Міністерство освіти і науки України
Харківський національний університет імені В.Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

«Затверджую»
в.о. завідуючого кафедри
комп'ютерних систем та робототехніки
_____ к. ф.-м. н., доцент Максим ХРУСЛОВ
«___» грудня 2024 р.

Пояснювальна записка

до кваліфікаційної роботи
магістра

на тему «**МОДЕЛЬ ПЕРЕДАЧІ ДАНИХ У КОМП'ЮТЕРНИХ МЕРЕЖАХ НА ОСНОВІ ГРАФОВИХ НЕЙРОННИХ МЕРЕЖ**»

Спеціальність 123 – Комп'ютерна інженерія
Галузь знань: 12 – Інформаційні технології
Освітня програма «Комп'ютерна інженерія»

Захищено на засіданні

Екзаменаційної комісії № 42

протокол № __ від __.12.2024 р.

Оцінка _____ / _____

Голова Екзаменаційної комісії

_____ СКОБ Ю. О.

Виконав:
Студент групи КІ – 61
ЛІТВІНОВ Григорій Віталійович

Керівник: доцент кафедри комп'ютерних систем та робототехніки, к.т.н.
СТРІЛЕЦЬ Вікторія Євгенівна

Рецензент: к.т.н., доцент, в.о. зав. кафедри теоретичної та прикладної інформатики
МЕНЯЙЛОВ Євген Сергійович

Харків – 2024

АНОТАЦІЯ

Пояснювальна записка до магістерської кваліфікаційної роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел і 4 додатків. Загальний обсяг роботи складає 81 сторінка, із яких 52 сторінки основної частини з 24 рисунками, 6 таблицями, списку використаних джерел із 51 найменувань та 4 додатками.

Мета кваліфікаційної роботи – підвищити ефективність мереж передачі даних через створення моделі оптимізації маршрутизації у мережах передачі даних за допомогою графових нейронних мереж.

Об'єкт дослідження – процес передачі даних у комп'ютерних мережах, маршрутизація трафіку та процес управління трафіком у мережах.

Предмет дослідження – моделі, методи та алгоритми оптимізації маршрутизації в мережах передачі даних, зокрема графові нейронні мережі (GNN).

Проблема, яка вирішується в кваліфікаційній роботі, полягає у створенні ефективної моделі, яка забезпечує оптимізовану маршрутизацію в мережах передачі даних шляхом зниження середньої затримки, покращення пропускної здатності та рівномірного розподілу навантаження між вузлами, з урахуванням динамічних умов мережі та складної топології, використовуючи можливості графових нейронних мереж.

Область застосування – програмно-конфігуровані мережі (SDN), телекомунікаційні системи, центри обробки даних, мережі Інтернет-провайдерів, IoT-мережі.

***Ключові слова*:** маршрутизація, графові нейронні мережі (GNN), EdgeGCN, оптимізація маршруту, комп'ютерна мережі, машинне навчання, Python, PyTorch.

ABSTRACT

An explanatory note to the master`s attestation work is created in the introduction, three sections, conclusions, a list of sources used and four additional substances. The total volume of work is 81 pages, of which 52 pages of the main part with 24 figures, 6 table, 51 names of the list of used sources and 4 additions. In this work, research and development of the software model for automating the processing of medical queries have been conducted.

The purpose of the qualification work is to increase the efficiency of data transmission networks by creating a routing optimization model in data transmission networks using graph neural networks.

The subject of research is the process of data transmission in computer networks, traffic routing and the process of traffic management in networks.

The subject of the study is models, methods and algorithms for optimizing routing in data transmission networks, in particular graph neural networks (GNN).

The problem to be solved in the qualification work is to create an effective model that provides optimized routing in data transmission networks by reducing the average delay, improving throughput and evenly distributing the load between nodes, taking into account dynamic network conditions and complex topology, using the capabilities of graph neural networks.

Application areas include software-defined networks (SDN), telecommunication systems, data centers, Internet provider networks, IoT networks.

Keywords: routing, graph neural networks (GNN), EdgeGCN, route optimization, computer networks, machine learning, Python, PyTorch.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ

API – англ. *Application Programming Interface*, укр. *програмний інтерфейс прикладного програмування*

GAT – англ. *Graph Attention Networks*, укр. *графові мережі уваги*

GCN – англ. *Graph Convolutional Networks*, укр. *графові згорткові мережі*

GNN – англ. *Graph Neural Networks*, укр. *графові нейронні мережі*

GraphSAGE – англ. *Graph Sample and Aggregate*, укр. *графове вибіркове агрегування*

PyG – PyTorch Geometric бібліотека на основі PyTorch для графових нейронних мереж.

QoS – англ. *Quality of Service*, укр. *якість обслуговування*

SDN – англ. *Software Defined Networking*, укр. *програмно-визначені мережі*

ВСТУП

У сучасну епоху стрімкого розвитку комп'ютерних технологій та глобалізації цифрових систем, мережі передачі даних стали невід'ємною частиною повсякденного життя, бізнесу та науки. Зростання обсягів інформації, що передається, а також збільшення складності та масштабності комп'ютерних мереж створюють нові виклики для оптимізації їхньої роботи. Забезпечення ефективності маршрутизації, надійності та швидкості передачі даних є ключовими завданнями для підтримки безперебійного функціонування таких систем.

Актуальність роботи. Основне завдання дослідження полягає в необхідності створення та впровадження автоматизованої моделі, яка використовує графові нейронні мережі для моделювання та оптимізації процесів передачі даних у комп'ютерних мережах. Подібна модель дозволить автоматично визначати оптимальні маршрути для передачі даних, адаптуючись до динамічних змін у мережевій топології та навантаженні.

Розробка автоматизованої моделі маршрутизації на основі графових нейронних мереж має потенціал значно підвищити загальну ефективність роботи комп'ютерних мереж, оптимізуючи передачу даних, зменшуючи затримки, підвищуючи пропускну здатність та забезпечуючи рівномірний розподіл навантаження. Крім того, така модель сприятиме спрощенню процесів управління трафіком і забезпечуватиме зручність та оперативність обробки інформації в умовах змінної структури мережі.

Метою дослідження є підвищення ефективності мереж передачі даних через створення моделі оптимізації маршрутизації у мережах передачі даних за допомогою графових нейронних мереж.

Об’єкт дослідження – процес передачі даних у комп’ютерних мережах, маршрутизація трафіку та процес управління трафіком у мережах.

Методи дослідження: моделювання мереж за допомогою Mininet, аналіз динамічних умов трафіку, навчання графових нейронних мереж у PyTorch Geometric, інтеграція SDN-контролера Ryu для управління трафіком.

Предмет дослідження – моделі, методи та алгоритми оптимізації маршрутизації в мережах передачі даних, зокрема графові нейронні мережі (GNN).

Завдання дослідження:

1. Проаналізувати існуючі методи оптимізації маршрутизації в мережах передачі даних.

2. Розробити модель маршрутизації на основі GNN, здатну адаптуватися до динамічних умов мережі.

3. Реалізувати алгоритми збору, обробки та подання даних у вигляді графової структури для подальшого навчання моделі.

4. Інтегрувати розроблену модель із SDN-контролером для автоматичного управління трафіком у реальних умовах.

5. Провести тестування моделі у віртуальному середовищі Mininet та оцінити її ефективність на основі ключових метрик, таких як середня затримка та пропускна здатність.

6. Порівняти результати моделі з традиційними методами маршрутизації та визначити переваги запропонованого підходу.

РОЗДІЛ 1.
МЕТОДИ ОПТИМІЗАЦІЇ МАРШРУТИЗАЦІЇ В КОМП'ЮТЕРНИХ МЕРЕЖАХ

У сучасних комп'ютерних мережах ефективна маршрутизація є ключовим елементом для забезпечення надійної та швидкої передачі даних. Традиційні методи маршрутизації, такі як протоколи дистанційно-векторної та стану каналу, мають свої переваги та недоліки. Зокрема, алгоритми адаптивної маршрутизації вимагають обліку і обробки поточної інформації про реальний стан мережі, що може призводити до додаткового навантаження та затримок [1].

З розвитком технологій з'являються нові підходи до оптимізації маршрутизації. Зокрема, методи багатопоточної маршрутизації дозволяють ефективніше використовувати мережеві ресурси та забезпечувати якість сервісу [2].

Однак, ці методи також стикаються з викликами, пов'язаними зі складністю управління та масштабованістю.

