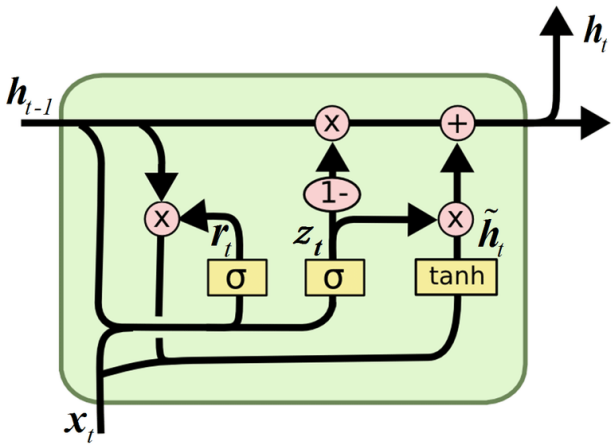

Рисунок 2.8 – Архітектура блока GRU [52]

Рівняння прямого поширення сигналу в блоці GRU мають такий вигляд:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z),$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r),$$
$$\tilde{h}_t = \phi(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h),$$
$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h}_t,$$

де x_t – вектор входу, h_t – вектор виходу, W, U, b – ваги, h_t – вектор виходу, z_t – вентиль оновлення (англ. update gate), r_t – вентиль скидання (англ. reset vector), а σ, ϕ – функції активації, в оригіналі – сигмоїда та гіперболічний тангенс відповідно.

Оскільки GRU має менше параметрів, ніж LSTM, GRU може бути корисним варіантом для ситуацій з обмеженими обчислювальними можливостями. Проте недоліком є те, що у складних задачах LSTM показує кращі результати.

2.3 Механізм уваги

Механізм уваги (англ. Attention) – механізм, вперше запропонований Багданау та ін. [53] у 2014 р. в контексті обробки природньої мови, а саме – машинного перекладу. На сьогоднішній день, різни види механізмів уваги успішно застосовуються у багатьох галузях глибокого навчання та відіграють ключову роль в сучасних моделях обробки послідовностей даних, таких як текст і зображення.

Ідея механізму полягає в імітуванні когнітивної уваги людей – наданні більшої ваги певним факторам. Це дозволяє розв'язати проблему обробки довгих послідовностей, притаманну RNN моделям, які намагаються закодувати інформацію в один вектор фіксованої довжини, втрачаючи важливий контекст. В RNN моделях без уваги, слова ближче до кінця послідовності мали більший вплив, тоді як використання механізму attention дозволяє вільно звертати більше уваги на будь-який елемент послідовності.

Перші моделі з увагою додавали її поверх RNN моделей, проте після появи Трансформерів (англ. Transformer) [54] у 2017, сучасні підходи відмовилися від RNN та використовують лише увагу, що дозволяє значно пришвидшити обчислення завдяки можливості паралельних обчислень у механізмах уваги.

2.3.1 Адитивна увага

Оригінальний механізм уваги, описаний у [53]. Представляє собою розширення архітектури кодувальник-декодувальник для спільного вирівнювання та перекладу (англ. align and translate). Щоразу, коли модель генерує слово в перекладі, вона шукає набір позицій у вхідному реченні, де знаходиться найбільш релевантна інформація. Потім модель прогнозує слово на основі векторів контексту, пов’язаних з цими вихідними позиціями та попереднього згенерованими словами перекладу. Це усуває проблему стиснення всього вхідного речення у вектор фіксованої довжини. Модель з адитивною увагою, що генерує y_t вихідної послідовності із вхідним реченням $(x_1, x_2, \dots, x_T)$ зображено на рис. 2.9.

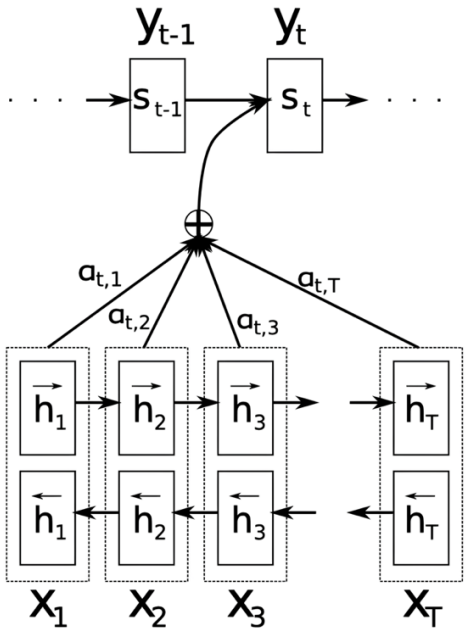

Рисунок 2.9 – Адитивна увага [53]

Нижче наведено алгоритм адитивної уваги.

1. Показники вирівнювання (англ. alignment scores). Модель вирівнювання використовує приховані стани кодувальника h_i, попереднього виходу декодувальника s_{t-1} для обчислення оцінки $e_{t,i}$, яка показує наскільки релевантні елементи вхідної послідовності до поточного виходу на позиції t. Модель представлена функцією $a(s, h)$, яку можна реалізувати за допомогою нейронної мережі прямого поширення (англ. feedforward neural network).

$$e_{t,i} = a(s_{t-1}, h_i).$$

2. Ваги. Обчислюються ваги $\alpha_{t,i}$ шляхом застосування softmax до попередньо обчислених оцінок вирівнювання:

$$\alpha_{t,i} = softmax(e_{t,i}),$$

де $softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$.

3. Вектор контексту. Унікальний вектор контексту c_t подається на вхід декодувальника на кожному часовому кроці. Він обчислюється як зважена сума всіх прихованих станів кодувальника.

$$c_t = \sum_i \alpha_{t,i} h_i.$$

2.3.2 Мультиплікативна увага

Інший тип уваги був запропонований Луонгом та ін. у 2015 [55]. Як і адитивна увага, цей механізм застосовувався у складі RNN моделі. На відміну від уваги Багданау, Луонг використовує декілька типів вирівнювання:

- загальна увага (англ. general attention):

$$score = softmax(h_t^T W_a h_s);$$

- конкатенаційна увага (англ. concatenation attention):

$$score = softmax(v_a^T \tanh(W_a[h_t; h_s]));$$

- скалярна увага (англ. dot attention):

$$score = softmax(h_t^T h_s),$$

де h_t – поточний прихований стан декодувальника, h_s – приховані стани кодувальника, W_a, v_a – параметри моделі, а [;] – операція конкатенації.

Ілюстрацію роботи уваги Луонга можна знайти на рис. 2.10.

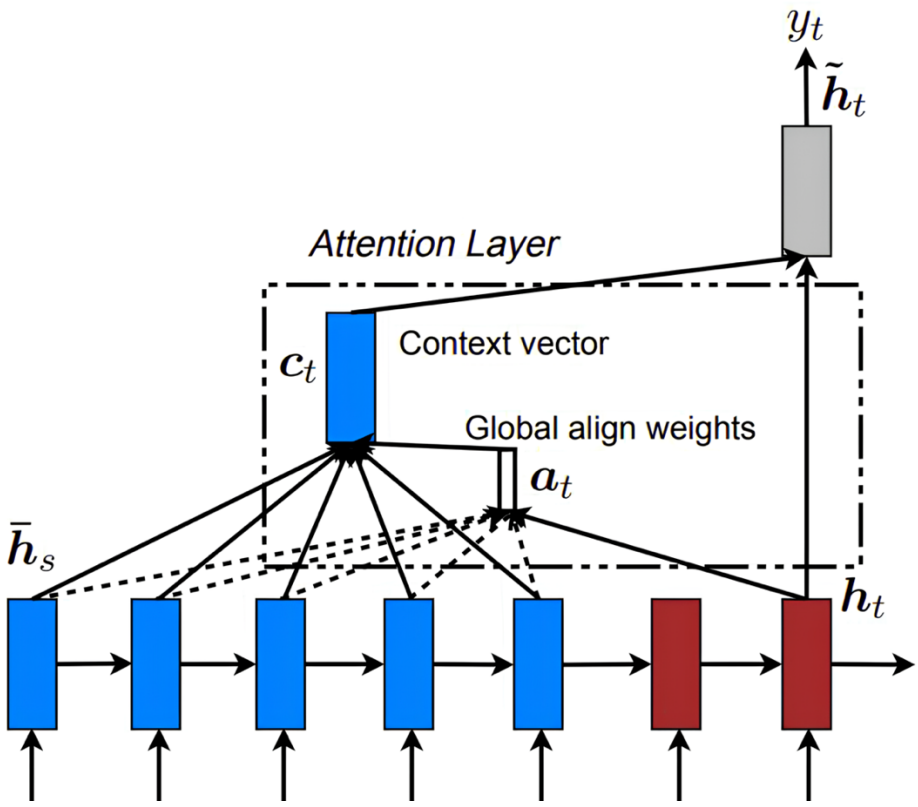

Рисунок 2.10 – Мультиплікативна увага [55]

2.3.3 Трансформер

RNN довгий час були найкращими моделями для машинного перекладу та інших задач обробки послідовностей. Але послідовна природа RNN перешкоджає розпаралелюванню навчання, що стає критичним при великій довжині послідовності.

Механізм уваги дозволяє моделювати залежності без урахування відстаней у вхідних та вихідних послідовностях. У 2017 Васвані та ін. запропонували архітектуру Трансформер (англ. Transformer), в якій відмовились від використання рекурентності та, натомість, повністю

покладаються на механізм уваги для виявлення глобальних залежностей між входом та виходом [54].

Схему запропонованої Трансформер моделі зображено на рис. 2.11.

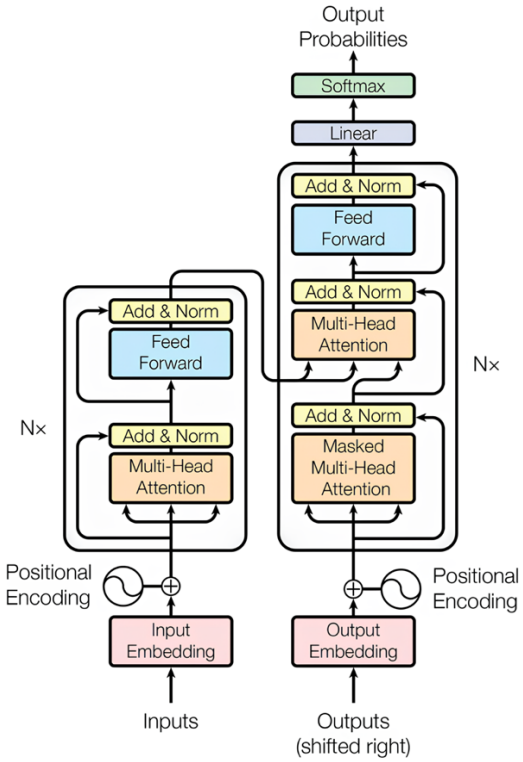

Рисунок 2.11 – архітектура Трансформер [54]

Основою моделі є механізм масштабованої скаляорнодобуткової уваги (англ. scaled dot-product attention). Для кожного вузла уваги модель навчається трьом матрицям – ваги запиту (англ. query weights) W_Q, ваги ключа (англ. key weights) W_K та ваги значення (англ. value weights) W_V. Для кожного елементу послідовності, його вкладення (англ. embedding) x_i множиться на кожну з цих матриць. Отримаємо:

- запит – $q_i = x_i W_Q$;
- ключ – $k_i = x_i W_K$;
- значення – $v_i = x_i W_V$.

Нехай Q, K, V – матриці, створені векторами q_i, k_i та v_i відповідно. Тоді scaled-dot attention можна обчислити за формулою:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V,$$

де d_k – розмірність вектору ключа.

Такий підхід є майже ідентичним мультиплікативній увазі, за винятком коефіцієнта масштабування $\frac{1}{\sqrt{d_k}}$. Цей коефіцієнт використовується для стабілізації градієнтів під час тренування, щоб запобігти потраплянню функції $softmax$ в область насичення з малими градієнтами.

Замість того, щоб використовувати одну «голову уваги» (англ. attention head), більш корисно проєктувати запити, ключі та значення h разів із різними проєкціями для кожної голови уваги. Результати конкатенуються та знову лінійно проєктуються в фінальний вихід «багатоголової уваги» (англ. multi-head attention). Це дозволяє звертати увагу на різні позиції виходячи із різних визначень «релевантності».

Scaled dot-product attention і multi-head attention зображено на рис. 2.12.

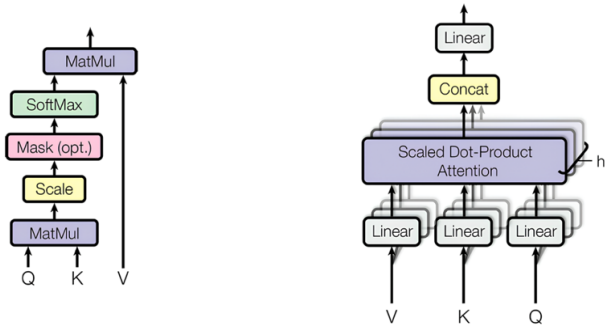

Рисунок 2.12 – Scaled dot-product attention (зліва) і multi-head attention (справа) [54]

В залежності від входів до механізму уваги виділяють самоувагу (англ. self-attention) та перехресну увагу (англ. cross-attention). В self-attention, запити, ключі і значення отримують з однієї послідовності, тоді як cross-attention дозволяє змішувати 2 послідовності, які можуть мати різну природу (наприклад текст і зображення). Одна із послідовностей використовується для введення запиту, а інша – для ключа та значення. Існує й альтернативний варіант – використання запиту і значення із однієї послідовності, а ключа – із іншої. Механізм cross-attention зображено на рис. 2.13.

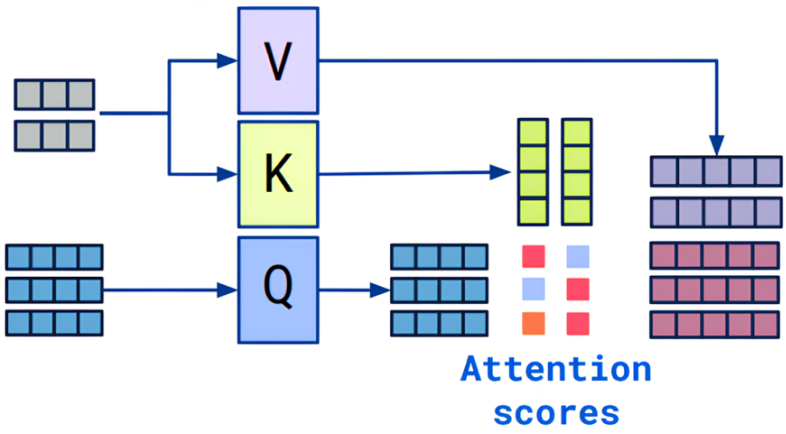

Рисунок 2.13 – Cross-attention[5]

В задачах прогнозування траєкторій послідовності зазвичай багатовимірні – вони мають просторовий та часовий виміри. Моделі Transformer погано масштабуються для багатовимірних послідовностей, оскільки вони мають обчислювальну складність $O(S_m^2 \times T^2)$, де S_m – довжина просторового виміру, а T – часового. Одним із способів боротьби із цією проблемою є використання факторизованої уваги (англ. factorized

[5]Джерело зображення: https://vaclavkosar.com/images/cross-attention-detail-perceiver-io.png

attention) [56]. При цьому підході, механізм уваги застосовується окремо до кожного з вимірів. Два варіанти factorized attention для такого випадку – послідовний (англ. sequential) та з чергуванням зображено на рис. 2.14.

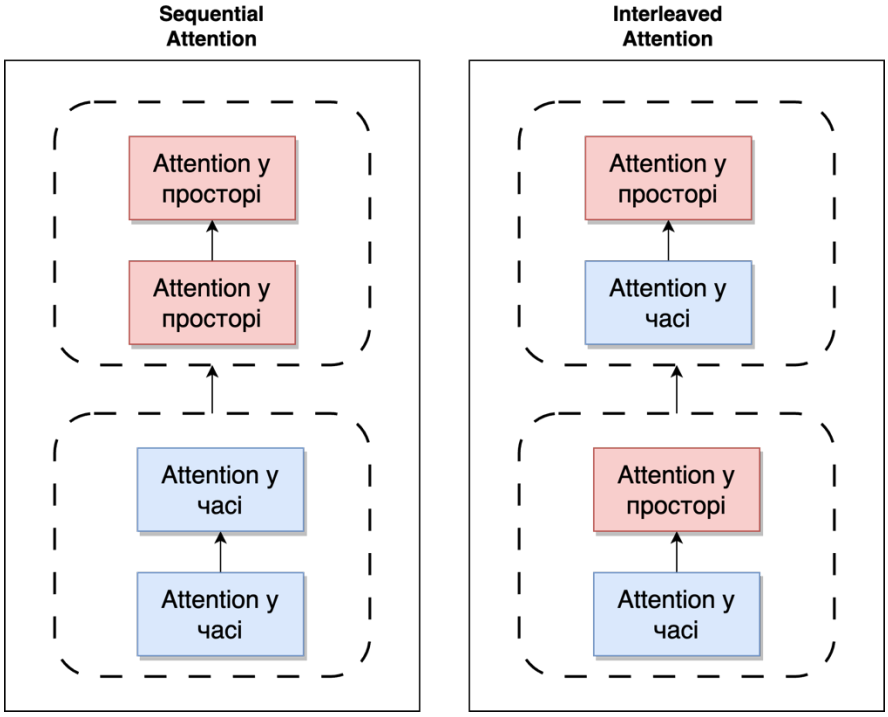

Рисунок 2.14 – Sequential та interleaved factorized attention

2.4 Графові нейронні мережі

Важливою проблемою при прогнозуванні траєкторії є моделювання взаємодії між агентами. Одним із природніх підходів до розв'язання цієї проблеми є представлення задачі у вигляді графу.

Граф – математична структура, що описує сукупність об’єктів із зв’язками між ними. Більш строго, граф G – це пара $G =< V, E >$, де V

– множина вершин або вузлів (англ. vertex), а E – множина зв'язків між ними, які називаються ребрами (англ. edges).

Приклад графу взаємодії агентів дорожнього руху зображено на рис. 2.15.

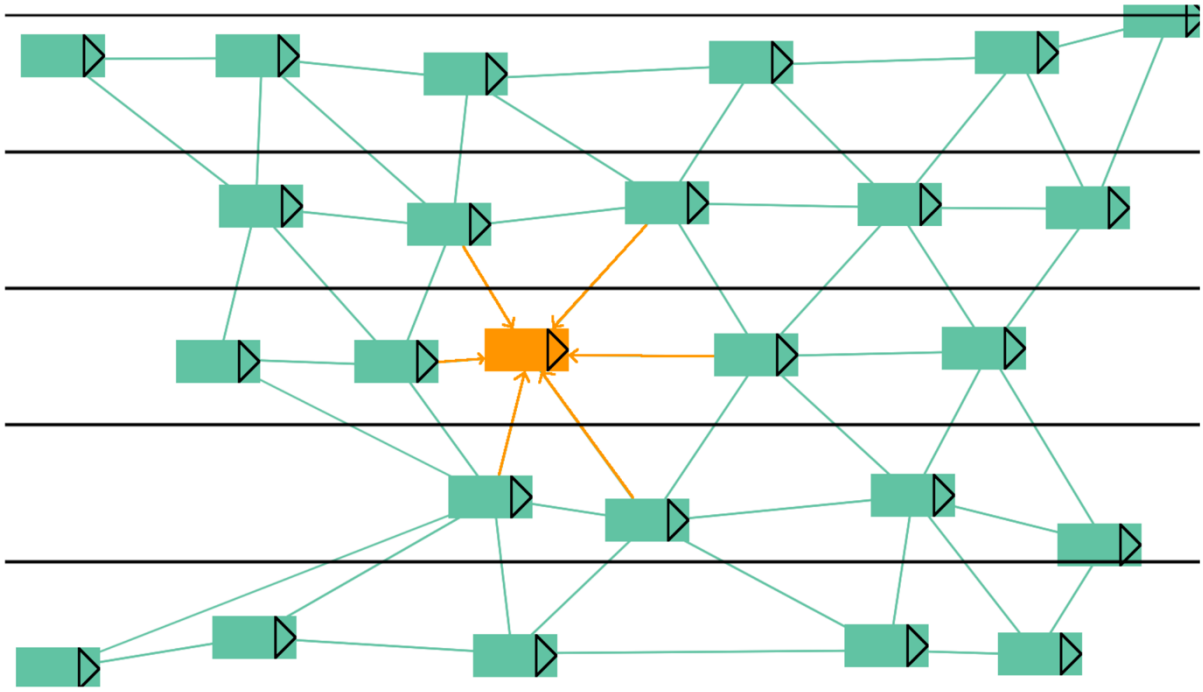

Рисунок 2.15 – Граф взаємодії агентів [63]

Унікальна неевклідова структура графів вимагає особливого підходу до побудови моделей машинного навчання. Виклики пов'язанні з аналізом графів охоплюють нижчеперелічене.

Нерегулярна структура. Стандартні нейронні мережі розраховані на роботу з матрицями фіксованого розміру, тоді як графи мають складну динамічну структуру.

Інваріантність перестановок. Графи інваріантні до перестановок і для кожного графа існує багато ізоморфних йому. Стандартним моделям може бути складно враховувати цю особливість.

Інформація про зв'язки. Зв'язки між вершинами графа містять важливу інформацію. Традиційним моделям зазвичай бракує механізмів для безпосереднього відображення цих зв'язків, які мають вирішальне значення для розуміння структури графа.

Гетерогенні графи. Деякі графи можуть мати вершини та ребра різних типів.

Масштабованість. Графи можуть мати надзвичайно велику кількість вершин або ребер. Для обробки великих графів необхідні спеціалізовані ефективні алгоритми.

Успіхи методів глибокого навчання у інших сферах штучного інтелекту (таких як комп'ютерний зір та обробка природних мов) спонукали дослідників до пошуку підходів, які б дозволили використати нейронні мережі для розв'язання цих проблем.

Ключовим елементом дизайну графових нейронних мереж (англ. graph neural network) або GNN є використання передачі повідомлень (англ. message passing), коли вершини графа ітеративно оновлюють свої представлення, обмінюючись інформацією із своїми сусідами. З моменту появи GNN було запропоновано багато архітектур, які відрізняються способами передачі повідомлень.

2.4.1 Передача повідомлень

Шари передачі повідомлень – еквіваріантні до перестановки шари, які відображають граф в оновлене представлення цього ж графу. У такому шарі, вузли графу оновлюють свої представлення агрегуючи повідомлення, отримані від своїх безпосередніх сусідів. Більш формально, нехай $\mathcal{G} =< \mathcal{V}, \mathcal{E} >$ – граф, $\mathcal{N}_u = \{v \in \mathcal{V} | (v, u) \in \mathcal{E}\}$ – околиця вузла u, x_u – атрибути

вершини u, а e_{uv} – атрибути ребра $(u, v) \in \mathcal{E}$. Тоді шар передачі повідомлень можна визначити як функцію h_u:

$$h_u = \phi\left(x_u, \bigoplus_{v \in \mathcal{N}_u} \psi(x_u, x_v, e_{uv})\right),$$

де ψ – диференційовна функція повідомлення, ϕ – диференційовна функція оновлення, а $\oplus$ – оператор агрегації, інваріантний до перестановок (наприклад сума, середнє або максимум). Приклад оновлення представлення в message passing layer зображено на рис. 2.16.

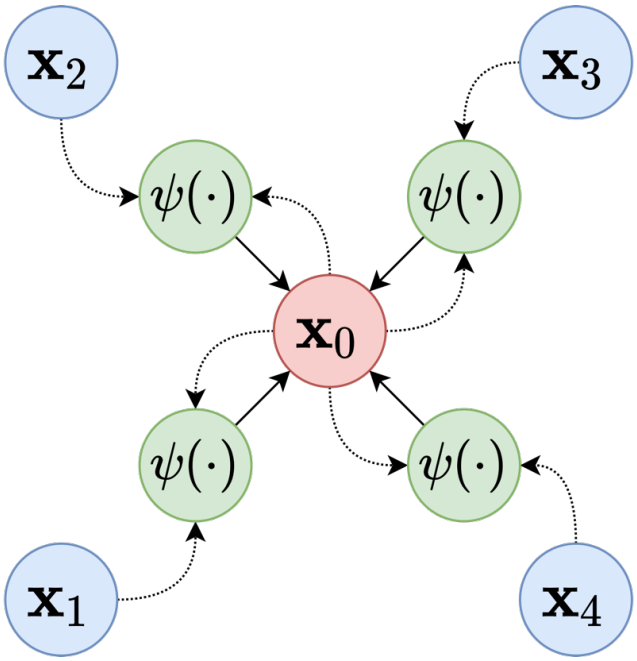

Рисунок 2.16 – Оновлення представлення

Для того, щоб вузли графу могли взаємодіяти з вузлами за n «стрибків» від нього, необхідно послідовно використати n шарів передачі повідомлень. Взагалі, щоб гарантувати, що кожна вершина отримує інформацію від будь-якої іншої вершини, необхідно покласти n рівним

діаметру графа. Проте на практиці, занадто велика кількість шарів призводить до проблеми згладжування, коли представлення всіх вершин стають нерозрізненними.

Ще однією проблемою притаманною передачі повідомлень, і, як наслідок, більшості графових нейронних мереж є існування деяких неізморофних графів, які GNN не може розрізняти. Одним із шляхів до подолання цієї проблеми є модифікація вхідного графу [57]. Два різних графи, які GNN вважає ізоморфними зображено на рис. 2.17.

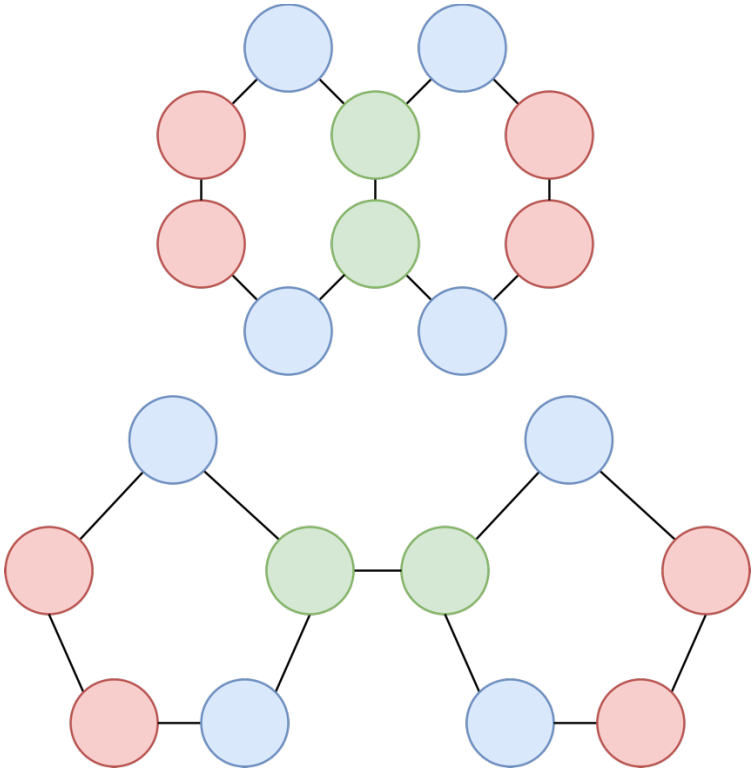

Рисунок 2.17 – Неізоморфні графи, які не розрізняються GNN

2.4.2 Графові згорткові нейронні мережі

Популярний різновид шарів передачі повідомлень, шар графової згорткової мережі (англ. graph convolutional network) або GCN був представлений у 2016 Кіпфом та Велінгом [58], як узагальнення ідей згорткових нейронних мереж до даних із структурою графів.

Математично шар можна визначити як:

$$x_i^{(k)} = \sum_{j \in \mathcal{N}_i \cup \{i\}} \frac{1}{\sqrt{\deg(i)} * \sqrt{\deg(j)}} \left(W^T x_j^{(k-1)} \right) + b$$

де W, b – ваги, а $\deg(i)$ – степінь вершини i.

Цю формулу можна розбити на кроки:

1. Додаємо петлі до графу.
2. Лінійно перетворюємо матрицю атрибутів вузлів.
3. Обчислюємо коефіцієнти нормалізації.
4. Нормалізуємо атрибути вузлів у ψ.
5. Знаходимо суму ознак трансформованих ознак сусідніх вершин – агрегація
6. Додаємо вектор зсуву b.

На практиці застосовується більш зручний для обчислення вираз у матричній формі:

$$H^{(k+1)} = \sigma \left(\widetilde{D}^{-\frac{1}{2}} \tilde{A} \widetilde{D}^{-\frac{1}{2}} H^{(k)} W^{(k)} \right)$$

де $H^{(k)}$ – матриця активації на k-му рівні, $H^{(0)} = X$, $\tilde{A} = A + I_N$ – матриця суміжності із доданими петлями, $\widetilde{D} = \sum_j \tilde{A}_{ij}$ – матриця степеней вершин, а σ – функція активації (наприклад ReLU).

Висновки до розділу 2

У розділі викладено основні теоретичні відомості необхідні для побудови моделі прогнозування траєкторії.

Згорткові нейронні мережі зазвичай використовуються на растеризованих зображеннях сцени або для аналізу послідовностей станів (у випадку одновимірних згорток). Згорткові мережі добре моделюють просторові зв’язки та дозволяють створювати швидкі, обчислювально ефективні моделі. Проте їх недоліком є проблеми у моделюванні довгострокових залежностей.

Рекурентні нейронні мережі моделюють послідовності станів. LSTM та GRU вміють моделювати довгострокові залежності в часі, проте мають проблеми із моделюванням просторових залежностей, які є важливими в контексті створення моделей прогнозування траєкторії із урахуванням взаємодії. Іншим помітним недоліком є послідовність обчислень в рекурентних мережах, що робить ці моделі повільними.

Трансформер – нова архітектура, заснована на механізмі уваги, який імітує принципи когнітивної уваги. Добре моделює як часові, так і просторові закономірності та поєднує послідовності різної природи, що важливо в контексті прогнозування траєкторії. На відміну від рекурентних нейронних мереж, легко паралелізуються. Завдяки всім своїм перевагам, набули значної популярності в різних галузях машинного навчання.

Графові нейронні мережі – спеціалізовані нейронні мережі, створенні для моделювання графів. Оскільки дорожня сцена добре описується графом, то спроби застосування графових нейронних мереж до задачі прогнозування траєкторії учасників дорожнього руху є природніми. Графові нейронні мережі схожі на згорткові та використовують порівнянно невелику

кількість параметрів, що сприяє створенню обчислювально ефективних моделей.

РОЗДІЛ 3 АРХІТЕКТУРА МОДЕЛЕЙ ПРОГНОЗУВАННЯ ТРАЄКТОРІЇ

3.1 Модель Transformer

У цьому розділі розглядаються деталі архітектури запропонованої моделі Transformer для розв’язання задачі прогнозування траєкторії. Вибір цього підходу обумовлений визнаною ефективністю Transformer моделей у моделюванні довгострокових залежностей, врахуванні контексту та в поєднанні вхідних послідовностей різних модальностей, що важливо для розв’язуваної задачі. Широка популярність моделі в інших галузях машинного навчання, таких як обробка природної мови та зображень, додатково підкреслює її потенціал.

3.1.1 Вхідні дані

Важливим етапом розробки моделі є вибір системи координат для представлення її входів та виходів. Існує 2 поширених варіанти, кожний із яких допускає певні компроміси та має своїм переваги та недоліки. Агентоцентричні (англ. agent-centric) моделі представляють вхідні дані та внутрішній стан у системі координат, центром якої є агент. Координати всіх елементів дороги (смуги, переходи тощо) і стани інших агентів описуються відносно положення та орієнтації агента. Це досягається шляхом перетворення координат таким чином, щоб центр его-агента став початком координат, і поворотом, щоб напрямок руху агента був одиничним вектором $(1, 0)^T$. Такий підхід можна вважати формою попередньої обробки даних, яка дозволяє моделям бути інваріантними до паралельних перенесень та

поворотів. На практиці, це дозволяє моделям мати найкращі показники в публічних рейтингах. Проте такий підхід має суттєвий недолік – оскільки кожен агент моделюється незалежно, зазвичай обчислення лінійні для кількості агентів і квадратичні при моделюванні взаємодій. Тобто для сцени з n агентами та m елементами доріг складність досягає $O\big(n(n+m)\big)$. Це може бути проблемно у реальних складних міських середовищах, що складаються з сотень агентів.

Сценоцентричні (англ. scene-centric) моделі, на відміну від агентоцетричних, виконують кодування елементів сцени та агентів у спільній для всіх фіксованій координатній системі. Одним із традиційних варіантів реалізації такого підходу є моделі, що працюють з растеризованим представленнями «вид зверху». Сценоцентричні моделі поділяють представлення між всіма елементами сцени, що дозволяє робити обчислювально ефективніші прогнози. Проте недоліком є те, що вони повинні самі навчатися інваріантності, тому зазвичай вони поступаються агентоцентричним моделям в точності [61].

В цій роботі пропонується сценоцентричний підхід, де за початок координат приймається фокальний агент. На відміну від більшості сценоцентричних підходів він напряму використовує векторні дані семантичної карти, а не растеризацію, яка призводить до втрати інформації.

3.1.2 Архітектура

В основі моделі лежить структура кодувальник-декодувальник (англ. encoder-decoder), що зображено на рис. 3.1.

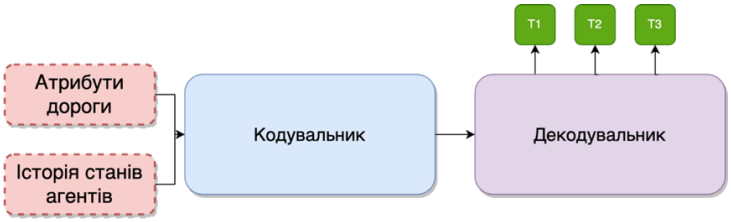

Рисунок 3.1 – Модель кодувальник-декодувальник

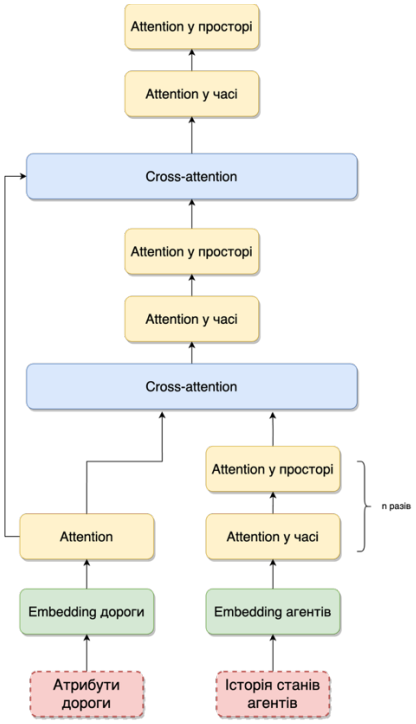

Рисунок 3.2 – Загальна схема кодувальника

Загальна структура кодувальника зображена на рис. 3.2. Він майже повністю складається з трансформер блоків із механізмами уваги. Перед потраплянням у ці блоки, відбувається вкладання (англ. embedding) або проєкція вхідних даних у багатовимірний простір. Проєкція відбувається за допомогою простої трансформації у повнозв'язному шарі $Proj\,(x_i) = relu\,(Wx_i + b)$, де $relu(x) = \max(0, x)$. Механізм уваги є еквіваріантним до перестановок. Для модальностей, де порядок даних містить корисну інформацію важливо порушити цю еквіваріантність. Зазвичай це робиться

за допомогою позиційних вкладень (англ. positional embeddings), які додаються до вхідних вкладень. Оскільки ще [54] показали, що використання синусоїди та вкладень із параметрами, що вивчаються моделлю, призводять до майже ідентичних результатів, то у цій роботі перевага надається простим синусоїдальним вкладенням, які не потребують вивчення додаткових параметрів.

Після цього дані проходять через блоки уваги, як у [54]. Щоб уникнути проблем, пов’язаних із великою кількістю обчислень для механізмів уваги на багатовимірних послідовностях, було використано факторизовану увагу. Тому увага послідовно і окремо застосовується для часового та просторового вимірів вхідних даних. Застосування уваги лише в часі дозволяє моделі вивчати траєкторії незалежно від агента. Аналогічно, застосування уваги лише до агентів дозволяє моделі вивчати взаємодії між ними незалежно від часового кроку. Для того, щоб одночасно враховувати ці залежності, модель просто чергує увагу до часу і агентів у наступних шарах.

Для поєднання даних різних модальностей (у цьому випадку – елементи дорожнього графа та станів агентів) використовується перехресна увага. В цій реалізації перехресної уваги запити обчислюються від агентів, а ключі та значення – від вкладень дорожньої інформації.