Останнім часом набуває популярності використання графових нейронних мереж для вирішення задач маршрутизації. Графові нейронні мережі дозволяють моделювати складні взаємозв'язки в мережах та адаптуватися до змінних умов, що робить їх перспективним інструментом для оптимізації процесів передачі даних [3].

Таким чином, аналіз існуючих методів оптимізації маршрутизації та дослідження можливостей застосування графових нейронних мереж є актуальним напрямом для підвищення ефективності комп'ютерних мереж.

1.1 Проблема маршрутизації в комп'ютерних мережах

У контексті комп'ютерних мереж та оптимізації маршрутизації з використанням графових нейронних мереж важливо розуміти ключові поняття.

Маршрутизація – це процес визначення оптимального шляху для передачі даних від джерела до місця призначення в комп'ютерних мережах. Вона забезпечує ефективний та надійний зв'язок між різними вузлами мережі, використовуючи протоколи та алгоритми для вибору найкращих маршрутів [6].

GNN – це тип нейронних мереж, призначених для роботи з даними, представленими у вигляді графів. Вони здатні моделювати складні взаємозв'язки між об'єктами, що дозволяє застосовувати їх у задачах, де дані мають графову структуру, таких як соціальні мережі, молекулярні структури або комп'ютерні мережі

Протоколи маршрутизації – це набір правил та алгоритмів, які визначають, як маршрутизатори в мережі обмінюються інформацією для вибору оптимальних шляхів передачі даних. Вони поділяються на внутрішні (IGP) та зовнішні (EGP) протоколи, залежно від області застосування [4].

Статична маршрутизація – це метод маршрутизації, при якому маршрути налаштовуються вручну адміністратором мережі і залишаються незмінними, доки не будуть змінені вручну. Вона є стабільнішою і вимагає мінімум апаратних ресурсів маршрутизатора для обслуговування таблиці [4].

Динамічна маршрутизація – це метод маршрутизації, при якому маршрути автоматично оновлюються за допомогою протоколів маршрутизації, що дозволяє мережі адаптуватися до змін у топології або умовах трафіку. Це забезпечує гнучкість, масштабованість та адаптивність мережі .

Розуміння цих понять є фундаментальним для дослідження та впровадження ефективних методів оптимізації маршрутизації в комп'ютерних мережах із використанням графових нейронних мереж.

1.1.1 Основні поняття в задачі маршрутизації

У контексті комп'ютерних мереж та оптимізації маршрутизації з використанням графових нейронних мереж важливо розуміти ключові поняття.

Маршрутизація – це процес визначення оптимального шляху для передачі даних від джерела до місця призначення в комп'ютерних мережах. Вона забезпечує ефективний та надійний зв'язок між різними вузлами мережі, використовуючи протоколи та алгоритми для вибору найкращих маршрутів [6].

GNN – це тип нейронних мереж, призначених для роботи з даними, представленими у вигляді графів. Вони здатні моделювати складні взаємозв'язки між об'єктами, що дозволяє застосовувати їх у задачах, де дані мають графову структуру, таких як соціальні мережі, молекулярні структури або комп'ютерні мережі

Протоколи маршрутизації – це набір правил та алгоритмів, які визначають, як маршрутизатори в мережі обмінюються інформацією для вибору оптимальних шляхів передачі даних. Вони поділяються на внутрішні (IGP) та зовнішні (EGP) протоколи, залежно від області застосування [4].

Статична маршрутизація – це метод маршрутизації, при якому маршрути налаштовуються вручну адміністратором мережі і залишаються незмінними, доки не будуть змінені вручну. Вона є стабільнішою і вимагає мінімум апаратних ресурсів маршрутизатора для обслуговування таблиці [4].

Динамічна маршрутизація – це метод маршрутизації, при якому маршрути автоматично оновлюються за допомогою протоколів маршрутизації, що дозволяє мережі адаптуватися до змін у топології або умовах трафіку. Це забезпечує гнучкість, масштабованість та адаптивність мережі .

Розуміння цих понять є фундаментальним для дослідження та впровадження оптимізації маршрутизації в комп'ютерних мережах із використанням графових ефективних методів нейронних мереж.

1.1.2 Класифікація методів маршрутизації

Методи маршрутизації в комп'ютерних мережах класифікуються за різними ознаками, що дозволяє обрати оптимальний підхід залежно від специфіки мережі та вимог до її функціонування (рис. 1.1).

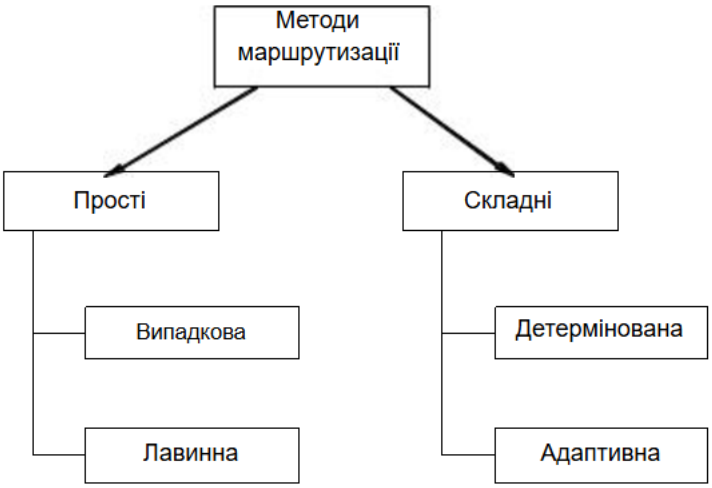


Рисунок 1.1 – Класифікація методів маршрутизації [7]

За способом оновлення маршрутів:

– Адміністратор вручну налаштовує маршрути, які залишаються незмінними до внесення нових налаштувань. Цей метод підходить для невеликих або стабільних мереж, де зміни топології відбуваються рідко. Перевагою є простота та передбачуваність, однак недоліком є відсутність автоматичної адаптації до змін у мережі.

– Використання протоколів маршрутизації для автоматичного оновлення маршрутів на основі змін у мережевій топології. Це забезпечує гнучкість та адаптивність, особливо в великих або часто змінюваних мережах. Проте динамічна маршрутизація може бути складнішою в налаштуванні та вимагати більше ресурсів.

За типом алгоритму маршрутизації:

– Кожен маршрутизатор обмінюється інформацією про відстань до інших вузлів з сусідніми маршрутизаторами. Прикладом є протокол RIP. Ці алгоритми прості у впровадженні, але можуть мати повільну конвергенцію та бути схильними до проблем, таких як "рахування до нескінченності".

– Кожен маршрутизатор збирає інформацію про стан каналів у мережі та будує повну карту мережі. На основі цього обчислюються найкоротші шляхи до всіх вузлів. Прикладом є протокол OSPF. Ці алгоритми забезпечують швидку

конвергенцію та точнішу інформацію про мережу, але вимагають більше ресурсів для обробки та зберігання даних.

За рівнем ієрархії:

– Використовуються для обміну маршрутною інформацією всередині автономної системи. Прикладами є OSPF та IS-IS.

– Призначені для обміну маршрутною інформацією між різними автономними системами. Основним прикладом є протокол BGP.

За способом передачі даних:

– Передача даних від одного відправника до одного отримувача.

– Передача даних від одного відправника до групи отримувачів.

– Передача даних від одного відправника до всіх вузлів у мережі.

Розуміння цих класифікацій дозволяє ефективно проектувати та керувати комп'ютерними мережами, забезпечуючи оптимальну продуктивність та надійність передачі даних.

1.2 Графові нейронні мережі (GNN) та їх застосування

Графові нейронні мережі ефективно захоплюють складні взаємозв'язки між об'єктами, що робить їх незамінними в багатьох сучасних застосуваннях. На рис. 1.2 показаний приклад архітектури шарів GNN.

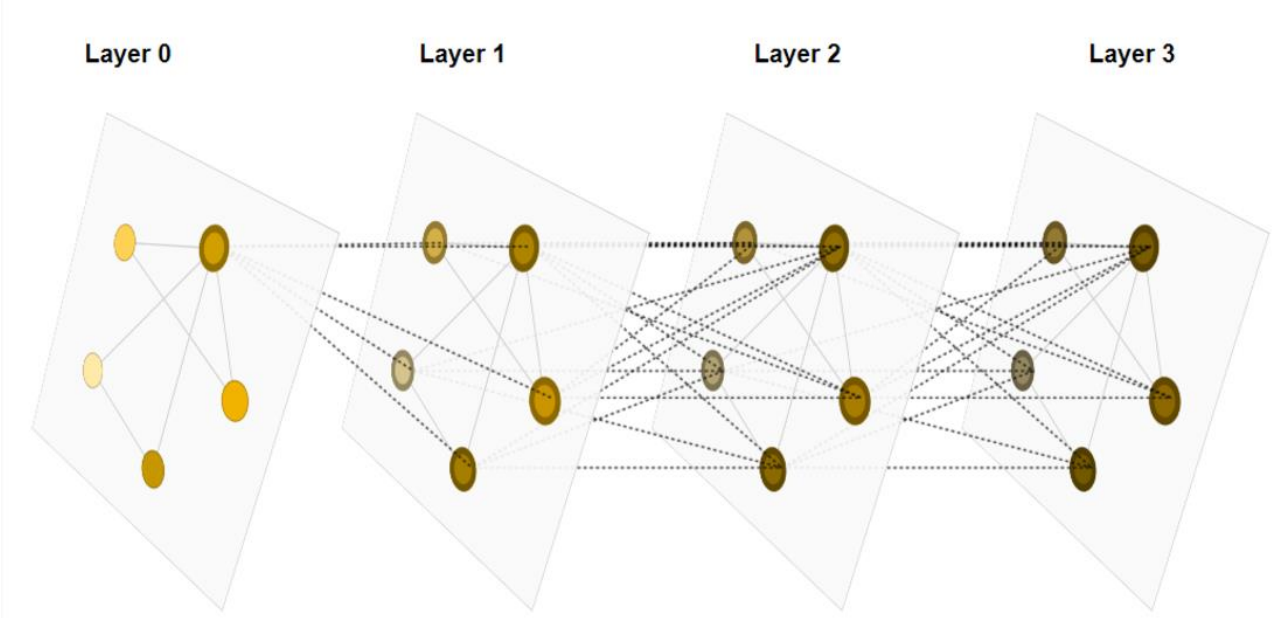


Рисунок 1.2 – Шари GNN [8]

Основні архітектури GNN:

1. *Graph Convolutional Networks (GCN)* узагальнюють концепцію згорткових нейронних мереж для графових структур, дозволяючи ефективно агрегувати інформацію від сусідніх вершин [8].

2. *Graph Attention Networks (GAT)* використовують механізм уваги для диференційованого зважування внеску сусідів при оновленні стану вершини, що підвищує гнучкість моделі [9].

3. *GraphSAGE (Graph Sample and AggregatE)* забезпечує масштабованість для великих графів шляхом вибіркового зразкування сусідів та агрегування їх інформації [10].

4. *Graph Isomorphism Networks (GIN)* мають високу дискримінативну здатність, дозволяючи розрізняти різні графові структури завдяки специфічним функціям агрегування [11].

У табл. 1.1 наведене порівняння основних архітектур графових нейронних мереж.

Таблиця 1.1

Порівняльна таблиця основних архітектур GNN

Архітектура	Особливості	Переваги	Недоліка
GCN	Використовує спектральні методи для згортки на графах	Ефективне агрегування інформації від сусідів	Обмежена здатність до моделювання складних залежностей
GAT	Застосовує механізм уваги для зважування сусідів	Гнучке врахування важливості різних сусідів	Вища обчислювальна складність
GraphSAGE	Вибіркове зразкування та агрегування сусідів	Масштабованість для великих графів	Можлива втрата інформації при зразкуванні
GIN	Використовує потужні функції агрегування для розрізнення графів	Висока дискримінативна здатність	Може вимагати більше даних для навчання

1.2.1 Основи графових нейронних мереж

Графи складаються з вузлів (вершин) та ребер, які моделюють об'єкти та їх взаємозв'язки відповідно. GNN дозволяють ефективно аналізувати складні структури, де взаємодії між елементами відіграють ключову роль.

Основні компоненти GNN:

1. *Вузли* представляють об'єкти або сутності в графі.

2. *Ребра* відображають взаємозв'язки або взаємодії між вузлами.

3. *Функції стану вузлів*, які оновлюється на основі інформації від сусідніх вузлів та власного попереднього стану.

Принцип роботи GNN

GNN використовують ітеративний процес оновлення станів вузлів, де кожен вузол агрегує інформацію від своїх сусідів для формування нового стану. Цей процес можна описати наступними кроками:

1. Кожен вузол збирає повідомлення від своїх сусідніх вузлів, які містять інформацію про їхні стани та ребра, що їх з'єднують.

2. На основі зібраної інформації та власного попереднього стану, вузол обчислює свій новий стан за допомогою функції оновлення.

3. Після кількох ітерацій оновлення станів, кожен вузол може генерувати вихідні дані, які використовуються для виконання конкретного завдання, наприклад, класифікації або прогнозування.

Графові нейронні мережі мають кілька переваг перед традиційними методами обробки даних. Вони здатні працювати з даними будь-якої структури та розміру, моделюючи складні залежності між елементами, що дозволяє ефективно обробляти інформацію, представлену у вигляді графів, таких як соціальні мережі або молекулярні структури. GNN враховують як локальну, так і глобальну інформацію в графі, що забезпечує більш точні прогнози та класифікації. Крім того, вони здатні узагальнювати знання на нові, раніше невідомі структури, підвищуючи свою гнучкість та застосовність у різних сферах. Ефективні алгоритми агрегації та оновлення станів дозволяють GNN обробляти великі графи.

Проте GNN мають і певні обмеження. Обчислювальні витрати можуть бути значними при роботі з дуже великими графами. Глибокі GNN можуть страждати від проблеми затухання градієнтів, що ускладнює їх навчання.

1.2.2 Застосування GNN у мережевих задачах

Графові нейронні мережі застосовуються в різних сферах людської діяльності, включаючи соціальні мережі, біоінформатику, комп'ютерні мережі та рекомендаційні системи. Вони дозволяють ефективно моделювати складні взаємозв'язки між об'єктами, що робить їх незамінними в багатьох сучасних застосуваннях.

1. Обчислення графів (рис. 1.3):

– співставлення графів (graph matching) [12-13];

– кластеризація графів (graph clustering) [14-16];
– генерація графів (graph generation) [17-18].

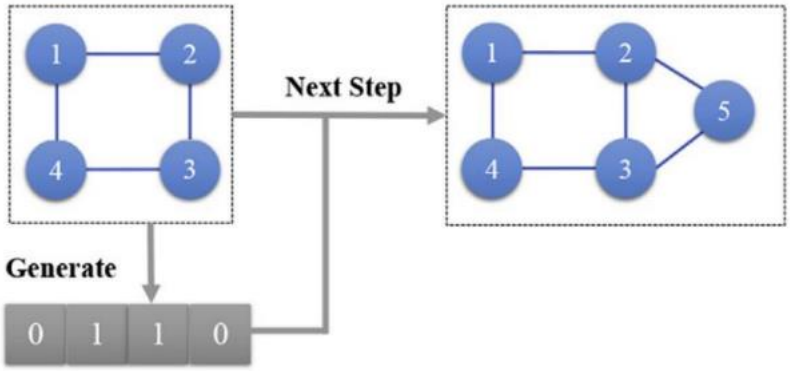


Рисунок 1.3 – Приклад роботи генерації графів [19]

2. Фізика:
– моделювання фізичних систем (physical systems modeling) [20].
3. Хімія (рис. 1.4):
– “молекулярні відбитки” (molecular fingerprints) [21];
– передбачення хімічної реакції (chemical reaction prediction) [22].

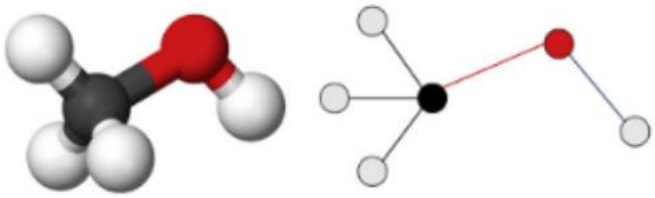


Рисунок 1.4 – Представлення молекул у вигляді графів [19]

4. Біологія:
– передбачення білкового інтерфейсу (protein interface prediction) [23];
– передбачення побічних ефектів (side effects prediction) [24];
– класифікація хвороб (disease classification) [25].