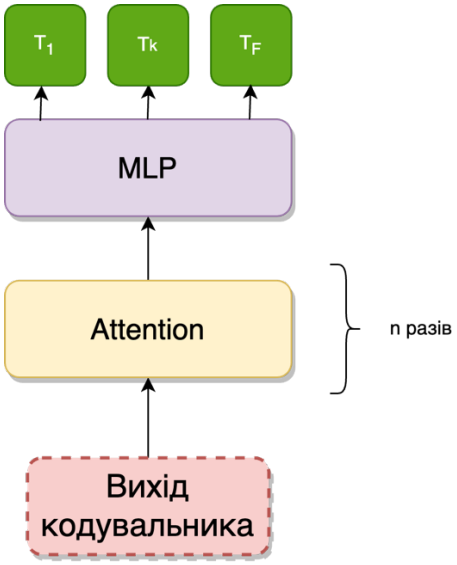

Рисунок 3.3 – Схема декодувальника

Декодувальник (рис. 3.3) також складається із блоків уваги. Майбутні рухи учасників дорожнього руху за своєю природою є стохастичними, оскільки учасники руху не можуть знати наміри один одного. Тому для моделі важливо вміти прогнозувати мультимодальні траєкторії. Один із найпопулярніших підходів було запропоновано у [62] – параметризація розподілу майбутніх траєкторій як суміші. В оригінальній роботі використовувалася суміш розподілів Ґауса, проте більш пізні роботи надають перевагу розподілу Лапласа. Для кожного агента i та компонента суміші f декодувальник повертає параметр локалізації $\mu_{if}^{t} \in \mathbb{R}^2$ та параметр масштабу $b_{if}^{t} \in \mathbb{R}^2$.

Ця модель дозволяє легко виконувати як спільні, так і маргінальні прогнози траєкторії. При спільному прогнозі, моделям необхідно враховувати майбутні взаємодії між агентами так, щоб прогнози були узгоджені один з одним. Різниця між задачами проілюстрована на рис. 3.4. Кожний колір позначає окремий прогноз. Зліва зображено маргінальний прогноз, в якому траєкторії незалежні між собою. Праворуч зображено

спільний прогноз і оцінки вказують на ймовірність всієї передбачуваної сцени, що складається з траєкторії трьох автомобілів.

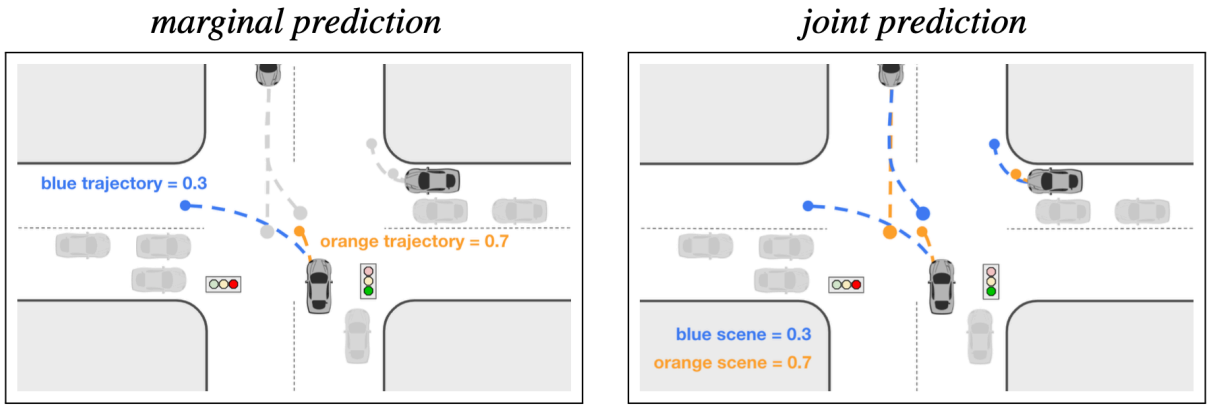

Рисунок 3.4 – Порівняння режимів прогнозування [61]

Для того, щоб виконувати спільне прогнозування, кожне майбутнє розглядається як узгоджене для всіх агентів. Тому, функція втрат зміщення агрегується для всіх агентів і часових кроків для побудови тензора втрат розміром F, де F – кількість мод майбутнього. Виконується зворотне поширення помилки через одне майбутнє, яке найкраще відповідає істині. Для маргінальних прогнозів, кожен агент розглядається окремо, тому агрегування по агентам не відбувається і функція втрат має розмірність $F \times A$, де A – кількість агентів. Замість агрегування, для кожного агента вибирається майбутнє з мінімальними втратами і виконується зворотне поширення помилки.

3.2 Графова нейронна мережа

Іншим підходом до прогнозування траєкторій може бути використання графової моделі. Це природній вибір для задачі, де взаємозв’язки між агентами та елементами сцени добре описуються графами. У цьому розділі пропонується модель глибокого навчання із використанням графових нейронних мереж або GNN, оскільки вони демонструють найкращі результати в моделюванні графів. За основу моделі взято ідеї з [63].

3.2.1 Вхідні дані

Одним із популярних підходів до представлення сцени, є використання растеризованих зображень, отриманих із семантичних карт. Проте, цей підхід має декілька недоліків. Одним із них є неминуча втрата інформації у процесі растеризації. Семантичні карти мають чітку структуру із складною топологією, якій моделям може бути важко навчитися при використанні растеризованих представлень. Наприклад пари смуг руху в протилежні сторони кардинально відрізняються від смуг руху в одну сторону, навіть якщо вони просторово близькі одна до одної. Вибір графової нейронної моделі дозволяє використовувати граф руху, побудований напряму із векторних даних семантичної карти, уникаючи втрати інформації.

Вхідні дані представлені у вигляді набору смуг руху та їх зв’язків. Кожна смуга руху містить послідовність точок із координатами. Вузлом графу є сегмент лінії між будь-якими послідовними точками. Координатами вузла є середнє координат його кінців. Для смуг визначається 4 типи з’єднань: попередник, наступник, лівий сусід та правий сусід – це смуги на які можна безпосередньо потрапити, не порушуючи правил дорожнього

руху. Цей формат даних надає важливу геометричну та семантичну інформацію для прогнозування руху. Приклад побудови графу сцени зображено на рис. 3.5. Зліва зображено центральну лінію смуги, її попередники, наступники, ліві та праві сусіди, які позначено червоним, помаранчевим, синім та фіолетовим кольорами відповідно. Праворуч зображено відповідний граф.

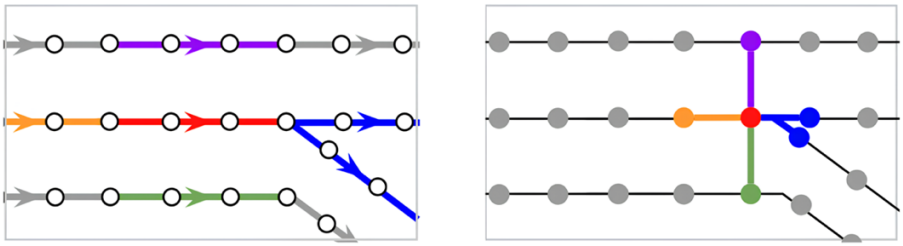

Рисунок 3.5 – Приклад побудови графу сцени [63]

3.2.2 Архітектура

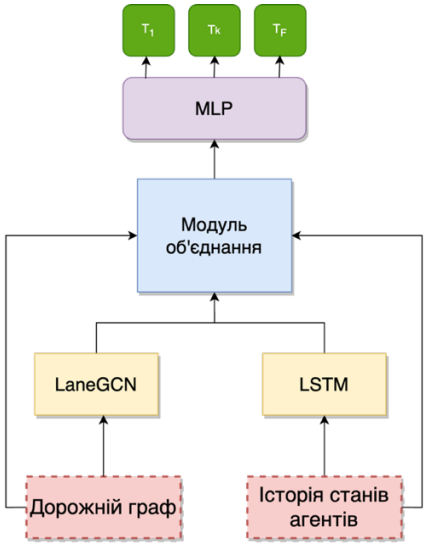

Рисунок 3.6 – Загальна архітектура моделі

Загальна архітектура моделі зображена на рис. 3.6. Модель складається з декількох модулів: кодувальник інформацію про минулі стани агентів, кодувальник дорожнього графу, модуль, якій об'єднує закодовану інформацію та модуль, що виконує прогнозування траєкторії.

Стани агентів – набір часових рядів, для моделювання яких зазвичай застосовують рекурентні нейронні мережі або одновимірні згорткові нейронні мережі. Врахування довгих закономірностей у часових рядах необхідне для покращення здатності моделі моделювати загальну тенденцію траєкторії та стиль водія. Для досягнення цього із згортковими мережами необхідно використовувати декілька різних шарів із різними розмірами ядра згортки, що збільшує складність моделі. Тому у цій моделі використовується різновид рекурентної нейронної мережі спеціально розроблений для аналізу довгострокових залежностей – LSTM.

Для кодування дорожнього графу використовується графова нейронна мережа. Популярним є використання оператору графової згортки або GCN, проте вона має обмеження. Звичайна GCN не розрізняє різні типи зв'язків, які були введені вище, що призводить до втрати інформації про напрямок. Також існує проблема в обробці довгих залежностей. Тому у [63] пропонується модифікація звичайної графової згортки – LaneGCN. Нижче наведено опис принципу її роботи.

Атрибут вершини у LaneGCN визначається наступним чином:

$$x_i = MLP_{shape}\left(v_i^{end} - v_i^{start}\right) + MLP_{loc}(v_i),$$

де MLP – багатошаровий перцептрон, v_i – координати вузла i, тобто центр між двома його крайніми точками, v_i^{start} та v_i^{end} – координати його кінців. Таким чином кодується інформація про форму (розмір та орієнтацію) і положення (координати центра) сегмента смуги.

Для агрегації із врахування типів ребер можна застосувати оператор:

$$Y = XW_0 + \sum_{i \in \{pre,suc,left,right\}} A_i X W_i,$$

де A_i та W_i – матриці суміжності та вагів відповідно. A_{suc} і A_{pre} – поширюють інформації наступників і попередників, а A_{left} та A_{right} – від лівих та правих сусідів відповідно.

Для більш точного прогнозування модель повинна враховувати залежності на великих відстанях вздовж напрямку руху, оскільки агенти із високою швидкістю можуть переміщатися на велику відстань за короткий час. У звичайних графах для цього застосовують розширення (англ. dilation), яке збільшує рецептивне поле нейронів. А саме:

$$Y = XW_0 + A_{pre}^k X W_{pre,k} + A_{suc}^k X W_{suc,k},$$

де A_i^k – матриця A_i в степені k. Розширений оператор застосовується тільки для попередника і наступника, оскільки довгострокові залежності, в основному, важливі вздовж цієї осі.

Після об'єднання попередніх рівнянь отримуємо багатомасштабний оператор LaneGCN, який використовує декілька розширень:

$$Y = XW_0 + \sum_{i \in \{left,right\}} A_i X W_i + \sum_{c=1}^{C} \left(A_{pre}^{k_c} X W_{pre,k_c} + A_{suc}^{k_c} X W_{suc,k_c} \right),$$

де k_c – розмір c-го розширення.

Після окремого кодування інформації про агентів та граф смуг руху, необхідно об'єднати цю інформацію та врахувати взаємодії між агентами та взаємодії між агентами та дорожніми елементами. Для моделювання

взаємодії між елементами дороги застосовується LaneGCN, а для інших взаємодій використовується стандартний механізм уваги.

Результат модуля об’єднання інформації передається в останній модуль, який виконує мультимодальне прогнозування траєкторії та складається із багатошарового перцептрона.

Висновки до розділу 3

У розділі наведено деталі архітектури запропонованих моделей – із використанням Трансформерів та заснованої на графовій нейронній мережі.

Обидві моделі використовують сценоцентричне представлення, яке є більш обчислювально ефективним ніж агентоцентричне. Проте, на відміну від більшості сценоцентричних моделей, позиція одного із агентів використовується в якості центра координатної системи.

Перша модель заснована на Трансформерах і має структуру кодувальник-декодувальник. Для ефективності обчислень уваги в багатовимірній послідовності, використовується факторизована увага. Тобто увага окремо обчислюється для часу і простору. Для поєднання даних дорожнього графу та історії станів використовується перехресна увага.

Друга модель також має структуру кодувальник-декодувальник. Вона використовує графову нейронну мережу для кодування дорожнього графу та рекурентну нейронну мережу для кодування історії станів агентів. Для поєднання цих представлень використовується механізм уваги.

Результатом роботи моделей є суміш розподілів Лапласа, тобто виконується мультимодальне прогнозування і враховується стохастичність намірів учасників дорожнього руху.

РОЗДІЛ 4 РОЗРОБКА МОДЕЛЕЙ ПРОГНОЗУВАННЯ ТРАЄКТОРІЇ

4.1 Набір даних

4.1.1 Вибір набору даних

Для успішного навчання моделі важливо обрати відповідний набір даних. У цьому розділі буде розглянуто існуючі в відкритому доступі набори даних.

Прогнозування траєкторії – задача, якій присвячена велика кількість робіт і ресурсів, тому для її розв'язання було зібрано багато даних. Історично велике значення мали набори ETH [47] та UCY [48], які містять історії координат агентів і присвячені прогнозу траєкторії пішоходів. Проте для сучасних моделей із урахуванням взаємодії цих наборів недостатньо, тому з’явилися значно більші набори із більшою кількістю інформації: Argoverse/Argoverse 2 [46], Lyft [59] та Waymo Open Motion [60].

Приклади із Waymo Open Motion, Lyft та Argoverse 2 Motion Forecasting зображено на рис. 4.1-4.3 відповідно.

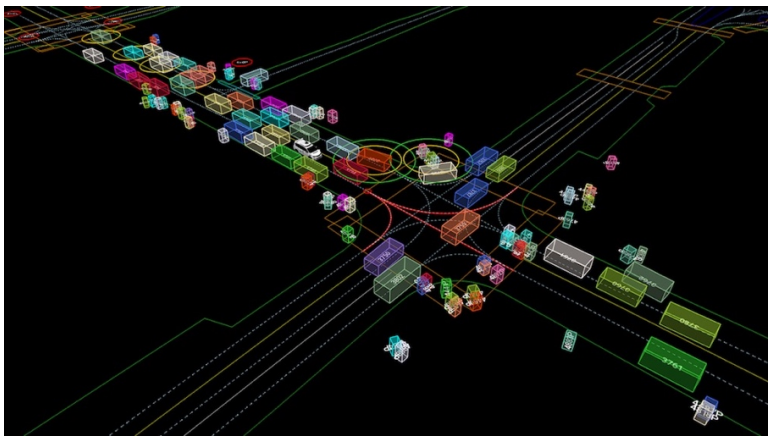

Рисунок 4.1 – Приклад сцени із Waymo Open Motion Dataset

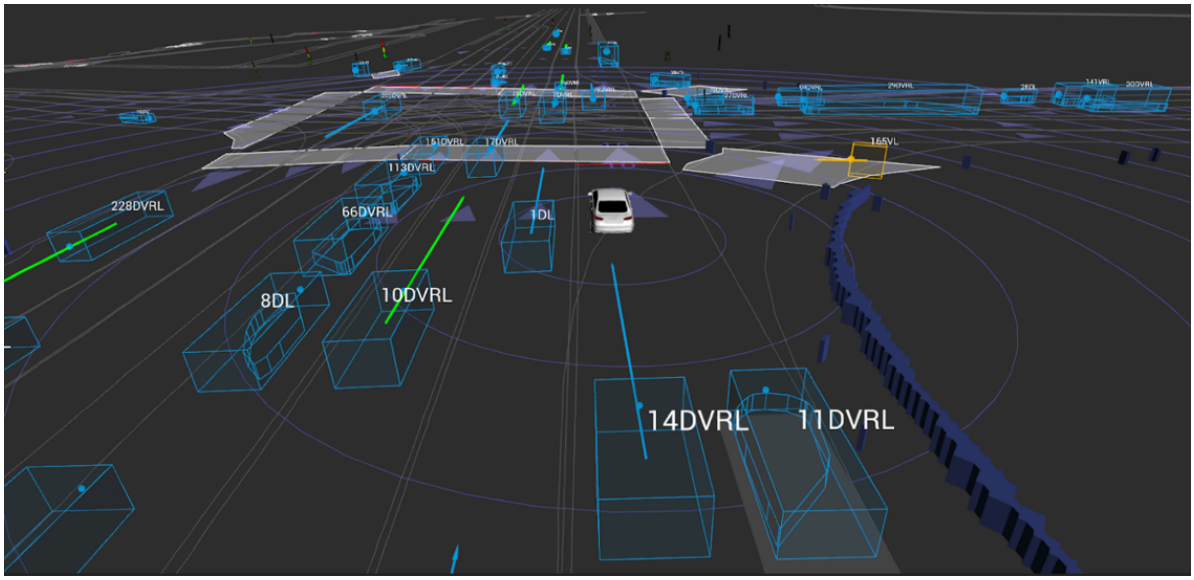

Рисунок 4.2 – Приклад сцени із Lyft Dataset

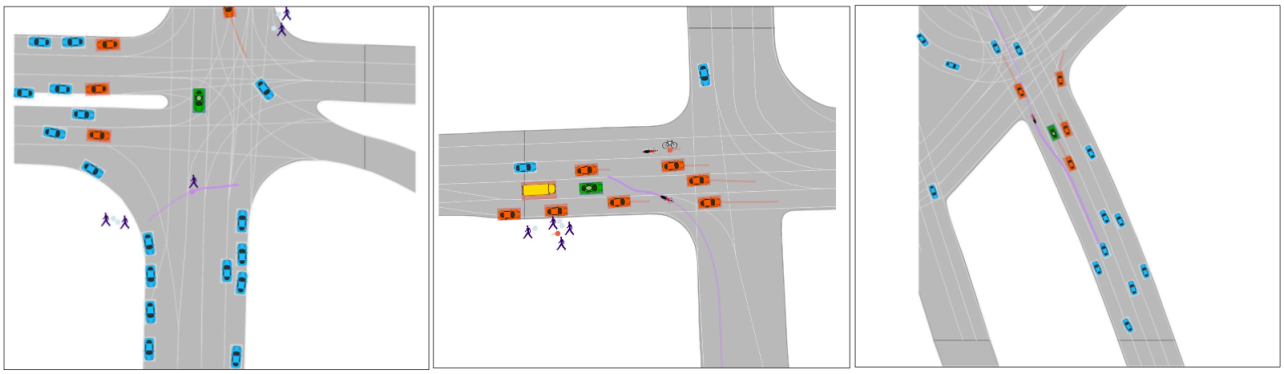

Рисунок 4.3 – Приклад сцени із Argoverse 2 Motion Forecasting Dataset

Порівняльну характеристику перелічених наборів даних наведено у таблиці 4.1.

Таблиця 4.1 – Порівняльна характеристика наборів даних [46]

	Argoverse	*Lyft*	*Waymo*	*Argoverse 2*
Сценаріїв	324000	170000	104000	250000
Унікальних траєкторій	11.7 млн	53.4 млн	7.6 млн	13.9 млн
Середня довжина траєкторії	2.48 с	1.8 с	7.04 с	5.16 с
Тривалість сценарію	5 с	25 с	9.1 с	11 с
Тривалість тестової частини	3 с	5 с	8 с	6 с
Частота вибірки	10 Гц	10 Гц	10 Гц	10 Гц
Міст	2	1	2	6
Унікальних доріг	290 км	10 км	1750 км	2220 км
Прогнозування багатьох агентів	-	+	+	+
«Цікавість»	+	-	+	+

«+» в графі «цікавість» означає, що після збору даних було відібрано найбільш складні та динамічні сценарії.