5. Мережа дорожнього руху: передбачення стану дор. руху (traffic state prediction) [26];

6. Рекомендаційні системи (рис. 1.5):

– передбачення інтеракцій виду користувач-предмет (user-item interaction prediction) [27];

– соціальні рекомендації (social recommendation) [28].

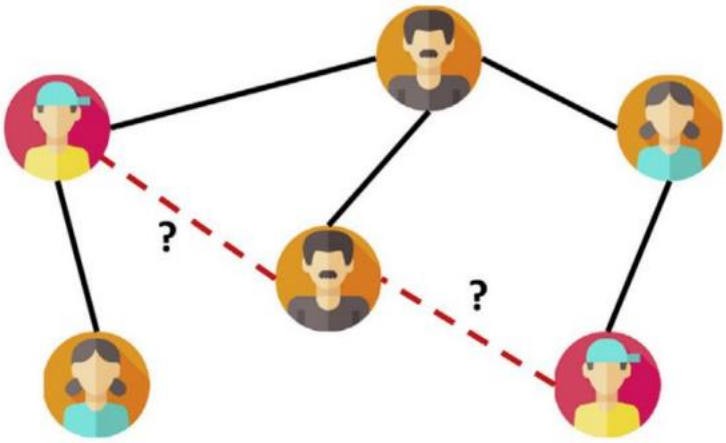


Рисунок 1.5 – Приклад соціально-рекомендаційної системи [19]

7. Обробка тексту:

– класифікація тексту (text classification) [29];

– маркування послідовностей (sequence labeling) [30];

– машинний переклад (neural machine translation) [31];

– витягання зв’язків (relation extraction) [32];

– витягання подій (event extraction) [33];

– верифікація фактів (fact verification) [34];

– відповіді на питання (question answering) [35].

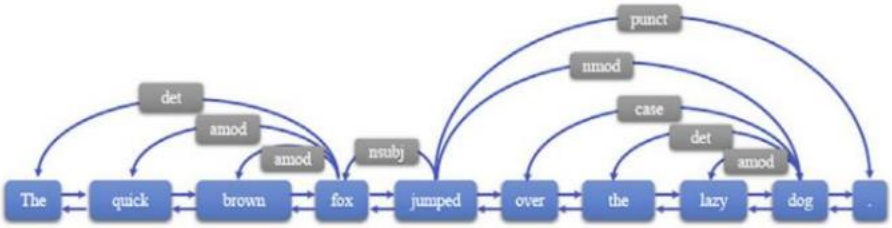


Рисунок 1.6 – Представлення текстових даних у вигляді графу [19]

8. Зображення (рис. 1.8):

– розуміння соціальних взаємозв’язків (social relationship understanding) [36];

– класифікація зображень (image classification) [37];

– візуальні відповіді на питання (visual question answering) [38];

– знаходження об’єктів (object detection) [39];

– знаходження інтеракцій (interaction detection) [40];

– класифікація регіонів (region classification) [41];

– семантична сегментація (semantic segmentation) [42].

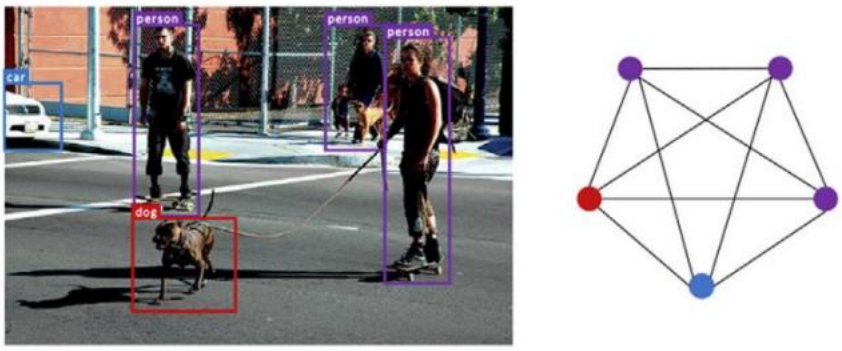


Рисунок 1.7 – Представлення знаходження зв’язків на зображенні у вигляді графів [19]

9. Інші:

– задачі фондового ринку (stock market) [43];

– програмно визначені мережі (software defined networks) [44];

– комбінаторна оптимізація (combinatorial optimization) [45];

– верифікація програм. забезпечення (program verification) [46].

Таким чином, GNN є потужним інструментом для роботи з даними, які мають складну структуровану природу, і продовжують знаходити нові застосування в різних галузях науки та техніки.

1.3 Інструменти та середовища для моделювання мереж та GNN

Складність і масштаб мережевих структур постійно зростають, ефективне моделювання комп'ютерних мереж та графових нейронних мереж (GNN) стає критично важливим. Це дозволяє дослідникам і практикам глибше розуміти поведінку систем, оптимізувати їхню продуктивність та передбачати можливі проблеми. Різноманітні інструменти та середовища для моделювання надають можливість створювати точні віртуальні репрезентації мереж, проводити експерименти в контрольованих умовах та розробляти нові алгоритми без ризику для реальних систем.

1.3.1 Mininet

Mininet – це один з небагатьох емуляторів комп’ютерних мереж з підтримкою протоколу OpenFlow. Mininet – це полегшена програмноконфігуровна мережа і тестова платформа; в ній використовується полегшена технологія віртуалізації, щоб зробити єдину систему схожою на повну мережу для запуску системи ядра і призначеного для користувача коду, про які було задумано. В рамках однієї віртуальної машини за допомогою Mininet можна розвернути мережі з довільною кількістю вузлів, з найрізноманітнішими топологіями за допомогою простого синтаксису в досить примітивному інтерпретаторі.

Важливою перевагою є те, що основна функціональність написана мовою Python, що дозволяє описувати багато топологій за допомогою спеціального синтаксису цієї мови.

Функції, які реалізовує Mininet:

- підтримка компонентів SDN, зокрема OpenFlow та Open vSwitch;
- можливість створення та налаштування складних і конфігурованих топологій;
- наявність Python API для гнучкого управління та автоматизації процесів;
- висока масштабованість, що дозволяє моделювати мережі з тисячами вузлів.

Завдяки цим можливостям Mininet став стандартним інструментом для досліджень та навчання в галузі мережевих технологій, особливо в контексті SDN та OpenFlow. Він дозволяє швидко та ефективно моделювати складні мережеві сценарії, сприяючи розвитку нових технологій та рішень у сфері комп'ютерних мереж.

1.3.2 PyTorch Geometric

PyTorch Geometric (PyG) – це бібліотека на основі PyTorch, призначена для спрощення розробки та навчання GNN для широкого спектра застосувань, пов'язаних зі структурованими даними. Вона містить різноманітні методи глибокого навчання на графах та інших нерегулярних структурах, відомих як геометричне глибоке навчання, запозичені з численних опублікованих робіт [47].

Основні можливості PyTorch Geometric:

- PyG надає зручні засоби для роботи з графовими структурами, включаючи підтримку різних типів графів, таких як спрямовані, неспрямовані та гетерогенні графи. Це дозволяє моделювати складні взаємозв'язки між об'єктами в даних.
- Бібліотека містить реалізації багатьох сучасних архітектур графових нейронних мереж, таких як GCN, GAT та GraphSAGE, що дозволяє швидко експериментувати з різними моделями.
- PyG підтримує обробку великих графів за допомогою ефективних алгоритмів міні-пакетної обробки, що дозволяє працювати з багатьма малими графами або одним великим графом. Крім того, бібліотека підтримує

багатогпушне навчання та інтеграцію з torch.compile для підвищення продуктивності [47].

- PyG надає доступ до великої кількості загальновживаних еталонних датасетів, а також прості інтерфейси для створення власних наборів даних, що спрощує процес підготовки даних для навчання моделей.
- Завдяки побудові на основі PyTorch, PyG забезпечує легку інтеграцію з існуючими проектами та використання знайомих інструментів і методів, що спрощує розробку та розгортання моделей.

1.3.3 Ryu SDN Controller

Ryu – це компонентно-орієнтований фреймворк для програмно-визначених мереж (SDN), розроблений для спрощення створення та управління мережевими додатками. Він надає набір програмних компонентів із чітко визначеними API, що дозволяє розробникам легко створювати нові додатки для управління мережею та контролю за нею [48].