Для прогнозування траєкторії критично важливим аспектом є безпека в області з «довгими хвостами». Тобто необхідно, щоб модель вміла передбачувати і рідкісні події, а не лише оптимізувала поведінку на простих сценаріях.

Argoverse 2 Motion Forecasting Dataset складається саме з таких сценаріїв. Він включає менше траєкторій та сценаріїв, ніж деякі інші, проте вони більш складні. Це дозволяє зробити набір даних достатньо великим для навчання складних моделей та достатньо малим, щоб бути легкодоступним.

4.1.2 Argoverse 2

Набір даних складається з 250 000 сценаріїв, що не перетинаються один з одним, які були зібрані в 6 містах США. Він містить 10 типів об’єктів – 5 динамічних та 5 статичних. Кожний сценарій містить локальну векторну карту і 11с (10 Гц) даних траєкторії (координати, швидкість та кут повороту) для всіх агентів, які спостерігав его-автомобіль (англ. ego-vehicle). Перші 5 секунд кожного сценарію є вікном спостереження, а наступні 6 – горизонтом прогнозування.

У кожному сценарії є один «фокальний агент». Для їх траєкторій гарантовано існують дані протягом всього сценарію. Фокальні агенти було спеціально відібрано для максимізації цікавих взаємодій із об’єктами карти та із іншими агентами. Також підтримується мультиагентне прогнозування (англ. multi-agent forecasting). Для цього визначена підмножина «оцінених агентів» (англ. scored agents), для яких виконуються гарантії релевантності та мінімальної якості даних.

«Цікавість» траєкторії оцінюється евристично із урахування 5 факторів: категорія об’єкта, кінематика (наприклад різкі повороти, значні прискорення/сповільнення), складність карти (наприклад перехрестя, зміни смуги), соціальний контекст (скупчення агентів та агенти, що не являються автомобілями) і зв'язок з его-автомобілем (траєкторії, що перетинають його бажаний маршрут). Агент, який повністю спостережний у сценарії та має найбільшу оцінку цікавості вибирається фокальним.

У табл. 4.2 наведено класифікацію об’єктів сцени:

Таблиця 4.2 – Класифікація об’єктів сцени

Динамічні	*Статичні*
VEHICLE (транспортний засіб)	STATIC (статичний)
PEDESTRIAN (пішохід)	BACKGROUND (фон)
MOTORCYCLIST (мотоцикліст)	CONSTRUCTION (будівництво)
CYCLIST (велосипедист)	RIDERLESS_BICYCLE (велосипед без велосипедиста)
BUS (автобус)	UNKOWN (невідомий)

Розподіл об’єктів по типам зображено на рис. 4.4.

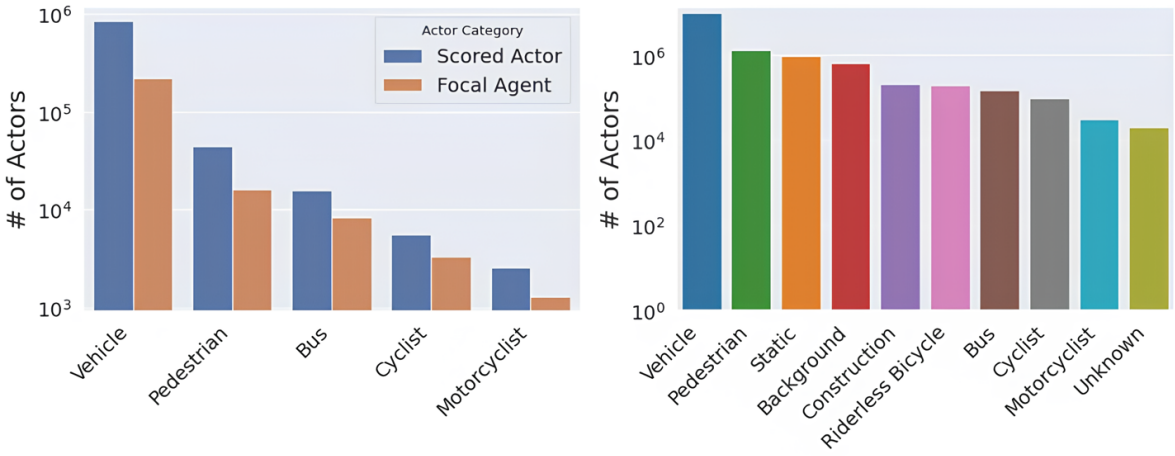

Рисунок 4.4 – Розподіл об’єктів по типам

Кожен сценарій набору даних містить власну локальну карту регіону (HD map). Вони містять:

Граф смуг руху. Це основний елемент HD map, який складається з графу $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, де $\mathcal{V}$ – індивідуальні сегменти смуг. Надаються 3D межі смуг руху у вигляді ламаних з роздільною здатністю 1 см.

Проїжджу частину. Надається у вигляді 3D полігонів, що дозволяє стисло передати інформацію і зберігати окремі карти для десятків тисяч сценаріїв. Полігони мають роздільну здатність 1 см.

Локальна карта кожного сценарію містить всі об’єкти в радіусі 100 метрів від траєкторії руху его-автомобілю.

Повний опис атрибутів HD map наведено в табл. 4.3.

Таблиця 4.3 – опис атрибутів семантичної карти

Сутність	*Атрибут карти*	*Опис*
Сегменти смуг руху	IS_INTERSECTION	Чи є сегмент перехрестям
	LANE_TYPE	Тип смуги
	LEFT/RIGHT LANE BOUNDARY	3D ламані із межею смуги
	LEFT/RIGHT MARK TYPE	Тип розмітки
	LEFT/RIGHT NEIGHBOR	Сусідні сегменти
	SUCCESSOR IDs	Наступні сегменти
Проїжджа частина	AREA BOUNDARY	3D полігони – місце, де автономний автомобіль може проїжджати без пошкоджень
Пішохідні переходи	EDGE1, EDGE2	Кінці двох сторін пішохідного переходу

4.2 Оцінка точності прогнозування траєкторії

Для порівняння моделей та визначення найкращої необхідно заздалегідь визначити критерії оцінювання – метрики. Правильна вибір оцінки особливо важливий в контексті прогнозування траєкторії та самокерованих транспортних засобів, оскільки неадекватні прогнози моделі можуть призвести до дорожньо-транспортних пригод при використанні в реальних умовах.

Більшість змагань та робіт із мультимодального прогнозування траєкторії використовують декілька метрик одночасно:

- частка помилок (англ. Miss Rate) (MR) – кількість сценаріїв, коли жодна з прогнозованих траєкторій не знаходиться в межах 2 метрів від справжньої;
- мінімальна остаточна похибка зміщення (англ. minimum Final Displacement Error) (minFDE) – Евклідова відстань між кінцевими точками найкращої з прогнозованих траєкторій та істиною;
- мінімальна середня помилка зміщення (англ. minimum Average Displacement Error) (minADE) – середня Евклідова відстань між найкращою прогнозованою траєкторію та істинною траєкторію;
- мінімальна остаточна похибка зміщення Браєра (англ. Brier minimum Final Displacement Error) (brier-minFDE) – метрика, схожа на minFDE, де до Евклідової відстані додається $(1-p)^2$, p – ймовірність найкращої прогнозованої траєкторії;
- мінімальна середня помилка зміщення Браєра (англ. Brier minimum Average Displacement Error) (brier-minADE) – метрика, схожа на minADE, де до середньої Евклідової відстані додається $(1-p)^2$, p – ймовірність найкращої прогнозованої траєкторії.

Ці метрики досить просто обчислювати та інтерпретувати, проте важливо пам’ятати, що вони не надають повної інформації. Прогнози із однаковою оцінкою можуть призводити до зовсім різних результатів. Наприклад, одним із завдань самокерованих транспортних засобів є

уникнення зіткнень із навколишніми об'єктами. Якщо прогнозована траєкторія агента не перетинається із майбутнім шляхом его-агента, а істинна – перетинається, то це можна вважати найгіршим сценарієм, який може призвести до ДТП. В той же час, проміжні випадки (такі як прогнозування більш раннього перетину траєкторії) не призводять до небезпечної ситуації і не є критичними, хоча деякі з них можуть мати ту ж оцінку, що і найгірший сценарій. Цю ситуацію проілюстровано на рис. 4.5.

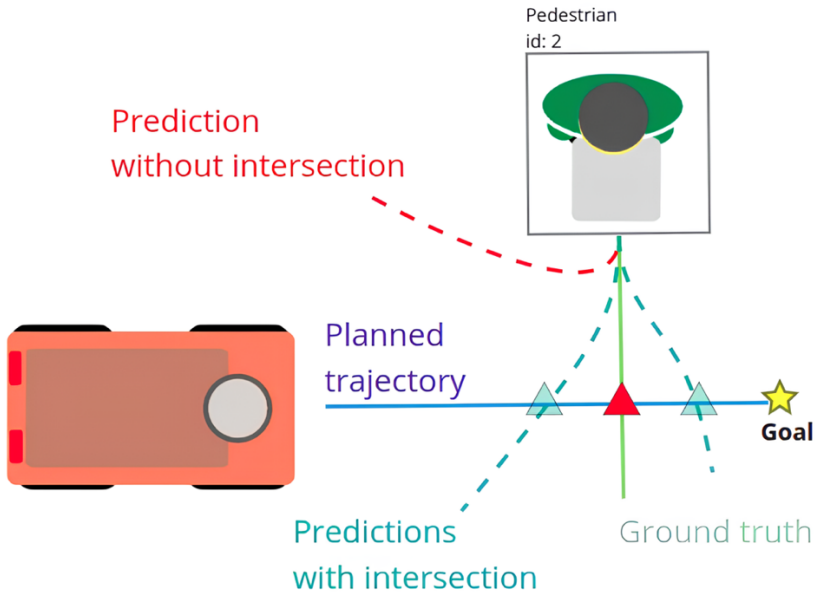

Рисунок 4.5 – Порівняння різних прогнозів [60]

Інший приклад зображено на рис. 4.6. Водій (у сірому) збирається повернути праворуч поряд із автономним транспортним засобом (у зеленому). Два прогнози поведінки водія зображено суцільними зеленою та фіолетовою лініями. Вони рівні за метриками, оскільки знаходяться на однаковій відстані від істинної траєкторії (у синьому). Проте зелений прогноз не має впливу на план руху безпілотного автомобіля (пунктирна синя лінія), тоді як інший прогноз змушує його виконати маневр для забезпечення безпеки.

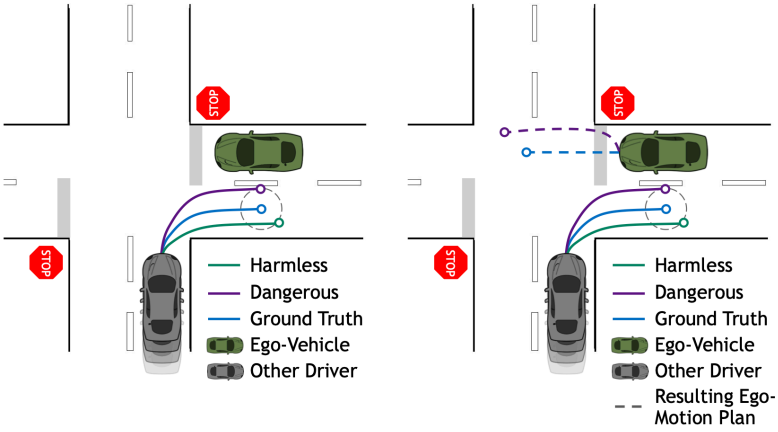

Рисунок 4.6 – Порівняння різних прогнозів [60]

Тому перед масовим застосування самокерованого автомобіля із модулем прогнозування траєкторії необхідні широкі контрольовані випробування в різноманітних умовах.

4.3 Вибір технологій

4.3.1 Мова програмування

Вибір мови програмування є важливим етапом у практичній реалізації моделей машинного навчання і може впливати на розвиток та продуктивність проекту. Найпопулярнішою мовою програмування у сфері аналізу даних та машинного навчання історично є мова Python. Вона відома своєю простотою та лаконічністю коду, що сприяє швидкому розробленню коду і дозволяє швидко будувати моделі та тестувати гіпотези. Python – кросплатформна інтерпретована мова, що дозволяє просто і наочно аналізувати та візуалізувати дані на будь-якому пристрої. Популярність

мови означає велику та активну спільноту розробників, що означає доступність документації, підтримку та обмін знаннями. Рішення проблем легко знаходити на спеціалізованих форумах. Python має багато потужних бібліотек та фреймворків для аналізу даних (NumPy, Pandas, SciPy тощо), машинного навчання (scikit-learn, TensorFlow, PyTorch) та візуалізації даних (matplotlib, seaborn, plotly).

Головним потенціальним недоліком Python є його швидкість. У порівнянні із більш низькорівневими мовами, такими як C++ або Rust, чистий Python набагато повільніший. Проте більшість функцій бібліотек машинного навчання написані на C/C++, і мають лише Python оболонку, що дозволяє поєднати швидкість виконання низькорівневих мов із зручністю Python. Це може означати, що у випадку, коли деяких специфічних функцій не існує в бібліотеках із оптимізованим кодом, на Python важко створити їх швидку реалізацію. Але кількість бібліотек та функцій, які вони надають достатньо велика і самі функції бувають достатньо низькорівневі, що ймовірність такої ситуації досить низька.

4.3.2 Бібліотеки

Машинне навчання мовою Python неможливе без спеціалізованих бібліотек.

Основні допоміжні бібліотеки, які були використані у цій роботі включають:

NumPy – потужна бібліотека, спеціально розроблена для наукових обчислень. Вона надає підтримку швидких операцій із багатовимірними масивами та широкий набір функцій. В основі бібліотеки лежить ndarray – багатовимірний масив певного типу

даних, який дозволяє ефективно маніпулювати великими наборами даних. NumPy надає розширені можливості для індексації та зрізів масивів, що дозволяє ефективно виконувати вибірку та модифікацію підмасивів даних. Ця бібліотека є основою для багатьох інших – таких як scipy, pandas та matplotlib. Більшість фреймворків, пов'язаних із машинним навчанням сумісні із Numpy.

Matplotlib – бібліотека для побудови графіків та створення статичних та динамічних візуалізацій даних у Python. Вона надає вбудовані засоби для побудови різноманітних типів графіків (ламаних, гістограм, стовпчастих діаграм тощо) та гнучку систему фігур та підграфікв для організації складних візуалізацій. Matplotlib також дозволяє легко експортувати отримані графіки в різні формати файлів – такі як png, pdf, svg та інші.

SciPy – бібліотека, яка використовується для наукових та технічних обчислень в Python. Вона заснована на Numpy та надає додатковий функціонал для оптимізації, обробки сигналів, статистичних функцій та іншого. Важливою перевагою бібліотеки є підтримка розріджених матриць та операцій лінійної алгебри, пов'язаних із ними.

Для моделювання нейронних мереж зазвичай використовується один із двох фреймворків – TensorFlow від Google та PyTorch від Meta (колишній Facebook). До основних переваг, притаманних обом продуктам відносяться:

Підтримка швидких паралельних обчислень, зокрема із використанням GPU, що необхідно для тренування великих нейронних мереж.

Автоматичне диференціювання. Цей механізм дозволяє обчислювати градієнти функцій без явного обчислення похідних. Основна ідея полягає в тому, що система відстежує всі операції, які виконуються над тензорами та обчислює похідні. Цей механізм є основним для тренування глибоких нейронних мереж, оскільки градієнти необхідні

для оновлення параметрів моделі за допомогою методів оптимізації, таких як градієнтний спуск.

Підтримка нейронних мереж. Фреймворки мають багатий набір інструментів для легкого створення та навчання нейронних мереж будь-якої складності.

PyTorch у середньому повільніший за TensorFlow в обчисленнях, проте має глибшу інтеграцію в мову Python та сприяє швидкому прототипуванню моделей. Тому PyTorch часто обирають дослідники, а TensorFlow використовується в промисловості.

Оскільки в цій роботі важливо швидко реалізувати та порівняти моделі, то для реалізації нейронних мереж було обрано фреймворк PyTorch.

4.3.3 Хмарні обчислення

Хмарні обчислення є важливим елементом для розвитку та ефективного використання систем машинного навчання. Вони надають доступ до великих обчислювальних ресурсів, що є ключовим чинником для тренування складних моделей та обробки великих обсягів даних.

Хмарні платформи надають широкий спектр інфраструктури: віртуальні машини, обчислювальні ресурси та сховища даних. Це дозволяє науковцям і інженерам зосередитися на розробці та оптимізації моделей, не витрачаючи значних зусиль на управління обчислювальною інфраструктурою.

У цій роботі для тренування моделей використовується AWS, як провідний постачальник послуг хмарних обчислень. Він пропонує низку інструментів та сервісів, призначених спеціально для машинного навчання. Сервіси, такі як Amazon SageMaker, дозволяють легко тренувати,

налаштовувати та розгортати моделі машинного навчання. AWS також забезпечує масштабовані обчислювальні ресурси для великих завдань навчання.

4.4 Результати

Результати моделей на тестовому наборі даних зображено в табл. 4.4. Для порівняння також наведено одну агентоцентричну модель – HiVT [66] та одну сценоцетрично модель, що використовує растеризовані зображення – Multipath [62].

Таблиця 4.4 – Порівняння моделей

Модель	Вхід	Brier-minFDE	minFDE	minADE	MR
Трансформер	Сценоцентрична	1.9	1.2	0.8	0.13
Графова	Сценоцентрична	2.0	1.35	0.9	0.16
HiVT	Агентоцентрична	1.84	1.14	0.77	0.12
Multipath	Сценоцентрична	2.4	2.04	1.0	0.4

Результати показують, що модель, заснована на архітектурі Трансформер демонструє результати порівнянні із агентоцентричною моделлю, яка вимагає набагато більше обчислювальних потужностей. Запропонована графова модель поступається Трансформеру, хоча обидві запропоновані сценоцентричні моделі кращі за Multipath, який використовує растеризовані зображення в якості вхідного представлення сцени.

Висновки до розділу 4

У розділі наведено деталі реалізації та тестування запропонованих моделей.

Для тренування та оцінки точності використовувався набір даних Argoverse 2 Motion Forecasting Dataset, оскільки в ньому зібрано багато сценаріїв, що були спеціально відібрані на «цікавість» та складність.

Застосовуються стандартні для задачі метрики, які оцінюють геометричні відхилення кращої із прогнозованих траєкторій від істинної. Існуючі метрики не показують повної картини, оскільки траєкторії із однаковими показниками можуть призвести до зовсім різних результатів. Тому перед масовим застосуванням необхідне їх широке тестування в реальних умовах.

Для практичної реалізації моделей обрано мову програмування Python та фреймворк PyTorch як найбільш зручні інструменти для машинного навчання. Для тренування використано сервіс хмарних обчислень AWS.

Аналіз результатів показує, що сценоцентрична модель із використанням Трансформерів демонструє точність на рівні із агентоцентричними моделями, використовуючи менше ресурсів. Обидві запропоновані моделі кращі за сценоцентричну модель, яка використовує растеризовані зображення в якості представлення вхідних даних.

РОЗДІЛ 5 РОЗРОБКА СТАРТАП-ПРОЄКТУ

У цьому розділі розглядається оцінка основних характеристик програмного продукту, розробленого для інтелектуальних систем контролю самокерованих автомобілів.

Методи функціонально-вартісного аналізу дозволять оцінити різні варіанти реалізації продукту та обрати найбільш оптимальну із стратегій.

Функціонально-вартісний аналіз (ФВА) – це методологія, що використовує технологічний підхід для об'єктивної оцінки реальної вартості продукту або послуги, незалежно від організаційної структури підприємства. Метою ФВА є виявлення потенційних можливостей зменшення витрат шляхом удосконалення виробничих процесів та досягнення оптимального співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для реалізації аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає визначення послідовності етапів розробки продукту, визначення загальних витрат (річних) та обсягу робочого часу, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

У ФВА розглядаються різні аспекти продукту або системи, такі як функції, які вони виконують, елементи, з яких вони складаються, взаємозв'язки між цими елементами, а також вартість, пов'язана з кожною функцією або елементом. Метою ФВА є забезпечення оптимального співвідношення між функціональністю і вартістю продукту або системи.

ФВА може бути застосований у різних галузях, включаючи розробку нових продуктів, оптимізацію виробничих процесів, підвищення якості продукції і зниження витрат. Він дозволяє виявляти недоліки і можливості для вдосконалення продукту або системи, сприяє прийняттю обґрунтованих

рішень щодо його подальшого розвитку і допомагає знижувати витрати без втрати функціональності.

5.1 Постановка задачі проєктування

Для проведення техніко-економічного аналізу розробки системи інтелектуального контролю застосовується метод ФВА. Оскільки вирішення завдань, пов'язаних із проектуванням та реалізацією компонентів, впливає на всю систему, кожна окрема підсистема повинна відповідати своїм функціональним вимогам. Фактичний аналіз охоплює вивчення функцій програмного продукту, призначеного для збору, обробки та аналізу даних у сфері класифікації.

Технічні вимоги до програмного продукту включають наступне:

- висока точність на надійність;
- можливість обробки даних в реальному часі;
- здатність інтегруватися із існуючими системами або програмними рішеннями для обміну даними та інформацією;
- мінімальні витрати на впровадження програмного продукту.

5.2 Обґрунтування функцій програмного продукту

Головною функцією F_0 є розробка програмного продукту для прогнозування траєкторії агентів. Серед основних функцій можна виділити наступні:

- F_1 – вибір мови програмування;

- F_2 – вибір бібліотеки або фреймворку для машинного навчання;
- F_3 – вибір середовища розробки.

Кожна з вищенаведених функцій має декілька варіантів реалізації.

Функція F_1:

- Pyhton;
- C++;
- Matlab.

Функція F_2:

- Scikit-learn;
- TensorFlow;
- PyTorch.

Функція F_3:

- PyCharm;
- Vim;
- Matl.ab.

Морфологічну карту варіантів реалізації основних функцій зображено на рис. 5.1

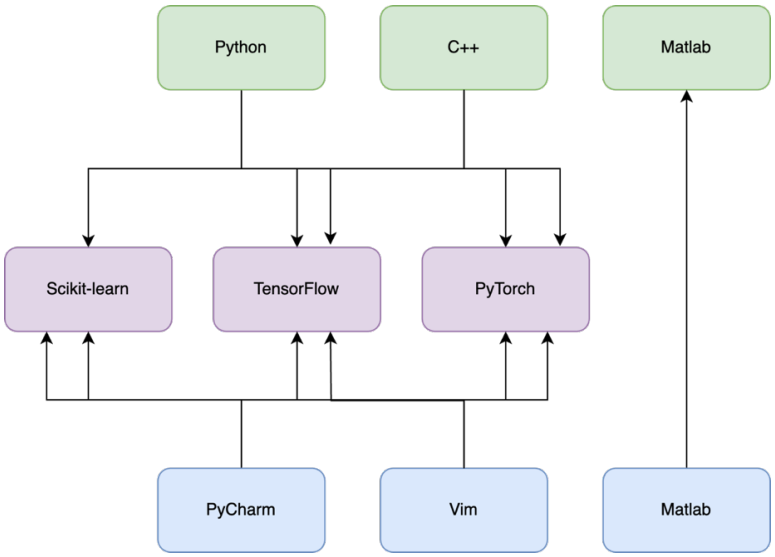

Рисунок 5.1 – Морфологічна карта

Позитивно-негативна матриця варіантів основних функцій наведена у табл. 5.1.

Таблиця 5.1 – Позитивно-негативна матриця

Функція	*Варіант реалізації*	*Переваги*	*Недоліки*
F_1	I	Кросплатформеність, безкоштовність, зручність, широкий вибір бібліотек	Швидкість, обмежена підтримка паралельного програмування
	II	Висока швидкість, повна підтримка паралельного програмування і всіх можливостей процесора та відеокарти	Складність у використанні, менша кількість доступних бібліотек машинного навчання, ручне керування пам’яттю
	III	Кросплатформеність, символьні обчислення	Ціна, ускладнена інтеграція із іншими мовами, низька популярність
F_2	I	Простий синтаксис, легкість у розробці	Обмежена підтримка нейронних мереж, відсутня підтримка використання відеокарти
	II	Підтримка нейронних мереж, підтримка використання відеокарти	Складний синтаксис, складне управління даними, посередня інтеграція із вбудованими засобами мови програмування та іншими бібліотеками
	III	Підтримка нейронних мереж і використання відеокарти, повна інтеграція із вбудованими засобами мови програмування та іншими бібліотеками	Синтаксис складніший, ніж в scikit-learn, відсутність готових функцій навчання

Закінчення табл. 5.1

F_3	I	Зручний інтерфейс, багато інструментів, підтримка можливості інтерактивної розробки, підтримка інтелектуальних підказок	Повний функціонал доступний лише у платній версії, займає багато місця на диску, потребує порівняно потужний комп'ютер
	II	Можливість для повної кастомізації, займає мало місця, не потребує ресурсів, можливість повної відмови від використання миші	Складність у навчанні
	III	Єдиний варіант програмування мовою Matlab, можливість інтерактивної розробки	Потребує багато ресурсів, менш зручний, ніж аналоги для інших мов програмування, підтримує лише Matlab

На основі аналізу позитивно-негативної матриці, можна зробити висновок, що деякі варіанти реалізації функції варто відкинути, оскільки вони не відповідають вимогам.

Функція F_1:

Перевага надається зручним, багатофункціональним та безкоштовним мовам. Тому Matlab має бути відкинутий. C++ має бути відкинутим як занадто складний вибір для машинного навчання

Функція F_2:

Необхідні бібліотеки, що підтримують використання відеокарти та нейронних мереж, тому варіант із scikit-learn може бути відкинутий.

Функція F_3:

Оскільки третій варіант підтримує лише мову Matlab, його необхідно відкинути. Перевага надається зручним та функціональним середовищам розробки, тому vim теж відкидається.

Таким чином, розглядають такі варіанти реалізації.

1. $F_{1a} - F_{2b} - F_{3a}$.
2. $F_{1a} - F_{2c} - F_{3a}$.

5.3 Обґрунтування системи параметрів програмного продукту

Для оцінювання якості розглянутих функцій необхідно розглянути вибір системи параметрів.

Для характеристики програмного продукту буде використано наступні параметри:

- X_1 – потенційний об’єм програмного коду;
- X_2 – швидкодія мови програмування;
- X_3 – споживання пам’яті.

Із урахуванням вимог замовника та умов, що визначають експлуатацію програмного продукту, визначаються гірші, середні та кращі значення параметрів, як вказано в таблиці 5.2.

Таблиця 5.2 – Основні параметри програмного продукту

Назва параметра	*Умовні позначення*	*Одиниці виміру*	*Значення параметра*		
			гірші	*середні*	*кращі*
Об’єм коду	X_1	Кількість рядків	2500	1500	1000
Швидкодія	X_2	оп/с	10^6	10^7	10^8
Пам’ять	X_3	Мб	4096	2048	1024

За даними таблиці можна побудувати графічні характеристики параметрів (рис. 5.2 – 5.4)

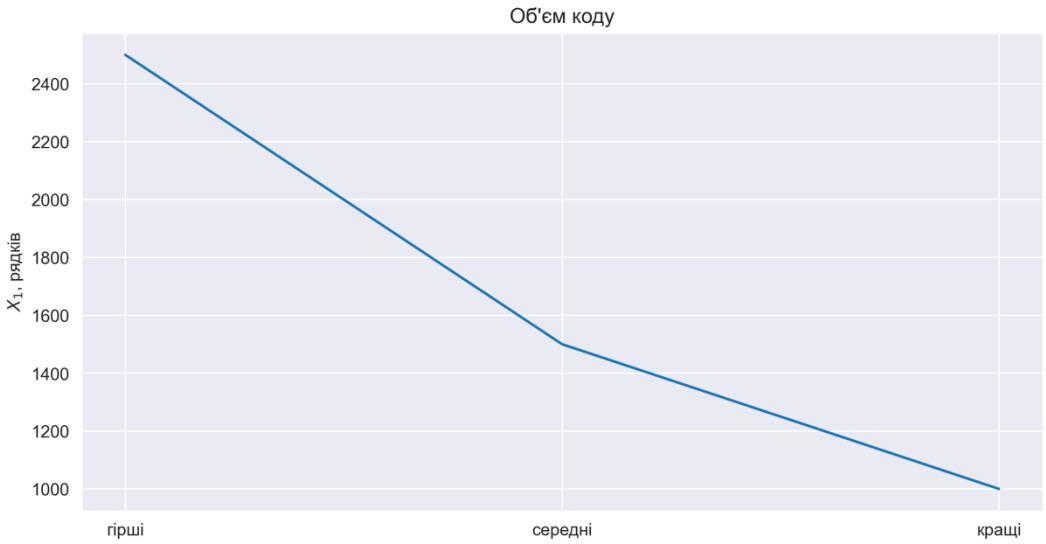

Рисунок 5.2 – Потенційний об’єм програмного коду

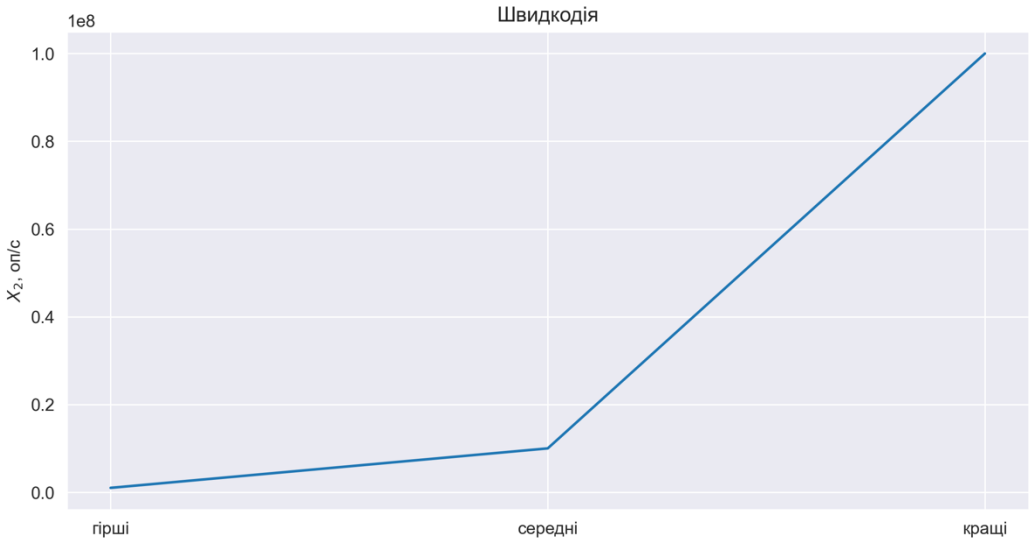

Рисунок 5.3 – Швидкодія мови програмування

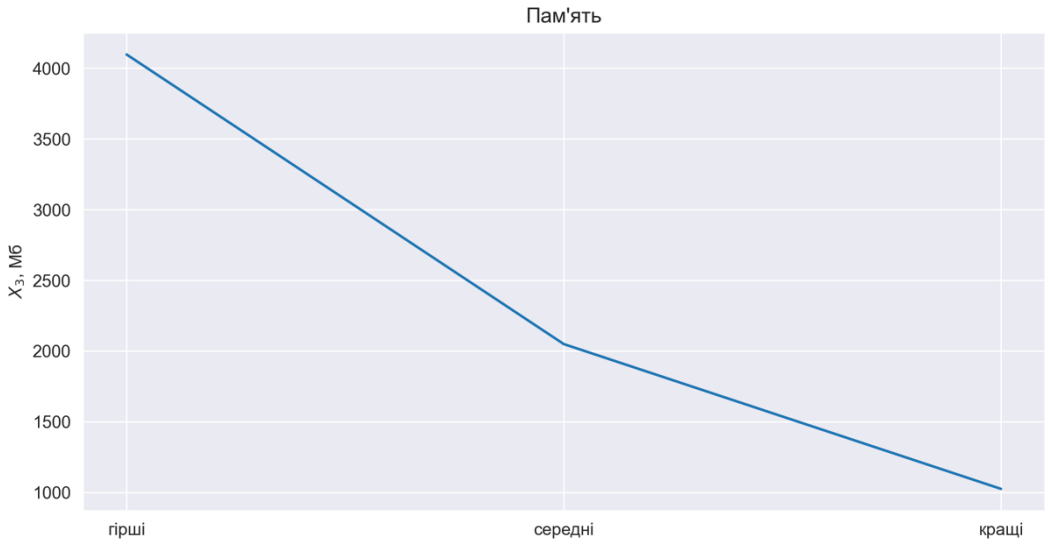

Рисунок 5.4 – Споживання пам’яті

Після визначення параметрів, проводиться їх ранжування за допомогою методу попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Після ретельного обговорення та аналізу кожен експерт оцінює значущість кожного параметра для поставленої мети.

Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведено в табл. 5.3.

Таблиця 5.3 – Результати експертного ранжування

Параметр	*Ранг*							*Сума* R_i	*Відхилення* Δ_i	Δ_i^2
	1	2	3	4	5	6	7			
X_1	1	2	2	1	2	2	2	12	-2	4
X_2	3	3	3	3	3	3	3	21	7	49
X_3	2	1	1	2	1	1	1	9	-5	25

Для перевірки ступені достовірності експертних оцінок, обчислимо наступні параметри.

а) Сума рангів кожного з параметрів та загальна сума рангів:

$$R = \sum_{ij} r_{ij} = \sum_{i} R_i = 42.$$

б) Середня сума рангів:

$$T = \frac{1}{n} R_i = 14.$$

в) Загальна сума квадратів відхилення:

$$S = \sum_{i} \Delta_i^2 = 78.$$

г) Коефіцієнт узгодженості (конкордації):

$$W = \frac{12S}{N^2(n^3 - n)} = 0.796 > W_k = 0.67.$$

Оскільки обчислений коефіцієнт конкордації перевищує нормативні 0.67, то ранжування вважається достовірним.

Використовуючи результати ранжування, можна провести попарне порівняння всіх параметрів. Результати наведено в табл. 5.4.

Таблиця 5.4 – Попарне порівняння параметрів

Параметри	*Експерти*							*Підсумкова оцінка*	*Числове значення коефіцієнтів переваги*
	1	2	3	4	5	6	7		
X_1, X_2	<	<	<	<	<	<	<	<	0.5
X_1, X_3	<	>	>	<	>	>	>	>	1.5
X_2, X_3	>	>	>	>	>	>	>	>	1.5

Тут числове значення переваги визначається як:

$$a_{ij} = \begin{cases} 1.5, & x_i > x_j \\ 1.0, & x_i = x_j \\ 0.5, & x_i < x_j \end{cases}$$

Нехай $A = \|a_{ij}\|$ – матриця числових переваг. Тоді вагомість K_{B_i}:

$$K_{B_i} = \frac{b_i}{\sum_i b_i},$$

де $b_i = \sum_j a_{ij}$ – вагомість параметра i

Вагомість розраховується ітеративно. Після першого кроку використовується наступна формула:

$$K_{B_i} = \frac{b_i'}{\sum_i b_i},$$

Результати ітерацій наведено в табл. 5.5.

Таблиця 5.5 – Розрахунок вагомості параметрів

i	j			*Перша ітерація*		*Друга ітерація*	
	X_1	X_2	X_3	B_i	K_{B_i}	$B_i^{(2)}$	$K_{B_i}^{(2)}$
X_1	1	0.5	1.5	3	0.(3)	8	0.32
X_2	1.5	1	1.5	4	0.(4)	11.5	0.46
X_3	0.5	0.5	1	2	0.(2)	5.5	0.22
Разом				9	1	25	1

Оскільки значення на другій ітерації відрізняються менше ніж на 2% від попередніх, додаткові ітерації не потрібні.

5.4 Аналіз якості варіантів реалізації функцій

Коефіцієнт технічного рівня для кожного варіанта реалізації програмного забезпечення можна обчислити за наступною формулою:

$$K_K(j) = \sum_i K_{B_i}(j) B_i(j)$$

Результати обчислень наведені у табл. 5.6.

Таблиця 5.6 – Показники якості

Варіант реалізації	*Бальна оцінка*			*Коефіцієнт вагомості*			*Коефіцієнт якості параметру*			*Коефіцієнт технічного рівня*
	X_1	X_2	X_3	X_1	X_2	X_3	X_1	X_2	X_3	
$F_{1a} - F_{2b} - F_{3a}$	8	7	9	0.32	0.46	0.22	2.56	3.22	1.98	7.76
$F_{1a} - F_{2c} - F_{3a}$	8	10	9				2.56	4.6	1.98	9.14

Другий варіант має найбільший коефіцієнт технічного рівня, отже є найкращим варіантом.

5.5 Економічний аналіз варіантів розробки програмного забезпечення

Для оцінки вартості розробки необхідно виконати розрахунок трудомісткості.

Розробка поділяється на 2 завдання:

1. Розробка архітектури моделі.
2. Розробка програмної оболонки.

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. Перше завдання має групу складності 1, а друге – третю.