Основні характеристики Ryu:

- Ryu підтримує кілька протоколів для управління мережевими пристроями, зокрема OpenFlow (версії 1.0, 1.2, 1.3, 1.4, 1.5) та розширення Nicira, а також Netconf і OF-config [48].
- Фреймворк повністю написаний мовою Python, що спрощує розробку та інтеграцію нових функцій.
- Весь код Ryu доступний під ліцензією Apache 2.0, що дозволяє вільно використовувати та модифікувати його [48].

Ryu широко застосовується для розробки та тестування SDN-додатків, оскільки надає гнучку платформу для управління мережевими пристроями та потоками трафіку. Завдяки підтримці різних протоколів і компонентно-орієнтованій архітектурі, Ryu дозволяє швидко адаптувати мережеву інфраструктуру до змінних вимог бізнесу та технологій.

Станом на останні оновлення, проект Ryu потребує нових мейнтейнерів для подальшого розвитку та підтримки. Як альтернативу, рекомендується використовувати проєкт os-ken, який є форком Ryu та активно підтримується спільнотою [48].

Ryu є потужним інструментом для розробки та впровадження SDN-рішень, надаючи розробникам гнучкість та простоту в управлінні мережевими додатками. Однак, враховуючи поточний стан проекту, користувачам рекомендується звернути увагу на активні форки, такі як os-ken, для забезпечення актуальності та підтримки своїх рішень.

1.4 Постановка задачі дослідження

Підсумовуючи викладене в попередніх розділах, можна стверджувати, що основним завданням кваліфікаційної роботи є розробка ефективної моделі GNN для аналізу та моделювання складних мережевих структур.

Для вирішення цього завдання необхідно вирішити такі задачі:

– провести детальний огляд сучасних методів моделювання мереж та застосування GNN, визначити їхні переваги та недоліки;

– ідентифікувати основні проблеми та прогалини в існуючих підходах, які обмежують ефективність застосування GNN у моделюванні комп’ютерних мереж;

– створити модель GNN для аналізу та моделювання мережевих структур, враховуючи специфіку виявлених проблем;

– провести серію експериментів для оцінки ефективності розробленої методики в порівнянні з існуючими підходами;

– проаналізувати отримані результати, визначити сильні та слабкі сторони запропонованої моделі, а також можливі напрямки її вдосконалення;

– на основі проведеного дослідження сформулювати висновки та надати рекомендації щодо практичного застосування розробленої моделі та подальших напрямків досліджень.

Висновки за розділом 1

Проведений огляд літератури дозволив виявити сучасні тенденції та підходи до застосування графових нейронних мереж у моделюванні складних мережевих структур. Було встановлено, що GNN демонструють високу ефективність у різних сферах, таких як соціальні мережі, біоінформатика та комп'ютерні мережі. Однак, існують певні обмеження та проблеми, які потребують подальшого дослідження та вдосконалення.

Багато існуючих моделей GNN стикаються з труднощами при обробці дуже великих графів, що призводить до значних обчислювальних витрат та затримок. Зі збільшенням глибини GNN виникає проблема затухання градієнтів, що ускладнює процес навчання та може призвести до зниження точності моделі. Більшість поточних досліджень зосереджені на однорідних графах, тоді як реальні мережі часто є гетерогенними, містять різні типи вузлів та ребер, що ускладнює їх моделювання. Результати, отримані за допомогою GNN, не завжди легко інтерпретувати, що може обмежувати їх практичне застосування в певних галузях.

Враховуючи виявлені проблеми та прогалини, доцільно зосередити дослідження на розробці моделей, які підвищують масштабованість та ефективність GNN при роботі з великими та гетерогенними графами. Особливу увагу слід приділити створенню моделей, стійких до проблеми затухання градієнтів, а також розробці інструментів, що покращують інтерпретованість результатів. Такий підхід дозволить розширити сферу застосування графових нейронних мереж та підвищити їх практичну цінність у моделюванні складних мережевих структур.

РОЗДІЛ 2.
ПОБУДОВА МОДЕЛІ МАРШРУТИЗАЦІЇ НА ОСНОВІ ГРАФОВИХ НЕЙРОНИХ МЕРЕЖ

Традиційні методи маршрутизації, такі як статичні та динамічні алгоритми, мають певні обмеження щодо адаптивності та масштабованості в умовах швидкозмінних мережевих середовищ. З появою графових нейронних мереж відкрилися нові можливості для моделювання та оптимізації маршрутів у складних мережах.

Графові нейронні мережі здатні ефективно обробляти дані, представлені у вигляді графів, що робить їх особливо корисними для аналізу мережевих структур. Вони дозволяють враховувати топологію мережі, стан окремих вузлів та каналів, а також динамічні зміни, що відбуваються в реальному часі. Це сприяє більш точному прогнозуванню та оптимізації маршрутів, підвищуючи загальну продуктивність мережі.

У цьому розділі розглядається процес побудови моделі маршрутизації на основі графових нейронних мереж. Особлива увага приділяється обґрунтуванню вибору архітектури GNN, враховуючи специфіку комп'ютерних мереж та вимоги до маршрутизації. Також аналізуються переваги використання GNN у порівнянні з традиційними методами, зокрема їх здатність до адаптивного навчання та обробки складних мережевих структур.

Розуміння та впровадження графових нейронних мереж у процес маршрутизації відкриває нові горизонти для підвищення ефективності та надійності комп'ютерних мереж, що є критично важливим у сучасному цифровому світі.

2.1 Обґрунтування вибору архітектури графових нейронних мереж

Оптимізація маршрутизації в комп'ютерних мережах є критичним аспектом забезпечення ефективної та надійної передачі даних між вузлами

мережі. Цей процес включає визначення оптимальних шляхів для передачі пакетів, враховуючи різні фактори, такі як пропускна здатність, затримки, надійність та поточне завантаження мережі.

Основні задачі оптимізації маршрутизації включають мінімізацію затримок, максимізацію пропускної здатності, балансування навантаження, забезпечення надійності та адаптивність до змін. Детальний опис цих задач та приклади їх застосування наведено в таблиці 2.1.

Таблиця 2.1

Основні задачі оптимізації маршрутизації

Задача	Опис	Приклад застосування
Мінімізація затримок	Зменшення часу доставки пакетів, вибір найшвидшого маршруту.	Відеоконференції, де важлива низька затримка сигналу.
Максимізація пропускної здатності	Забезпечення максимальної передачі даних за одиницю часу.	Хмарні сервіси для передачі великих обсягів даних.
Балансування навантаження	Розподіл трафіку між маршрутами для уникнення перевантаження окремих каналів.	Онлайн-ігри для забезпечення стабільного з'єднання.
Забезпечення надійності	Вибір маршрутів, які мінімізують втрату пакетів та витримують відмови вузлів.	Інтернет-банкінг, де критична надійність передачі даних.
Адаптивність до змін	Можливість оперативної реакції на зміни в топології мережі або умовах трафіку.	IoT-системи для моніторингу в реальному часі.

Для вирішення зазначених задач використовуються різні методи оптимізації маршрутизації, такі як статичні та динамічні методи, мультишляхова та багатокритеріальна маршрутизація. Кожен з цих методів має свої переваги та недоліки, які детально розглянуто в таблиці 2.2.

Таблиця 2.2

Методи оптимізації маршрутизації

Метод	Опис	Переваги	Недоліки
Статичні методи	Використовують заздалегідь визначені маршрути.	Простота реалізації.	Не враховують динамічні зміни в мережі.
Динамічні методи	Алгоритми оновлюють маршрути в реальному часі.	Висока адаптивність до змін.	Вимоги до обчислювальних ресурсів.
Мультишляхова маршрутизація	Застосування кількох альтернативних маршрутів.	Збільшення надійності та ефективності.	Складність управління трафіком.
Багатокритеріальна маршрутизація	Враховує кілька параметрів одночасно (затримка, пропускна здатність, надійність).	Оптимальні рішення для складних сценаріїв.	Високі вимоги до ресурсів та складність реалізації.

Сучасні підходи до оптимізації маршрутизації включають використання алгоритмів машинного навчання для аналізу великих обсягів даних про трафік та стан мережі, що дозволяє прогнозувати можливі проблеми та пропонувати оптимальні маршрути. Багатопотокова маршрутизація дозволяє одночасно використовувати кілька маршрутів для передачі даних, підвищуючи пропускну здатність та надійність з'єднання. Протоколи стану каналу, такі як OSPF (Open Shortest Path First), використовують алгоритм Дейкстри для обчислення найкоротших шляхів, забезпечуючи швидку конвергенцію та ефективне управління маршрутизацією в масштабних мережах.

Таким чином, завдання оптимізації маршрутизації охоплює широкий спектр проблем: від мінімізації затримок і балансування навантаження до забезпечення якості обслуговування. Застосування новітніх технологій, таких як графові нейронні мережі, дозволяє покращити ефективність роботи комп'ютерних мереж, роблячи їх більш гнучкими, стійкими та продуктивними, здатними адаптуватися до змін умов та навантаження в реальному часі.

2.2 Алгоритми пошуку найкоротшого шляху у мережах

У комп'ютерних мережах топологія може бути розглянута як граф, де вузли відповідають маршрутизаторам чи комутаторам, а ребра представляють мережеві канали або зв'язки між цими пристроями (рис. 2.1).

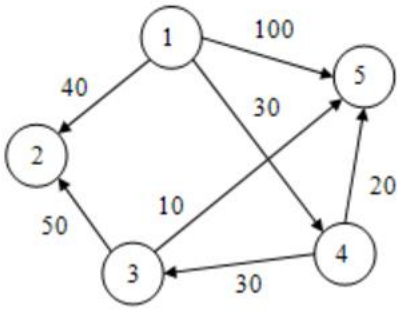


Рисунок 2.1 – Деякий орієнтований граф [49]

Застосування графових нейронних мереж у маршрутизації дозволяє ефективно моделювати ці зв'язки та визначати оптимальні маршрути для передачі даних, що значно підвищує ефективність функціонування мережі.

У комп'ютерних мережах виникає багато задач, які можуть бути вирішені за допомогою графових нейронних мереж. Основною задачею є визначення оптимального маршруту для передачі даних.

2.2.1 Алгоритм Дейкстри

Алгоритм Дейкстри призначений для знаходження найкоротших шляхів від однієї вихідної вершини до всіх інших у графі з невід'ємними вагами ребер. Основна ідея полягає в поступовому розширенні множини оброблених вузлів, починаючи з вихідного, та оновленні найкоротших відстаней до сусідніх вузлів.

Математично алгоритм можна описати наступним чином:

1. Ініціалізувати відстані до всіх вузлів як нескінченність, окрім вихідного вузла, для якого відстань дорівнює нулю.

2. Вибрати вузол з найменшою відстанню, який ще не оброблений.

3. Для кожного сусіднього вузла обчислити нову можливу відстань як суму відстані до поточного вузла та ваги ребра між ними. Якщо нова відстань менша за поточну, оновити її.

4. Позначити поточний вузол як оброблений.

5. Повторювати кроки 2-4, доки не будуть оброблені всі вузли.

Часова складність алгоритму Дейкстри становить $O(V^2)$ для реалізації на основі матриці суміжності або $O((V + E) \log V)$ при використанні черги з пріоритетами, де V – кількість вузлів, E – кількість ребер.

2.2.2 Алгоритм Беллмана-Форда

Алгоритм Беллмана-Форда також знаходить найкоротші шляхи від однієї вихідної вершини до всіх інших, але може працювати з графами, що містять ребра з від'ємними вагами. Принцип роботи базується на поступовому покращенні оцінок відстаней шляхом релаксації всіх ребер протягом кількох ітерацій.

Основні кроки алгоритму:

1. Ініціалізувати відстані до всіх вузлів як нескінченність, окрім вихідного вузла, для якого відстань дорівнює нулю.

2. Повторювати (V-1) разів:
   - для кожного ребра (u, v) перевірити, чи можна покращити відстань до v через u. Якщо так, оновити її.

3. Перевірити наявність циклів з від'ємною вагою: якщо після додаткової ітерації знаходяться покращення, це свідчить про наявність таких циклів.

Алгоритм Беллмана-Форда шукає функцію d(v) як розв’язання рівняння

$$d(v) = \min \{d(w) + f(e) \mid e = (w, v) \in E\}, \forall v \neq u, \quad (2.1)$$

з початковою умовою $d(u) = 0$. Основною операцією алгоритму є релаксація ребра: якщо $e = (w, v) \in E$ і $d(v) > d(w) + f(e)$, то проводиться присвоювання $d(v) \leftarrow d(w) + f(e)$.

Часова складність алгоритму Беллмана-Форда становить *O(V * E)*, що робить його менш ефективним для великих графів порівняно з алгоритмом Дейкстри.

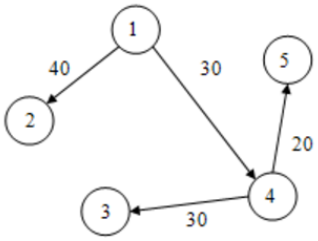


Рисунок 2.2 – Маршрут найкоротшої довжини [49]

2.2.3 Порівняльний аналіз

Продовжуючи порівняння класичних алгоритмів пошуку найкоротшого шляху з графовими нейронними мережами, розглянемо їхні переваги та недоліки в контексті маршрутизації в комп'ютерних мережах (табл. 2.3).

Таблиця 2.3

Порівняльна таблиця

Критерій	Алгоритм Дейкстри	Алгоритм Беллмана-Форда	Графові нейронні мережі
Часова складнісь	O(V²) або O((V + E) log V) залежно від реалізації	O(V * E)	Залежить від архітектури та обсягу навчання; може бути оптимізована для великих графів
Підтримка від’ємних ваг	Не підтримує	Підтримує	Може навчатися на графах з від'ємними вагами, якщо такі присутні в навчальних даних
Адаптивність	Обмежена; потребує повторного запуску при зміні топології	Обмежена; потребує повторного запуску при зміні топології	Висока; здатність адаптуватися до змін у реальному часі
Масштабованість	Ефективний для середніх графів	Менш ефективний для великих графів	Висока; підходить для великих та складних мереж
Розподіленість	Потребує централізованого контролю	Може бути реалізований у розподілених системах	Може бути реалізована як у централізованих, так і в розподілених системах

Графові нейронні мережі набувають популярності завдяки здатності навчатися з топологічної інформації, що міститься в графах. На відміну від традиційних нейронних мереж, які не враховують структуру даних, GNN дозволяють:

- аналізувати мережі з урахуванням їхньої топології;
- вирішувати завдання, де структура даних відіграє ключову роль.

Це відкриває можливості для:

- пошуку оптимальних маршрутів;
- класифікації вузлів;
- прогнозування навантаження на мережу.

GNN пропонують гнучкий підхід, що дозволяє адаптивно визначати маршрути залежно від поточного стану мережі. Основними перевагами GNN є:

- Здатність працювати з великими мережевими графами.
- Ефективне використання інформації з різних джерел.

У комп'ютерних мережах, де трафік і топологія постійно змінюються, GNN дозволяють:

- швидко адаптувати маршрути для забезпечення якісної передачі даних;
- уникати перевантажень шляхом динамічного оновлення моделей на основі нових даних про мережу.

Наприклад, зміни в завантаженні каналів або поява нових вузлів можуть бути швидко враховані, що забезпечує оптимальні маршрути та підвищує ефективність функціонування мережі.

2.3 Порівняння архітектур згорткової графової нейронної мережі (GCN) та графової мережі уваги (GAT)

GCN є одним з найпоширеніших типів GNN. Вони використовують згорткові операції для агрегації та оновлення вбудовувань вузлів на основі їхнього локального оточення.