Загальна трудомісткість:

$$Т_0 = Т_Р * К_П * К_{СК} * К_М * К_{СТ} * К_{СТ.М},$$

де $Т_Р$ – трудомісткість розробки програмного забезпечення,

$К_П$ – поправочний коефіцієнт,

$К_{СК}$ – коефіцієнт складності вхідної інформації,

$К_М$ – коефіцієнт рівня мови програмування,

$К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм,

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

Для першого завдання трудомісткість дорівнює $Т_Р = 30$ людино-днів, як для завдання степеню новизни А та групи складності 1. Вид нормативно-довідкової інформації враховує поправочний коефіцієнт $К_П = 1.6$. $К_{СК} = 1$ – коефіцієнт, що враховує складність контрою вхідної та вихідної інформації. При розробці використовуються стандартні модулі, тому $К_{СТ} = 0.9$.

Тоді, загальна трудомісткість першого завдання рівна:

$$T_1 = 30 * 1.6 * 0.9 = 43.2 \text{ людино-днів.}$$

Для другого завдання маємо $Т_Р = 26$, $К_П = 1.25$, $К_{СК} = 1$ та $К_{СТ} = 0.9$. Маємо:

$$T_2 = 26 * 1.25 * 0.9 = 29.25 \text{ людино-днів.}$$

Тоді загальна трудомісткість для кожного з можливих варіантів:

$$T_I = (43.2 + 29.25 + 9) * 8 = 651.6 \text{ людино-годин,}$$

$$T_{II} = (43.2 + 29.25 + 7) * 8 = 635.6 \text{ людино-годин.}$$

Перший варіант є найбільш трудомістким.

Розробкою займаються два спеціалісти із машинного навчання із окладом 51354 грн. та один програміст із окладом 46920 грн. Тоді середня зарплата за годину:

$$С_ч = \frac{M}{T_m t},$$

де M – місячний оклад працівників, T_m – кількість робочих днів у тижні, t – кількість робочих годин в день.

$$С_ч = \frac{51354 * 2 + 46920}{3 * 21 * 8} = 296.88.$$

Тоді, заробітна плата обчислюється за формулою:

$$С_{ЗП} = С_ч * Т_i * К_Д,$$

де $С_ч$ – погодинна оплата праці програміста,

$Т_i$ – трудомісткість завдання i,

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Маємо:

I. $С_{ЗП} = 296.88 * 651.6 * 1.2 = 232136.41$ грн.

II. $С_{ЗП} = 296.88 * 651.6 * 1.2 = 226436.31$ грн.

Відрахування на ЄСВ (єдиний соціальний внесок) становить 22%:

I. $С_{ЗП} = С_{ЗП} * 0.22 = 51070.01$ грн.

II. $С_{ЗП} = С_{ЗП} * 0.22 = 49816$ грн.

Необхідно обчислити витрати на оплату однієї машино-години ($С_Г$). Одна електронно-обчислювальна машина (ЕОМ) обслуговує одного програміста з окладом 46920 грн. Із коефіцієнтом зайнятості 0.3 отримуємо:

$$С_Г = 12 * 46920 * 0.3 = 168912.$$

З урахування додаткової заробітної плати:

$$С_{ЗП} = С_Г(1 + К_З) = 202694.4.$$

Відрахування ЄСВ:

$$С_{ВІД} = С_{ЗП} * 0.22 = 44592.77.$$

Амортизаційні відрахування розраховується при амортизації 25% та вартості ЕОМ – 40000 грн.

$$С_А = К_{ТМ} * К_А * Ц_{ПР} = 1.2 * 0.25 * 40000 = 12000,$$

де $К_{ТМ}$ – коефіцієнт, що враховує витрати на транспортування та монтаж,
$К_А$ – річна норма амортизації,
$Ц_{ПР}$ – ціна приладу.

Витрати на ремонт і профілактику обчислюються як:

$$С_Р = К_{ТМ}Ц_{ПР}К_Р = 1.2 * 40000 * 0.05 = 2400,$$

де $К_Р$ – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховується за формулою:

$$Т_{ЕФ} = (Д_К - Д_В - Д_С - Д_Р) * t_З * К_В,$$

де $Д_К$ – кількість днів у календарному році,

$Д_В, Д_С$ – кількість вихідних та святкових днів відповідно,

$Д_Р$ – кількість планових ремонтів устаткування,

$t_З$ – кількість робочих годин у день,

$К_В$ – коефіцієнт використання приладу.

Маємо:

$$Т_{ЕФ} = (365 - 104 - 8 - 11) * 8 * 0.7 = 1355.2.$$

Витрати на оплату електроенергії розраховуються за формулою:

$$С_{ЕЛ} = Т_{ЕФ} * N_С * К_З * Ц_{ЕН},$$

де $N_С$ – середньо-споживча потужність приладу,

$К_З$ – коефіцієнт зайнятості приладу,

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Маємо:

$$С_{ЕЛ} = 1355.2 * 0.55 * 0.2 * 5.1 = 760.27.$$

Накладні витрати обчислюються за формулою:

$$С_Н = Ц_{ПР} * 0.67 = 26800.$$

Тоді, маємо річні експлуатаційні витрати:

$$C_{ЕКС} = C_{ЗП} + C_{ВІД} + C_{А} + C_{Р} + C_{ЕЛ} + C_{Н},$$

$$C_{ЕКС} = 202694.4 + 44592.77 + 12000 + 2400 + 760.27 + 26800 = 289247.$$

Собівартість однієї машино-години ЕОМ:

$$C_{М-Г} = \frac{C_{ЕКС}}{T_{ЕФ}} = 213.4.$$

Оскільки всі роботи, пов’язані з розробкою програмного забезпечення потребують ЕОМ, витрати на оплату машинного часу становлять:

$$C_{М} = C_{М-Г} * T$$

I. $C_{М} = 213.4 * 651.6 = 139051.44$

II. $C_{М} = 213.4 * 635.6 = 135637.04$

Накладні витрати складають 67% заробітної плати:

$$C_{Н} = C_{ЗП} * 0.67.$$

I. $C_{Н} = 232136.41 * 0.67 = 155531.4$

II. $C_{Н} = 226436.31 * 0.67 = 151712.3$

Отже, вартість розробки програмного забезпечення становить:

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_{М} + C_{Н}.$$

I. $\text{С}_{\text{ПП}} = 232136.41 + 44592.77 + 139051.44 + 155531.4 = 571312.02$ грн

II. $\text{С}_{\text{ПП}} = 226436.31 + 43497.8 + 135637.04 + 151712.3 = 557283.45$ грн

5.6 Вибір кращого варіанту програмного забезпечення техніко-економічного рівня

Обчислимо коефіцієнт техніко-економічного рівня за формулою:

$$\text{К}_{\text{ТЕР}\,j} = \frac{K_{Kj}}{\text{С}_{\Phi j}}.$$

I. $\text{К}_{\text{ТЕР}j} = \frac{7.76}{571312.02} = 1.36 \times 10^{-5}.$

II. $\text{К}_{\text{ТЕР}j} = \frac{9.14}{557283.45} = 1.64 \times 10^{-5}.$

З оцінок випливає, що найбільш ефективним є другий варіант реалізації програми.

Цей варіант передбачає використання:

- мови програмування Python;
- фреймворку для машинного навчання PyTorch;
- середовища розробки програмного забезпечення PyCharm.

Даний набір опцій надає можливість швидкої та зручної реалізації програмного забезпечення, при цьому зберігаючи швидкість роботи програми.

Висновок до розділу 5

У розділі розглянуто оцінку основних характеристик програмного продукту, розробленого для інтелектуальних систем контролю самокерованих автомобілів.

Методи функціонально-вартісного аналізу дозволили оцінити різні варіанти реалізації продукту та обрати найбільш оптимальну із стратегій, яка полягає у використанні мови програмування Python, фремйворку для машинного навчання PyTorch та середовища розробки програмного забезпечення PyCharm.

ВИСНОВКИ

У цій роботі розглянуто прогнозування траєкторії учасників дорожнього руху. Запропоновано та реалізовано моделі нейронних мереж для розв’язання цієї задачі. Для цього використано мову програмування Python та фреймворк PyTorch.

У першому розділі висвітлена актуальність задачі, пов’язана із стрімким зростанням кількості транспортних засобів на дорогах та відповідним зростанням помилок водіїв, що призводять до ДТП. Проведено аналіз існуючих робіт, із їх перевагами та недоліками. Головною проблемою сучасних моделей є неможливість швидкого виконання точних прогнозів для використання у режимі реального часу.

В другому розділі розглянуті теоретичні відомості про основні елементи архітектур моделей прогнозування траєкторій – згорткові нейронні мережі, рекурентні нейронні мережі, моделі із механізмом уваги та Трансформери, і графові нейронні мережі.

В третьому розділі наведено деталі архітектури двох запропонованих моделей – заснованій на Трансформерах та на графових нейронних мережах. Обидві моделі використовують сценоцентричне представлення вхідних даних для забезпечення швидких прогнозів.

В четвертому розділі розглянуто деталі практичної реалізації та тестування запропонованих моделей. Описано набір даних та метрик, які використовувались для оцінки точності, обґрунтовано вибір мови програмування та бібліотек. Наведено результати оцінки точності, які показують, що запропонована швидка модель заснована на Трансформерах демонструє результати порівнянні із повільнішими агентоцентричними аналогами.

У п'ятому розділі проведено функціонально-вартісний аналіз розробленого програмного продукту.

Побудовані моделі демонструють високий рівень точності при значному прирості у швидкості в порівнянні із існуючими агентоцентричними аналогами та можуть бути використані для подальших наукових досліджень.

Сценоцентричні моделі можна розвивати у подальшому. Одним із напрямків розвитку може слугувати використання методу дистиляції знань (англ. knowledge distillation), де агентоцентричні моделі виступають в якості вчителя.

ПЕРЕЛІК ПОСИЛАНЬ

1. Jean-Pierre Bardou, Jean-Jacques Chanaron, Patrick Fridenson, and James M. Laux. The Automobile Revolution: The Impact of an Industry. *Chapel Hill, The University of North Carolina Press, pp. xvi + 335,* 1982.
2. UK Department for Transport. Research on the impacts of connected and autonomous vehicles (CAVs) on traffic flow: Summary report. *URL: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/530091/impacts-of-connected-and-autonomous-vehicles-on-traffic-flow-summary-report.pdf* (дата звернення: 07.09.2023)
3. Phantom Auto" to Be Operated Here. The Free-Lance Star, 17 June 1932 *URL: https://news.google.com/newspapers?id=PthNAAAAIBAJ&pg=6442,3879017* (дата звернення: 12.09.2023).
4. Jameson M. Wetmore Consortium for Science, Policy & Outcomes Arizona State University. Driving the Dream: The History and Motivations Behind 60 Years of Automated Highway Systems in America. *Automotive History Review,* 2003, pp. 4-19.
5. Daniel Bartz. Autonomous Cars Will Make Us Safer.c*URL: https://www.wired.com/2009/11/autonomous-cars/* (дата звернення: 12.09.2023)
6. J. Schmidhuber. Prof. Schmidhuber's highlights of robot car history. *URL:* http://people.idsia.ch/~juergen/robotcars.html (дата звернення: 12.09.2023)
7. Pomerleau, Dean. ALVINN: an autonomous land vehicle in a neural network. *Advances in Neural Information Processing Systems*, 1989.

8. Lefevre, Stephanie & Vasquez, Dizan & Laugier, Christian. A survey on motion prediction and risk assessment for intelligent vehicles. *Robomech Journal*, 2014.
9. Sajjad Mozaffari, Omar Y. Al-Jarrah, Mehrdad Dianati, Paul Jennings, and Alexandros Mouzakitis. Deep Learning-based Vehicle Behaviour Prediction for Autonomous Driving Applications: a Review, 2020. *arXiv:1912.11676v2*. DOI: https://doi.org/10.48550/arXiv.1912.11676
10. Ammoun, S., Nashashibi, F. Real time trajectory prediction for collision risk estimation between vehicles. *Proc. IEEE Intelligent Computer Communication and Processing, p. 417–422.*
11. Kaempchen, N., Weiss, K., Schaefer, M., Dietmayer, K.C.J. IMM object tracking for high dynamic driving maneuvers. *Proc. IEEE Intelligent Vehicles Symposium, pp. 825-830,* 2004.
12. Rajamani, R.: Vehicle Dynamics and Control. Springer, New York City, 2006.
13. Brännström, M., Coelingh, E., Sjöberg, J. Model-based threat assessment for avoiding arbitrary vehicle collisions. *IEEE Transactions on Intelligent Transportation Systems 11(3), 658–669*, 2010.
14. Vasquez, D., Fraichard, T. Motion prediction for moving objects: a statistical approach. Proc. *IEEE International Conference on Robotics and Automation, vol. 4, pp. 3931–3936*, 2004.
15. Hermes, C., Wohler, C., Schenk, K., Kummert, F. Long-term vehicle motion prediction. *Proc. IEEE Intelligent Vehicles Symposium, pp. 652–657*, 2009.
16. Greene, D., Liu, J., Reich, J., Hirokawa, Y., Shinagawa, A., Ito, H., Mikami, T. An efficient computational architecture for a collision early-warning system for vehicles, pedestrians, and bicyclists. *IEEE Transactions on Intelligent Transportation Systems 11(4), 1–12*, 2010.

17. Tamke, A., Dang, T., Breuel, G. A flexible method for criticality assessment in driver assistance systems. *Proc. IEEE Intelligent Vehicles Symposium, pp. 697–702,* 2011.
18. Lefèvre, S., Laugier, C., Ibañez-Guzmán, J. Risk assessment at road intersections: comparing intention and expectation. *Proc. IEEE Intelligent Vehicles Symposium, pp. 165–171*, 2012.
19. D. J. Phillips, T. A. Wheeler, and M. J. Kochenderfer. Generalizable intention prediction of human drivers at intersections. *IEEE Intelligent Vehicles Symposium (IV), pp. 1665–1670*, 2017
20. A. Zyner, S. Worrall, and E. Nebot. A recurrent neural network solution for predicting driver intention at unsignalized intersections. *IEEE Robotics and Automation Letters, vol. 3, no. 3, pp. 1759–1764*, 2018.
21. A. Zyner, S. Worrall, J. Ward, and E. Nebot. Long short term memory for driver intent prediction. *IEEE Intelligent Vehicles Symposium (IV), pp. 1484–1489,* 2017.
22. A. Zyner, S. Worrall, and E. M. Nebot. Naturalistic driver intention and path prediction using recurrent neural networks. *CoRR, vol. abs/1807.09995*, 2018.
23. L. Xin, P. Wang, C. Chan, J. Chen, S. E. Li, and B. Cheng. Intention-aware long horizon trajectory prediction of surrounding vehicles using dual lstm networks. *21st International Conference on Intelligent Transportation Systems (ITSC), pp. 1441–1446*, 2018.
24. S. H. Park, B. Kim, C. M. Kang, C. C. Chung, and J. W. Choi. Sequence-to-sequence prediction of vehicle trajectory via lstm encoder- decoder architecture. *IEEE Intelligent Vehicles Symposium (IV), pp. 1672–1678*, 2018.
25. N. Deo and M. M. Trivedi. Multi-modal trajectory prediction of surrounding vehicles with maneuver based lstms. *IEEE Intelligent Vehicles Symposium (IV), pp. 1179–1184,* 2018.

26. S. Dai, L. Li, and Z. Li. Modeling vehicle interactions via modified lstm models for trajectory prediction. *IEEE Access, vol. 7, pp. 38 287–38 296*, 2019.
27. Hu, W. Zhan, and M. Tomizuka. Probabilistic prediction of vehicle semantic intention and motion. *IEEE Intelligent Vehicles Symposium (IV), pp. 307–313*, 2018.
28. W. Ding, J. Chen, and S. Shen. Predicting vehicle behaviors over an extended horizon using behavior interaction network. *CoRR, vol. abs/1903.00848*, 2019.
29. F. Altch and A. de La Fortelle. An lstm network for highway trajectory prediction. *IEEE 20th International Conference on Intelligent Transportation Systems (ITSC), pp. 353–359*, 2017.
30. Y. Ma, X. Zhu, S. Zhang, R. Yang, W. Wang, and D. Manocha. Trafficpredict: Trajectory prediction for heterogeneous traffic-agents. *Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 6120–6127,* 2019.
31. D.Lee, Y.P.Kwon, S.McMains, and J.K.Hedrick. Convolution neural network-based lane change intention prediction of surrounding vehicles for acc. *IEEE 20th International Conference on Intelligent Transportation Systems (ITSC), pp. 1–6,* 2017.
32. H. Cui, V. Radosavljevic, F. Chou, T. Lin, T. Nguyen, T. Huang, J. Schneider, and N. Djuric. Multimodal trajectory predictions for autonomous driving using deep convolutional networks. *CoRR, vol. abs/1809.10732*, 2018.
33. M. T. L. A. K. Frederik Diehl, Thomas Brunner. Graph neural networks for modelling traffic participant interaction. *IEEE Intelligent Vehicles Symposium 2019,* 2019.

34. N. Djuric, V. Radosavljevic, H. Cui, T. Nguyen, F. Chou, T. Lin, and J. Schneider. Motion prediction of traffic actors for autonomous driving using deep convolutional networks. *CoRR, vol. abs/1808.05819*, 2018.
35. N. Deo and M. M. Trivedi. Convolutional social pooling for vehicle trajectory prediction. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops,* 2018.
36. T. Zhao, Y. Xu, M. Monfort, W. Choi, C. Baker, Y. Zhao, Y. Wang, and Y. N. Wu. Multi-agent tensor fusion for contextual trajectory prediction. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2019.
37. S. Hoermann, M. Bach, and K. Dietmayer. Dynamic occupancy grid prediction for urban autonomous driving: A deep learning approach with fully automatic labeling. *IEEE International Conference on Robotics and Automation (ICRA), pp. 2056–2063*, 2018.
38. M. Schreiber, S. Hoermann, and K. Dietmayer. Long-term occupancy grid prediction using recurrent neural networks. *CoRR, vol. abs/1809.03782*, 2018.
39. W. Luo, B. Yang, and R. Urtasun. Fast and furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* June 2018.
40. S. Casas, W. Luo, and R. Urtasun. Intentnet: Learning to predict intention from raw sensor data. *Proceedings of The 2nd Conference on Robot Learning,* 2018.
41. X. Li, X. Ying, and M. C. Chuah. Grip: Graph-based interaction-aware trajectory prediction. *2019 IEEE Intelligent Transportation Systems Conference (ITSC), Auckland, New Zealand, pp. 3960-3966,* 2019.