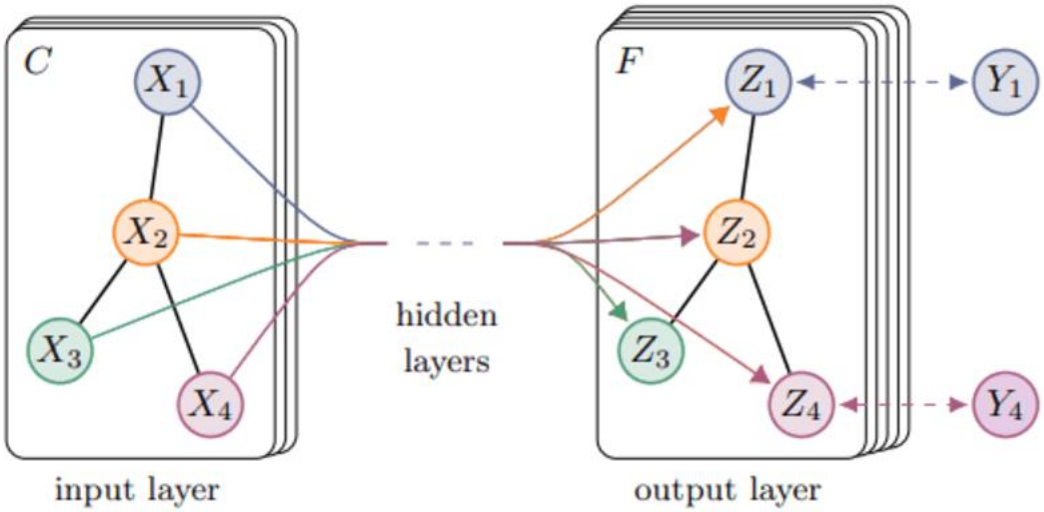


Рисунок 2.3 – Згорткова мережа графів [50]

Основне рівняння для GCN можна записати як:

$$H^{(l+1)} = \sigma(\widetilde{D}^{-\frac{1}{2}}\tilde{A}\widetilde{D}^{-\frac{1}{2}}H^{(l)}\,W^{(l)}, \tag{2.2}$$

де $\tilde{A} = A + I$ – матриця суміжності з доданими одиничними елементами на діагоналі (для врахування самих вузлів), $\widetilde{D}$ – діагональна матриця ступенів для $\tilde{A}$, $H(l)$ – матриця вбудовувань на шарі $l, W(l)$ – матриця ваг [50].

GAT вводять механізми уваги у GNN, що дозволяє вузлам вибірково звертати увагу на релевантних сусідів під час передавання повідомлень та агрегації.

Основне рівняння для GAT:

$$h_v^{(l+1)} = \sigma\left(\sum_{u\in(v)} a_{uv}^{(l)} W h_u^{(l)}\right), \tag{2.3}$$

де $a_{uv}^{(l)}$ – ваги уваги, що обчислюються за допомогою механізму уваги

У табл. 2.4 наведено порівняння GCN та GAT.

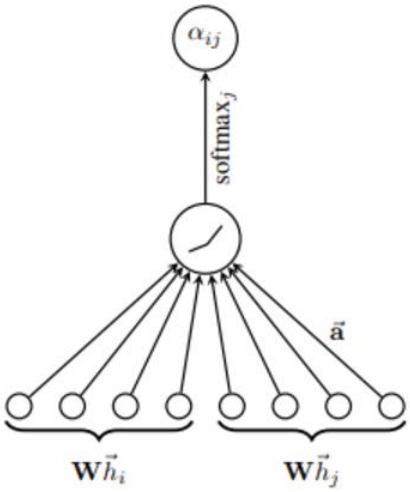


Рисунок 2.4 – Графічні мережі уваги [51]

Таблиця 2.4

Порівняльна таблиця основних характеристик GCN та GAT

Характеристика	GCN	GAT
Механізм агрегації	Використовує спектральні методи для агрегації інформації від сусідніх вузлів, застосовуючи фіксовані ваги, що залежать від структури графа.	Інтегрує механізм уваги, який дозволяє призначати різні ваги сусіднім вузлам на основі їхньої важливості, що підвищує гнучкість моделі.
Обробка різних типів графів	Підходить для неорієнтованих графів; для роботи з орієнтованими графами потребує модифікацій.	Може працювати як з орієнтованими, так і з неорієнтованими графами без додаткових змін.
Обробка великих графів	Може бути обмеженим при роботі з великими графами через необхідність обробки всього графа одночасно, що впливає на продуктивність.	Завдяки механізму уваги може ефективніше обробляти великі графи, фокусуючись на важливих зв'язках, що знижує обчислювальні витрати.
Гнучкість вагових коефіцієнтів	Використовує фіксовані ваги для всіх сусідніх вузлів, що може не враховувати різну важливість зв'язків.	Дозволяє призначати індивідуальні ваги кожному сусідньому вузлу, що підвищує точність моделі.
Складність реалізації	Має простішу архітектуру та легше реалізується, що робить її придатною для базових завдань.	Має складнішу архітектуру через інтеграцію механізму уваги, що може ускладнювати реалізацію та налаштування.

GCN мають простішу архітектуру та легше реалізуються, що робить їх придатними для базових завдань. Однак, вони використовують фіксовані ваги для всіх сусідніх вузлів, що може не враховувати різну важливість зв'язків. Крім того, GCN можуть бути обмеженими при роботі з великими графами через необхідність обробки всього графа одночасно, що впливає на продуктивність.

З іншого боку, GAT інтегрують механізм уваги, який дозволяє призначати різні ваги сусіднім вузлам на основі їхньої важливості, що підвищує гнучкість моделі. Це дозволяє ефективніше обробляти великі графи, фокусуючись на важливих зв'язках, що знижує обчислювальні витрати. Однак, GAT мають складнішу архітектуру через інтеграцію механізму уваги, що може ускладнювати реалізацію та налаштування. Саме тому в якості архітектури було обрано GCN.

2.4 Проектування системи на основі GCN для оптимізації маршрутизації

Оптимізація маршрутизації є ключовим аспектом у багатьох галузях, включаючи транспортні системи, телекомунікаційні мережі та логістику. Застосування згорткових графових нейронних мереж дозволяє ефективно обробляти складні графові структури, що представляють мережі доріг, комунікацій або інші системи з вузлами та зв'язками.

2.4.1 Навчальні дані

Для навчання моделі використовуються дані про мережеву топологію та зібрані у процесі роботи програми. Зокрема, збирається інформація про MAC-адреси, порти, статистику трафіку та пропускну здатність каналів, що дозволяє моделювати мережеву поведінку та шляхи. Важливою частиною є також визначення з'єднань між вузлами, яке дозволяє побудувати графову модель мережі, що є входом для графової нейронної мережі.

Збір цих даних відбувається через періодичні запити до комутаторів, що дозволяє відстежувати зміни в мережі, наприклад, пропускну здатність каналів, що є критичним для прогнозування ефективних маршрутів для трафіку.

2.4.2 Попередня обробка даних

Попередня обробка даних є важливою частиною підготовки даних для моделі, оскільки необхідно правильно трансформувати їх у формат, зручний для навчання графових нейронних мереж.

Для цього виконуються наступні кроки:

– Топологія мережі представлена у вигляді графа, де вузли відповідають комутаційним пристроям, а ребра — це з'єднання між пристроями через порти. Для цього використовується бібліотека networkx, яка дозволяє працювати з графами в Python.

– Для кожного порту збираються статистичні дані, зокрема кількість отриманих і відправлених байтів, що дозволяє оцінити трафік і пропускну здатність каналу.

– Дані топології мережі перетворюються в структуру, яка є сумісною з бібліотекою PyTorch Geometric, зокрема, створюється об'єкт типу Data, який містить індекси ребер та ознаки для вузлів і ребер.

2.4.3 Налаштування параметрів навчання

Параметри навчання моделі визначають, як модель буде оптимізувати свої ваги для досягнення найкращих результатів у прогнозуванні маршрутів.

Для налаштування параметрів навчання використовуються:

– визначення кількості нейронів у прихованому шарі GNN, що впливає на здатність моделі виявляти складні залежності в даних;

– визначення кроку, з яким змінюються ваги мережі на кожній ітерації навчання;

– визначення, скільки разів модель буде обробляти всі навчальні дані, перш ніж завершити навчання;

– можливість використання методів регуляризації, таких як L2-норма, щоб запобігти перенавчанню моделі.

У реалізації коду налаштування параметрів здійснюється через методи GCN для створення нейронної мережі з необхідною кількістю шарів та параметрів.

2.4.4 Поділ даних

Тренувальний набір: містив дані про топологію мережі та стани вузлів. Використовувався для безпосереднього навчання GCN, щоб модель могла навчитися розпізнавати оптимальні маршрути залежно від різних умов.

Валідаційний набір використовувався для оцінки якості навчання під час тренувального процесу. Це допомагало налаштовувати параметри моделі, щоб забезпечити її оптимальну продуктивність.

Тестовий набір після завершення навчання модель оцінювалася на нових даних, щоб перевірити її здатність прогнозувати оптимальні маршрути в умовах, відмінних від навчальних.

2.4.5 Оцінка якості моделі

Для оцінки точності GCN використовувалися кілька метрик:

– Точність визначення того, наскільки часто модель правильно передбачала оптимальні маршрути.

– Precision та Recall використовувалися для оцінки якості передбачення, особливо у випадках, коли мережа має складну топологію або динамічно змінюється.