42. C. Ju, Z. Wang, C. Long, X. Zhang, G. Cong, and D. E. Chang. Interaction-aware Kalman Neural Networks for Trajectory Prediction. *2020 IEEE Intelligent Vehicles Symposium (IV), pp. 1793-1800*, 2019.
43. N. Lee, W. Choi, P. Vernaza, C. B. Choy, P. H. S. Torr, and M. Chandraker. Desire: Distant future prediction in dynamic scenes with interacting agents. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2017.
44. Agrim Gupta, Justin Johnson et al. Social gan: Socially acceptable trajectories with generative adversarial networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2255–2264*, 2018.
45. Zikang Zhou, Jianping Wang, Yung-Hui Li, Yu-Kai Huang. Query-Centric Trajectory Prediction. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023
46. Benjamin Wilson and William Qi and Tanmay Agarwal et al. Argoverse 2: Next Generation Datasets for Self-Driving Perception and Forecasting. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS Datasets and Benchmarks 2021),* 2021.
47. Andreas Ess, Bastian Leibe, and Luc Van Gool. Depth and appearance for mobile scene analysis. *ICCV, pp. 1–8*, 2007.
48. L. Leal-Taixe, M. Fenzi, A. Kuznetsova, B. Rosenhahn, and S. Savarese. Learning an image-based motion context for multiple people tracking. *CVPR, pp. 3542–3549*, 2014.
49. Ke Guo, Wenxi Liu, Jia Pan. End-to-End Trajectory Distribution Prediction Based on Occupancy Grid Maps. *CVPR*, 2022.
50. Bock, Julian and Krajewski, Robert et al. The inD Dataset: A Drone Dataset of Naturalistic Road User Trajectories at German Intersections. *2020 IEEE Intelligent Vehicles Symposium (IV)*, 2020.

51. Hochreiter, Sepp & Schmidhuber, Jürgen. Long Short-term Memory. *Neural computation. 9. 1735-80,* 1997.
52. Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *NIPS 2014 Deep Learning and Representation Learning Workshop,* 2014
53. Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *ICLR 2015,* 2015.
54. Ashish Vaswani, Noam Shazeer et al. Attention Is All You Need. 2017. *arXiv:1706.03762,* DOI: https://doi.org/10.48550/arXiv.1706.03762
55. Minh-Thang Luong, Hieu Pham, Christopher D. Manning. Effective Approaches to Attention-based Neural Machine Translation. *EMNLP 2015,* 2015.
56. J. Ho, N. Kalchbrenner, D. Weissenborn, and T. Salimans. Axial attention in multidimensional transformers. *arXiv:1912.12180,* 2019. DOI: https://doi.org/10.48550/arXiv.1912.12180
57. Petar Veličković. Message passing all the way up. *arXiv:2202.11097,* 2022. DOI: https://doi.org/10.48550/arXiv.2202.11097
58. Thomas N. Kipf, Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv:1609.02907*, 2016. DOI: https://doi.org/10.48550/arXiv.1609.02907
59. Amy Jang, Christy, Luca Bergamini et al. Lyft Motion Prediction for Autonomous Vehicles. *Kaggle. URL: https://kaggle.com/competitions/lyft-motion-prediction-autonomous-vehicles* (дата звернення: 12.09.2023), 2020
60. Scott Ettinger, Shuyang Cheng, Benjamin Caine et al. Large Scale Interactive Motion Forecasting for Autonomous Driving: The Waymo Open Motion Dataset. *CoRR, abs/2104.10133*, 2021.

61. Andy Su, Bertrand Douillard et al. Narrowing the coordinate-frame gap in behavior prediction models: Distillation for efficient and accurate scene-centric motion forecasting. *2022 International Conference on Robotics and Automation (ICRA),* 2022.
62. Yuning Chai, Benjamin Sapp, Mayank Bansal, Dragomir Anguelov. MultiPath: Multiple Probabilistic Anchor Trajectory Hypotheses for Behavior Prediction. *Proceedings of the Conference on Robot Learning, PMLR 100:86-99,* 2020
63. Ming Liang, Bin Yang, Rui Hu et al. Learning Lane Graph Representations for Motion Forecasting. *ECCV 2020,* 2020.
64. van Dyck LE, Kwitt R, Denzler SJ, Gruber WR. Comparing Object Recognition in Humans and Deep Convolutional Neural Networks-An Eye Tracking Study. *Front Neurosci. 2021 Oct 6,* 2021.
65. Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, Backpropagation Applied to Handwritten Zip Code Recognition. *AT&T Bell Laboratories*, 1989
66. Zhou, Zikang and Ye, Luyao and Wang, Jianping and Wu, Kui and Lu, Kejie. HiVT: Hierarchical Vector Transformer for Multi-Agent Motion Prediction. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.
67. Ajayi O, Gangopadhyay A, Erbacher RF, Bursat C. Developing Cross-Domain Host-Based Intrusion Detection. *Electronics. 2022; 11(21):3631,* 2022

ДОДАТОК А ЛІСТИНГ ПРОГРАМНОГО ПРОДУКТУ

```
import math
import os
import pickle
import shutil
import sys
from pathlib import Path
from urllib import request

import numpy as np
import pandas as pd
import torch
import torch_cluster
from av2.geometry.interpolate import compute_midpoint_line
from av2.map.map_api import ArgoverseStaticMap
from torch_geometric.data import Dataset
from torch_geometric.data import extract_tar
from tqdm.notebook import tqdm


def safe_list_index(ls, elem):
    try:
        return ls.index(elem)
    except ValueError:
        return None


def get_files_from_root(root, file_type='dir', file_ext=None):
    if not os.path.isdir(root):
        return []

    if file_type == 'dir':
        return [name for name in os.listdir(root) if os.path.isdir(os.path.join(root, name))]
    elif file_type == 'file':
        return [name for name in os.listdir(root) if
                os.path.isfile(os.path.join(root, name)) and name.endswith(file_ext)]
    else:
        raise ValueError(f"File type {file_type} is not valid.")


def side_to_directed_lineseg(
        query_point,
        start_point,
        end_point):
    cond = ((end_point[0] - start_point[0]) * (query_point[1] - start_point[1]) -
            (end_point[1] - start_point[1]) * (query_point[0] - start_point[0]))
    if cond > 0:
        return 'LEFT'
    elif cond < 0:
        return 'RIGHT'
```

```
    else:
        return 'CENTER'


def get_n_paths(scenario_dir, n=10, file_ext='parquet'):
    scenarios = []
    n = float('inf') if n is None else n
    for i, path in enumerate(Path(scenario_dir).rglob(f'*.{file_ext}')):
        if i > n:
            break
        scenarios.append(path)

    return scenarios


def wrap_angle(
        angle,
        min_val: float = -math.pi,
        max_val: float = math.pi):
    return min_val + (angle + max_val) % (max_val - min_val)


def rotate_coords(coords, thetas, origin, origin_theta):
    rotation_matrix = torch.tensor([[torch.cos(origin_theta), -torch.sin(origin_theta)],
                                    [torch.sin(origin_theta), torch.cos(origin_theta)]])
    relative_coords = torch.matmul((coords - origin), rotation_matrix)
    relative_angles = wrap_angle(thetas - origin_theta)

    return relative_coords, relative_angles


class Argoverse2Dataset(Dataset):
    """
    Argoverse2 motion forecasting dataset. Long version.
    """

    def __init__(self, root, raw_dir=None, processed_dir=None, split='train', download_data=False,
                 num_historical_steps=50,
                 num_future_steps=60, continuous_agent_validity=True, max_agents=32,
max_points=15_000,
                 transform=None, pre_transform=None):

        self.root = os.path.expanduser(os.path.normpath(root))

        if split in ('train', 'val', 'test'):
            pass
            # self.root = os.path.join(self.root, split)
        else:
            raise ValueError(f"Split {split} is not valid.")
        self.split = split

        os.makedirs(self.root, exist_ok=True)

        if raw_dir is not None:
            self._raw_dir = os.path.expanduser(os.path.normpath(raw_dir))
```

```
        else:
            self._raw_dir = os.path.join(self.root, split, 'raw')
        self._raw_file_names = get_files_from_root(self._raw_dir, file_type='dir')

        if processed_dir is not None:
            self._processed_dir = os.path.expanduser(os.path.normpath(processed_dir))
        else:
            self._processed_dir = os.path.join(self.root, split, 'processed')

        self._processed_file_names = get_files_from_root(self._processed_dir,
                                                         file_type='file',
                                                         file_ext=('pickle', 'pkl'))

        self.transform = transform
        self.dim = 2
        self.num_historical_steps = num_historical_steps
        self.num_future_steps = num_future_steps
        self.num_steps = num_historical_steps + num_future_steps
        self.continuous_agent_validity = continuous_agent_validity
        self.max_agents = max_agents
        self.max_points = max_points
        self.download_data = download_data

        self._url = f'https://s3.amazonaws.com/argoverse/datasets/av2/tars/motion-forecasting/{split}.tar'

        self._num_samples = {
            'train': 4250,
            'val': 750,
            'test': 0,
        }[split]
        self._agent_types = ['vehicle', 'pedestrian', 'motorcyclist', 'cyclist', 'bus', 'static', 'background',
                             'construction', 'riderless_bicycle', 'unknown']
        self._agent_categories = ['TRACK_FRAGMENT', 'UNSCORED_TRACK',
'SCORED_TRACK', 'FOCAL_TRACK']
        self._polygon_types = ['VEHICLE', 'BIKE', 'BUS', 'PEDESTRIAN']
        self._polygon_is_intersections = [True, False, None]
        self._point_types = ['DASH_SOLID_YELLOW', 'DASH_SOLID_WHITE',
'DASHED_WHITE', 'DASHED_YELLOW',
                             'DOUBLE_SOLID_YELLOW', 'DOUBLE_SOLID_WHITE',
'DOUBLE_DASH_YELLOW', 'DOUBLE_DASH_WHITE',
                             'SOLID_YELLOW', 'SOLID_WHITE', 'SOLID_DASH_WHITE',
'SOLID_DASH_YELLOW', 'SOLID_BLUE',
                             'NONE', 'UNKNOWN', 'CROSSWALK', 'CENTERLINE']
        self._point_sides = ['LEFT', 'RIGHT', 'CENTER']
        self._polygon_to_polygon_types = ['NONE', 'PRED', 'SUCC', 'LEFT', 'RIGHT']

        super(Argoverse2Dataset, self).__init__(root, transform, pre_transform, pre_filter=None)

    def process(self, skip_processed=False):
        for raw_file_name in tqdm(self._raw_file_names):
            raw_path = os.path.join(self._raw_dir, raw_file_name)
            processed_file_path = os.path.join(self._processed_dir, f'{raw_file_name}.pkl')
            if skip_processed and os.path.exists(processed_file_path):
                continue
```

```
        agent_df = pd.read_parquet(get_n_paths(raw_path, n=1, file_ext='parquet')[0])
        map_api = ArgoverseStaticMap.from_json(get_n_paths(raw_path, n=1, file_ext='json')[0])

        map_features_list = self.get_map_features(map_api)
        agent_features_list = self.get_agent_features(agent_df)

        map_features, map_valid_mask = self.pad_or_cut_map_features(map_features_list)
        agent_features, agent_valid_mask, predict_mask, agent_category =
self.pad_or_cut_agent_features(
            agent_features_list)

        map_features, agent_features = self.rotate_scene(map_features, agent_features,
agent_category)

        scene_data = {'map_features': map_features, 'map_valid_mask': map_valid_mask,
                'agent_features': agent_features, 'agent_valid_mask': agent_valid_mask,
                'predict_mask': predict_mask}

        with open(processed_file_path, 'wb') as handle:
            pickle.dump(scene_data, handle, protocol=pickle.HIGHEST_PROTOCOL)

    def rotate_scene(self, map_features, agent_features, agent_categories):
        agent_coordinates = agent_features[..., :2]
        agent_headings = agent_features[..., 2]

        focal_agent = agent_categories == 3
        origin = agent_coordinates[focal_agent][0, self.num_historical_steps - 1]
        origin_heading = agent_headings[focal_agent][0, self.num_historical_steps - 1]

        agent_features[..., :2], agent_features[..., 2] = rotate_coords(agent_coordinates, agent_headings,
                                                         origin, origin_heading)

        map_features[:, 2:4], map_features[:, 4] = rotate_coords(map_features[:, 2:4], map_features[:, 4],
                                                 origin, origin_heading)

        return map_features, agent_features

    def pad_or_cut_map_features(self, map_features_list):
        map_features_expanded = [torch.unsqueeze(torch.cat(x, dim=0), -1) for x in map_features_list]
        map_features_expanded[2] = torch.squeeze(map_features_expanded[2])

        map_features = torch.cat(map_features_expanded, dim=-1)

        map_valid_mask = torch.ones(self.max_points, dtype=torch.bool)
        if map_features.shape[0] > self.max_points:
            # randomly sample max_points points
            spacing = map_features.shape[0] // self.max_points
            map_features = map_features[::spacing]
            indices = np.random.choice(map_features.shape[0], self.max_points, replace=False)
            map_features = map_features[indices]
        elif map_features.shape[0] < self.max_points:
            # pad with zeros
            n_zeroes = self.max_points - map_features.shape[0]
            zero_padding = torch.zeros(n_zeroes, map_features.shape[1])
            map_features = torch.cat([map_features, zero_padding], dim=0)
```

```
            map_valid_mask[-n_zeroes:] = False

        return map_features, map_valid_mask

    def pad_or_cut_agent_features(self, agent_features_list):
        (num_agents, av_idx, agent_valid_mask, predict_mask, agent_id, agent_type,
         agent_category, position, heading, velocity) = agent_features_list

        # agent_type_one_hot = torch.nn.functional.one_hot(agent_type.long(), num_classes=10)

        agent_type = agent_type.unsqueeze(1).repeat(1, self.num_steps).unsqueeze(-1)
        agent_features = torch.cat([position, heading.unsqueeze(-1), velocity, agent_type], dim=-1)

        if len(agent_features) < self.max_agents:
            n_zeroes = self.max_agents - len(agent_features)
            zero_padding = torch.zeros(n_zeroes, agent_features.shape[1], agent_features.shape[2])
            agent_features = torch.cat([agent_features, zero_padding], dim=0)
            agent_valid_mask = torch.cat(
                [agent_valid_mask, torch.zeros(n_zeroes, *agent_valid_mask.shape[1:], dtype=torch.bool)],
dim=0)
            predict_mask = torch.cat([predict_mask, torch.zeros(n_zeroes, *predict_mask.shape[1:],
dtype=torch.bool)],
                                     dim=0)
            agent_category = torch.cat([agent_category, torch.zeros(n_zeroes, dtype=torch.uint8)], dim=0)
        elif len(agent_features) > self.max_agents:
            # take the closest max_agents agents
            last_positions = position[:, self.num_historical_steps - 1, :]
            focal_agent_position = last_positions[agent_category == 3]
            knn_indices = torch_cluster.knn(last_positions, focal_agent_position, k=self.max_agents)[1]
            agent_features = agent_features[knn_indices]
            agent_valid_mask = agent_valid_mask[knn_indices]
            predict_mask = predict_mask[knn_indices]
            agent_category = agent_category[knn_indices]

        return agent_features, agent_valid_mask, predict_mask, agent_category

    def get_agent_features(self, df):
        # filter out agents that are unseen during the historical time steps
        historical_df = df[df['timestep'] < self.num_historical_steps]
        agent_ids = list(historical_df['track_id'].unique())
        df = df[df['track_id'].isin(agent_ids)]

        num_agents = len(agent_ids)
        av_idx = agent_ids.index('AV')

        # initialization
        valid_mask = torch.zeros(num_agents, self.num_steps, dtype=torch.bool)
        current_valid_mask = torch.zeros(num_agents, dtype=torch.bool)
        predict_mask = torch.zeros(num_agents, self.num_steps, dtype=torch.bool)
        agent_id = [None] * num_agents
        agent_type = torch.zeros(num_agents, dtype=torch.uint8)
        agent_category = torch.zeros(num_agents, dtype=torch.uint8)
        position = torch.zeros(num_agents, self.num_steps, self.dim, dtype=torch.float)
        heading = torch.zeros(num_agents, self.num_steps, dtype=torch.float)
        velocity = torch.zeros(num_agents, self.num_steps, self.dim, dtype=torch.float)
```

```
        for track_id, track_df in df.groupby('track_id'):
            agent_idx = agent_ids.index(track_id)
            agent_steps = track_df['timestep'].values

            valid_mask[agent_idx, agent_steps] = True
            current_valid_mask[agent_idx] = valid_mask[agent_idx, self.num_historical_steps - 1]
            predict_mask[agent_idx, agent_steps] = True
            if self.continuous_agent_validity:
                valid_mask[agent_idx, 1: self.num_historical_steps] = (
                        valid_mask[agent_idx, :self.num_historical_steps - 1] &
                        valid_mask[agent_idx, 1: self.num_historical_steps])
                valid_mask[agent_idx, 0] = False
            predict_mask[agent_idx, :self.num_historical_steps] = False
            if not current_valid_mask[agent_idx]:
                predict_mask[agent_idx, self.num_historical_steps:] = False

            agent_id[agent_idx] = track_id
            agent_type[agent_idx] = self._agent_types.index(track_df['object_type'].values[0])
            agent_category[agent_idx] = track_df['object_category'].values[0]
            position[agent_idx, agent_steps, :2] =
torch.from_numpy(np.stack([track_df['position_x'].values,
                                                       track_df['position_y'].values],
                                                      axis=-1)).float()
            heading[agent_idx, agent_steps] = torch.from_numpy(track_df['heading'].values).float()
            velocity[agent_idx, agent_steps, :2] =
torch.from_numpy(np.stack([track_df['velocity_x'].values,
                                                       track_df['velocity_y'].values],
                                                      axis=-1)).float()

        if self.split == 'test':
            predict_mask[current_valid_mask
                         | (agent_category == 2)
                         | (agent_category == 3), self.num_historical_steps:] = True

        return (num_agents, av_idx, valid_mask, predict_mask, agent_id, agent_type,
                agent_category, position, heading, velocity)

    def get_map_features(self, map_api):
        lane_segment_ids = map_api.get_scenario_lane_segment_ids()
        cross_walk_ids = list(map_api.vector_pedestrian_crossings.keys())
        polygon_ids = lane_segment_ids + cross_walk_ids
        num_polygons = len(lane_segment_ids) + len(cross_walk_ids) * 2

        polygon_type = [None] * num_polygons
        polygon_is_intersection = [None] * num_polygons
        point_position = [None] * num_polygons
        point_orientation = [None] * num_polygons
        point_magnitude = [None] * num_polygons
        point_type = [None] * num_polygons
        point_side = [None] * num_polygons

        for lane_segment in map_api.get_scenario_lane_segments():
            lane_segment_idx = polygon_ids.index(lane_segment.id)
```

```
        centerline =
torch.from_numpy(map_api.get_lane_segment_centerline(lane_segment.id)).float()

        left_boundary = torch.from_numpy(lane_segment.left_lane_boundary.xyz).float()
        right_boundary = torch.from_numpy(lane_segment.right_lane_boundary.xyz).float()
        point_position[lane_segment_idx] = torch.cat([left_boundary[:-1, :self.dim],
                                        right_boundary[:-1, :self.dim],
                                        centerline[:-1, :self.dim]], dim=0)
        left_vectors = left_boundary[1:] - left_boundary[:-1]
        right_vectors = right_boundary[1:] - right_boundary[:-1]
        center_vectors = centerline[1:] - centerline[:-1]
        total_vectors = len(left_vectors) + len(right_vectors) + len(center_vectors)

        polygon_type[lane_segment_idx] = torch.full((total_vectors,),
                                       self._polygon_types.index(lane_segment.lane_type.value),
                                       dtype=torch.uint8)

        polygon_is_intersection[lane_segment_idx] = torch.full((total_vectors,),
                                            self._polygon_is_intersections.index(
                                               lane_segment.is_intersection),
                                            dtype=torch.uint8)

        point_orientation[lane_segment_idx] = torch.cat([torch.atan2(left_vectors[:, 1], left_vectors[:,
0]),
                                         torch.atan2(right_vectors[:, 1], right_vectors[:, 0]),
                                         torch.atan2(center_vectors[:, 1], center_vectors[:, 0])],
                                        dim=0)
        point_magnitude[lane_segment_idx] = torch.norm(torch.cat([left_vectors[:, :2],
                                             right_vectors[:, :2],
                                             center_vectors[:, :2]], dim=0), p=2, dim=-1)
        left_type = self._point_types.index(lane_segment.left_mark_type.value)
        right_type = self._point_types.index(lane_segment.right_mark_type.value)
        center_type = self._point_types.index('CENTERLINE')

        point_type[lane_segment_idx] = torch.cat(
           [torch.full((len(left_vectors),), left_type, dtype=torch.uint8),
            torch.full((len(right_vectors),), right_type, dtype=torch.uint8),
            torch.full((len(center_vectors),), center_type, dtype=torch.uint8)], dim=0)

        point_side[lane_segment_idx] = torch.cat(
           [torch.full((len(left_vectors),), self._point_sides.index('LEFT'), dtype=torch.uint8),
            torch.full((len(right_vectors),), self._point_sides.index('RIGHT'), dtype=torch.uint8),
            torch.full((len(center_vectors),), self._point_sides.index('CENTER'), dtype=torch.uint8)],
dim=0)

      for crosswalk in map_api.get_scenario_ped_crossings():
        crosswalk_idx = polygon_ids.index(crosswalk.id)
        edge1 = torch.from_numpy(crosswalk.edge1.xyz).float()
        edge2 = torch.from_numpy(crosswalk.edge2.xyz).float()
        start_position = (edge1[0] + edge2[0]) / 2
        end_position = (edge1[-1] + edge2[-1]) / 2

        if side_to_directed_lineseg((edge1[0] + edge1[-1]) / 2, start_position, end_position) ==
'LEFT':
            left_boundary = edge1
```

```
            right_boundary = edge2
        else:
            left_boundary = edge2
            right_boundary = edge1
        num_centerline_points = math.ceil(torch.norm(end_position - start_position, p=2, dim=-
1).item() / 2.0) + 1
        centerline = torch.from_numpy(
            compute_midpoint_line(left_ln_boundary=left_boundary.numpy(),
                                  right_ln_boundary=right_boundary.numpy(),
                                  num_interp_pts=int(num_centerline_points))[0]).float()

        point_position[crosswalk_idx] = torch.cat([left_boundary[:-1, :self.dim],
                                                   right_boundary[:-1, :self.dim],
                                                   centerline[:-1, :self.dim]], dim=0)
        point_position[crosswalk_idx + len(cross_walk_ids)] = torch.cat(
            [right_boundary.flip(dims=[0])[:-1, :self.dim],
             left_boundary.flip(dims=[0])[:-1, :self.dim],
             centerline.flip(dims=[0])[:-1, :self.dim]], dim=0)
        left_vectors = left_boundary[1:] - left_boundary[:-1]
        right_vectors = right_boundary[1:] - right_boundary[:-1]
        center_vectors = centerline[1:] - centerline[:-1]
        total_vectors = len(left_vectors) + len(right_vectors) + len(center_vectors)

        polygon_type[crosswalk_idx] = torch.full((total_vectors,),
                                                 self._polygon_types.index('PEDESTRIAN'),
                                                 dtype=torch.uint8)
        polygon_type[crosswalk_idx + len(cross_walk_ids)] = torch.full((total_vectors,),
                                                                       self._polygon_types.index('PEDESTRIAN'),
                                                                       dtype=torch.uint8)
        polygon_is_intersection[crosswalk_idx] = torch.full((total_vectors,),
                                                            self._polygon_is_intersections.index(None),
                                                            dtype=torch.uint8)
        polygon_is_intersection[crosswalk_idx + len(cross_walk_ids)] = torch.full((total_vectors,),
                                                                                  self._polygon_is_intersections.index(
                                                                                      None), dtype=torch.uint8)
        point_orientation[crosswalk_idx] = torch.cat(
            [torch.atan2(left_vectors[:, 1], left_vectors[:, 0]),
             torch.atan2(right_vectors[:, 1], right_vectors[:, 0]),
             torch.atan2(center_vectors[:, 1], center_vectors[:, 0])], dim=0)
        point_orientation[crosswalk_idx + len(cross_walk_ids)] = torch.cat(
            [torch.atan2(-right_vectors.flip(dims=[0])[:, 1], -right_vectors.flip(dims=[0])[:, 0]),
             torch.atan2(-left_vectors.flip(dims=[0])[:, 1], -left_vectors.flip(dims=[0])[:, 0]),
             torch.atan2(-center_vectors.flip(dims=[0])[:, 1], -center_vectors.flip(dims=[0])[:, 0])],
dim=0)
        point_magnitude[crosswalk_idx] = torch.norm(torch.cat([left_vectors[:, :2],
                                                               right_vectors[:, :2],
                                                               center_vectors[:, :2]], dim=0), p=2, dim=-1)
        point_magnitude[crosswalk_idx + len(cross_walk_ids)] = torch.norm(
            torch.cat([-right_vectors.flip(dims=[0])[:, :2],
                       -left_vectors.flip(dims=[0])[:, :2],
                       -center_vectors.flip(dims=[0])[:, :2]], dim=0), p=2, dim=-1)

        crosswalk_type = self._point_types.index('CROSSWALK')
        center_type = self._point_types.index('CENTERLINE')
        point_type[crosswalk_idx] = torch.cat([
```

```
            torch.full((len(left_vectors),), crosswalk_type, dtype=torch.uint8),
            torch.full((len(right_vectors),), crosswalk_type, dtype=torch.uint8),
            torch.full((len(center_vectors),), center_type, dtype=torch.uint8)], dim=0)
        point_type[crosswalk_idx + len(cross_walk_ids)] = torch.cat(
            [torch.full((len(right_vectors),), crosswalk_type, dtype=torch.uint8),
             torch.full((len(left_vectors),), crosswalk_type, dtype=torch.uint8),
             torch.full((len(center_vectors),), center_type, dtype=torch.uint8)], dim=0)
        point_side[crosswalk_idx] = torch.cat(
            [torch.full((len(left_vectors),), self._point_sides.index('LEFT'), dtype=torch.uint8),
             torch.full((len(right_vectors),), self._point_sides.index('RIGHT'), dtype=torch.uint8),
             torch.full((len(center_vectors),), self._point_sides.index('CENTER'), dtype=torch.uint8)],
dim=0)
        point_side[crosswalk_idx + len(cross_walk_ids)] = torch.cat(
            [torch.full((len(right_vectors),), self._point_sides.index('LEFT'), dtype=torch.uint8),
             torch.full((len(left_vectors),), self._point_sides.index('RIGHT'), dtype=torch.uint8),
             torch.full((len(center_vectors),), self._point_sides.index('CENTER'), dtype=torch.uint8)],
dim=0)

    return polygon_type, polygon_is_intersection, point_position, point_orientation,
point_magnitude, point_type, point_side

def len(self):
    return self._num_samples

@property
def raw_dir(self) -> str:
    return self._raw_dir

@property
def processed_dir(self) -> str:
    return self._processed_dir

@property
def raw_file_names(self):
    return self._raw_file_names

@property
def processed_file_names(self):
    return self._processed_file_names

def get(self, idx):
    with open(self.processed_paths[idx], 'rb') as handle:
        return pickle.load(handle)

def _process(self):
    if os.path.isdir(self.processed_dir) and len(self.processed_file_names) == len(self):
        return

    print('Processing...')
    if os.path.isdir(self.processed_dir):
        os.rmdir(self.processed_dir)

    os.makedirs(self.processed_dir, exist_ok=True)
    self._processed_file_names = [f'{raw_file_name}.pkl' for raw_file_name in self.raw_file_names]
    self.process()
```

```
        print('Done!')

    def _download(self):
        if not self.download_data:
            return
        self.download()

    def download(self) -> None:
        if not os.path.isfile(os.path.join(self.root, f'{self.split}.tar')):
            print(f'Downloading {self._url}', file=sys.stderr)
            request.urlretrieve(self._url, os.path.join(self.root, f'{self.split}.tar'))
        if os.path.isdir(os.path.join(self.root, self.split)):
            shutil.rmtree(os.path.join(self.root, self.split))
        if os.path.isdir(self.raw_dir):
            shutil.rmtree(self.raw_dir)
        os.makedirs(self.raw_dir)
        extract_tar(path=os.path.join(self.root, f'{self.split}.tar'), folder=self.raw_dir, mode='r')
        self._raw_file_names = [name for name in os.listdir(os.path.join(self.raw_dir, self.split)) if
                                os.path.isdir(os.path.join(self.raw_dir, self.split, name))]
        for raw_file_name in self.raw_file_names:
            shutil.move(os.path.join(self.raw_dir, self.split, raw_file_name), self.raw_dir)
        os.rmdir(os.path.join(self.raw_dir, self.split))


import math

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


def apply_flat_mask(x, mask, fill_value=-float('inf'), mask_type='both'):
    if mask_type == 'both':
        x[:, mask] = fill_value
        x.transpose(-2, -1)[:, mask] = fill_value
    elif mask_type == 'row':
        x[:, mask] = fill_value
    elif mask_type == 'col':
        x.transpose(-2, -1)[:, mask] = fill_value
    else:
        raise ValueError(f'Unknown type: {mask_type}')

    return x


class FactorizedMultiHeadAttention(nn.Module):
    def __init__(self, in_dim_kv, in_dim_q, out_dim, num_heads=4, across_time=False):
        super(FactorizedMultiHeadAttention, self).__init__()
        self.in_dim = in_dim_kv
        self.out_dim = out_dim
        self.across_time = across_time
        self.num_heads = num_heads
        assert out_dim % num_heads == 0, f'out_dim ({out_dim}) must be divisible by num_heads
({num_heads})'
```

```
        self.head_dim = out_dim // num_heads
        self.scale = math.sqrt(in_dim_kv)

        self.query = nn.Sequential(
            nn.Linear(in_dim_q, out_dim),
            nn.ReLU()
        )
        self.key = nn.Sequential(
            nn.Linear(in_dim_kv, out_dim),
            nn.ReLU()
        )
        self.value = nn.Sequential(
            nn.Linear(in_dim_kv, out_dim),
            nn.ReLU()
        )

        self.out = nn.Sequential(
            nn.Linear(out_dim, out_dim),
            nn.ReLU()
        )

    def forward(self, seq_1, seq_2, src_mask=None, tgt_mask=None):
        """
        :param seq_1: (batch_size, agents, time_steps, in_dim) | (batch_size, map_points, time_steps,
in_dim)
        :param seq_2: (batch_size, agents, time_steps, in_dim)
        :param src_mask: (batch_size, agents, time_steps) | (batch_size, map_points, time_steps)
        :param tgt_mask: (batch_size, agents, time_steps) | (batch_size, map_points, time_steps)
        :return: (batch_size, agents, time_steps, out_dim)
        """
        k = self.key(seq_1)

        k = k.view(-1, k.shape[1], k.shape[2], self.num_heads, self.head_dim).permute(3, 0, 1, 2, 4)
        v = self.value(seq_1)
        v = v.view(-1, v.shape[1], v.shape[2], self.num_heads, self.head_dim).permute(3, 0, 1, 2, 4)
        q = self.query(seq_2)

        bsz, agents, time_steps, _ = q.shape

        q = q.view(-1, q.shape[1], q.shape[2], self.num_heads, self.head_dim).permute(3, 0, 1, 2, 4)

        if self.across_time:
            # agents-to-agents only, throws exception due to incompatible dimensions otherwise
            scores = torch.matmul(q, k.transpose(-2,
                                                 -1)) / self.scale

            if src_mask is not None:
                scores = apply_flat_mask(scores, ~src_mask, mask_type='row')
            if tgt_mask is not None:
                scores = apply_flat_mask(scores, ~tgt_mask, mask_type='col')

            scores = F.softmax(scores, dim=-1)
            att_output = torch.matmul(scores, v)
        else:
            scores = torch.matmul(q.transpose(-2, -3),
```

```
                        k.permute(0, 1, 3, 4,
                                  2)) / self.scale

        if src_mask is not None:
            scores = apply_flat_mask(scores, ~src_mask.permute(0, 2, 1), mask_type='col')
        if tgt_mask is not None:
            scores = apply_flat_mask(scores, ~tgt_mask.permute(0, 2, 1), mask_type='row')

        scores = F.softmax(scores, dim=-1)
        att_output = torch.matmul(scores, v.permute(0, 1, 3, 2, 4)).permute(0, 1, 3, 2, 4)

    att_output = att_output.permute(1, 2, 3, 0, 4).contiguous().view(bsz, agents, time_steps, -1)

    att_output = self.out(att_output)

    return att_output


class FactorizedTransformerBlock(nn.Module):
    def __init__(self, in_dim_kv, in_dim_q, out_dim, num_heads=4, across_time=False):
        super(FactorizedTransformerBlock, self).__init__()
        self.in_dim = in_dim_kv
        self.out_dim = out_dim
        self.across_time = across_time
        self.num_heads = num_heads
        assert out_dim % num_heads == 0, f'out_dim ({out_dim}) must be divisible by num_heads
({num_heads})'

        self.attention = FactorizedMultiHeadAttention(in_dim_kv, in_dim_q, out_dim, num_heads,
across_time)
        self.norm_1 = nn.LayerNorm(out_dim)
        self.norm_2 = nn.LayerNorm(out_dim)

        self.fc = nn.Sequential(
            nn.Linear(out_dim, out_dim),
            nn.ReLU()
        )

    def forward(self, seq_1, seq_2, src_mask=None, tgt_mask=None):
        """
        :param seq_1: (batch_size, agents, time_steps, in_dim) | (batch_size, map_points, time_steps,
in_dim)
        :param seq_2: (batch_size, agents, time_steps, in_dim)
        :param src_mask: (batch_size, agents, time_steps) | (batch_size, map_points, time_steps)
        :param tgt_mask: (batch_size, agents, time_steps) | (batch_size, map_points, time_steps)
        :return: (batch_size, agents, time_steps, out_dim)
        """
        att_output = self.attention(seq_1, seq_2, src_mask, tgt_mask)
        x = self.norm_1(att_output + seq_2)
        x = self.norm_2(x + self.fc(x))

        return x

import torch.nn as nn
```

```
from .attention import FactorizedTransformerBlock
from .embedding import PositionalEncoding


class SceneEncoder(nn.Module):
    def __init__(self, agent_in_dim, map_in_dim, out_dim=256, num_heads=4):
        super(SceneEncoder, self).__init__()
        self.agent_in_dim = agent_in_dim
        self.map_in_dim = map_in_dim
        self.out_dim = out_dim
        self.num_heads = num_heads

        self.agent_embeddings = nn.Sequential(
            nn.Linear(agent_in_dim, out_dim),
            nn.ReLU(),
            PositionalEncoding(out_dim)
        )

        self.map_embeddings = nn.Sequential(
            nn.Linear(map_in_dim, out_dim),
            nn.ReLU(),
            PositionalEncoding(out_dim)
        )

        self.agent_transformer_1 = nn.Sequential(
            FactorizedTransformerBlock(out_dim, out_dim, out_dim, num_heads, across_time=True),
            FactorizedTransformerBlock(out_dim, out_dim, out_dim, num_heads, across_time=False),
            FactorizedTransformerBlock(out_dim, out_dim, out_dim, num_heads, across_time=True),
            FactorizedTransformerBlock(out_dim, out_dim, out_dim, num_heads, across_time=False),
            FactorizedTransformerBlock(out_dim, out_dim, out_dim, num_heads, across_time=True),
            FactorizedTransformerBlock(out_dim, out_dim, out_dim, num_heads, across_time=False)
        )

        self.map_transformer = FactorizedTransformerBlock(out_dim, out_dim, out_dim, num_heads,
across_time=False)

        self.cross_attention_1 = FactorizedTransformerBlock(out_dim, out_dim, out_dim, num_heads,
across_time=False)

        self.agent_transformer_2 = nn.Sequential(
            FactorizedTransformerBlock(out_dim, out_dim, out_dim, num_heads, across_time=True),
            FactorizedTransformerBlock(out_dim, out_dim, out_dim, num_heads, across_time=False),
        )

        self.cross_attention_2 = FactorizedTransformerBlock(out_dim, out_dim, out_dim, num_heads,
across_time=False)

        self.agent_transformer_3 = nn.Sequential(
            FactorizedTransformerBlock(out_dim, out_dim, out_dim, num_heads, across_time=True),
            FactorizedTransformerBlock(out_dim, out_dim, out_dim, num_heads, across_time=False),
        )

    def forward(self, agent_seq, map_seq, agent_mask=None, map_mask=None):
        """
        :param agent_seq: (batch_size, agents, time_steps, agent_in_dim)
```

```
:param map_seq: (batch_size, map_points, time_steps, map_in_dim)
:param agent_mask: (batch_size, agents, time_steps)
:param map_mask: (batch_size, map_points, time_steps)
:return: (batch_size, agents, time_steps, out_dim)
"""
agent_seq = self.agent_embeddings(agent_seq)
map_seq = self.map_embeddings(map_seq)

agent_seq = self.agent_transformer_1(agent_seq, agent_seq, agent_mask, agent_mask)
map_seq = self.map_transformer(map_seq, map_seq, map_mask, map_mask)

agent_seq = self.cross_attention_1(map_seq, agent_seq, map_mask, agent_mask)
agent_seq = self.agent_transformer_2(agent_seq, agent_seq, agent_mask, agent_mask)
agent_seq = self.cross_attention_2(map_seq, agent_seq, map_mask, agent_mask)
agent_seq = self.agent_transformer_3(agent_seq, agent_seq, agent_mask, agent_mask)

return agent_seq
```