– F1-Score використовувався для збалансованої оцінки якості моделі, враховуючи як Precision, так і Recall.

Ці метрики дозволяли комплексно оцінити, наскільки добре модель справляється із завданням оптимізації маршрутизації, та виявляти слабкі місця, які потребують додаткового налаштування або доопрацювання.

Висновки за розділом 2

У цьому розділі описано проектування системи для оптимізації маршрутизації в комп'ютерних мережах, що використовує графові нейронні мережі. Розглянуто етапи підготовки даних, налаштування параметрів навчання, а також оцінки ефективності моделі, яка передбачає маршрути на основі даних про мережеву топологію та статистику трафіку.

РОЗДІЛ 3.
ПРОГРАМНА РЕАЛІЗАЦІЯ МОДЕЛІ КОМП'ЮТЕРНОЇ МЕРЕЖІ

3.1 Топологія мережі

У якості контролера було використано RemoteController, що взаємодіє з топологією через IP 127.0.0.1 та порт 6633. Це дозволяє інтегрувати мережу Mininet з зовнішніми контролерами OpenFlow, такими як POX, Ryu або інші рішення, які забезпечують централізоване управління мережею і реалізацію алгоритмів маршрутизації та балансування навантаження.

Вузли джерела трафіку представляють собою десять вузлів (h1-h10), кожен з яких має унікальну IP-адресу формату 10.0.0.x, де x варіюється від 1 до 10, а також відповідну MAC-адресу. Ці вузли відповідають за генерування різних видів трафіку в мережі (рис. 3.1). Вузли приймачі трафіку (SINK) складаються з десяти вузлів (h11-h20), які служать кінцевими точками для трафіку. Вони мають IP-адреси формату 10.0.0.x, де x варіюється від 11 до 20.

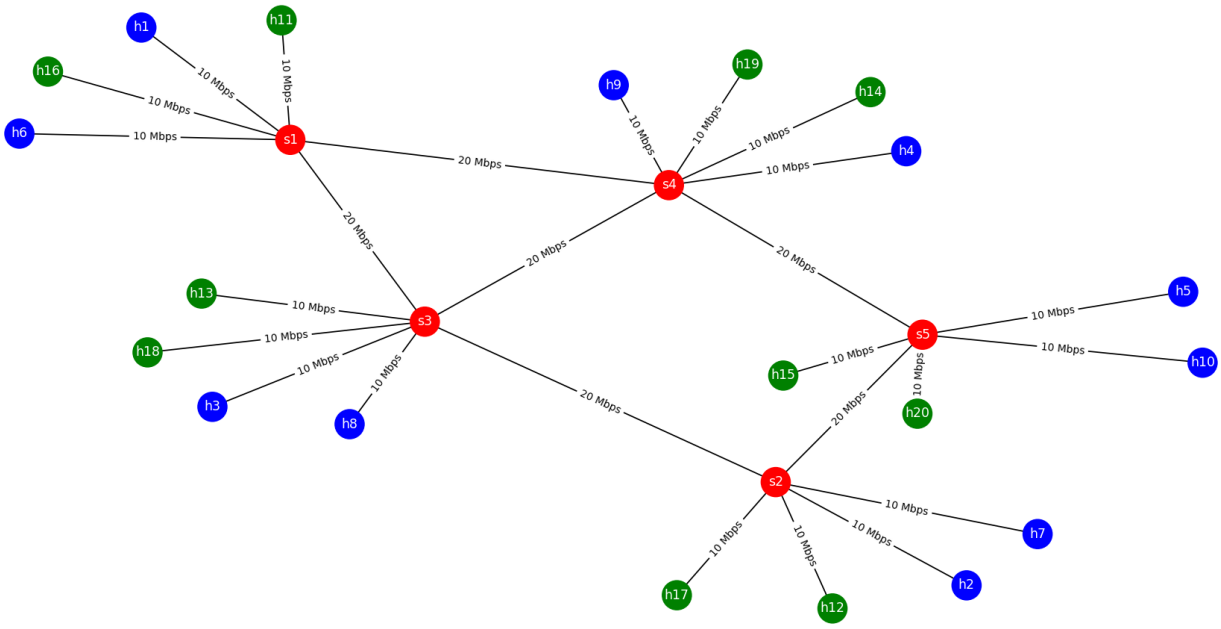


Рисунок 3.1 – Топологія мережі

Комутатори (s1-s5) підтримують протокол STP (Spanning Tree Protocol) для уникнення мережевих циклів та забезпечення надлишковості з'єднань. STP дозволяє зменшити ризик виникнення петльових маршрутів і підвищує стабільність мережі.

Вузли SRC з'єднуються з комутаторами, і пропускна здатність кожного з'єднання обмежена до 10 Мбіт/с, що дозволяє моделювати реальні обмеження мережевих ресурсів. Аналогічно, вузли SINK з'єднуються з комутаторами з пропускною здатністю 10 Мбіт/с. Комутатори з'єднані між собою для забезпечення надійності і резервування, і ці з'єднання мають більшу пропускну здатність – 20 Мбіт/с, що дозволяє їм обробляти великий обсяг трафіку між комутаторами та забезпечує альтернативні шляхи передачі.

Для моніторингу роботи мережі використовувалися інструменти, що дозволяють аналізувати основні показники продуктивності, такі як затримка, пропускна здатність, втрата пакетів, а також характеристики трафіку, зокрема розподіл потоків і обсяг переданих даних. Це дозволяє ідентифікувати вузькі місця мережі, оцінювати ефективність алгоритмів маршрутизації та оптимізувати використання мережевих ресурсів.

Топологія мережі була створена з акцентом на масштабованість і гнучкість. Завдяки цьому можна додавати нові вузли чи комутатори без значних змін у конфігурації. У процесі дослідження використовувалися симуляції з різними типами трафіку, включаючи TCP і UDP, що дозволило протестувати мережу в умовах реальних навантажень.

Загалом, така конфігурація дозволяє ефективно досліджувати поведінку мережі, аналізувати вплив різних параметрів на продуктивність і виявляти оптимальні стратегії управління трафіком у програмно-конфігурованих мережах (SDN).

3.2 Тестування та збір мережевих даних

Мережеві дослідження та експерименти з використанням симуляційних середовищ є ключовими для розуміння продуктивності, стабільності та стійкості мережі під різними типами навантажень. Для цього в топології, описаній у попередньому розділі, було реалізовано серію тестів і сценаріїв навантаження, які дозволяють отримати дані про основні характеристики мережі, включаючи затримки, пропускну здатність, втрати пакетів та стійкість до атак.

3.2.1 Навантаження мережі

У процесі тестування було реалізовано такі тестові сценарії навантаження:

– Для моделювання TCP SYN атаки використовувався інструмент hping3, який генерував численні SYN-пакети, спрямовані на порти 80 вузлів SINK. Цей сценарій дозволяє оцінити стійкість мережі до розподілених атак типу DoS (Denial of Service) та ефективність обробки атакуючого трафіку комутаторами. TCP SYN атака створює масивний трафік, що імітує спроби встановлення з'єднань, і може викликати перевантаження ресурсів на вузлах чи комутаторах.

– Перевірка доступності вузлів за допомогою ICMP пінгу була важливим етапом тестування. Вимірювався RTT (Round-Trip Time) для оцінки затримок між вузлами, а також перевірялася стабільність і доступність з'єднань під час навантаження. Цей тест дозволяє визначити вплив різних сценаріїв трафіку на загальну продуктивність мережі.

– Для оцінки продуктивності комутаторів і обробки з'єднань із високою частотою передачі пакетів було створено UDP-трафік на порту 53 (стандартний порт DNS). Цей сценарій дає змогу визначити, наскільки ефективно комутатори маршрутизують трафік із низькою затримкою та високою частотою пакетів.

– У додаткових сценаріях тестування було створено UDP-трафік на порту 123 (типовий для NTP) із різними параметрами інтенсивності та обсягу пакетів. Це дозволило проаналізувати поведінку мережі при комбінованих типах навантаження і визначити можливі вузькі місця в обробці трафіку.

Використання інструмента hping3

Інструмент hping3 є потужним рішенням для генерації мережевого трафіку та моделювання різних видів атак. Завдяки можливості створювати TCP, UDP або ICMP пакети з детальними параметрами, він використовується для тестування стійкості мережі. У випадку TCP SYN атаки hping3 генерував велику кількість SYN-запитів до певного порту на цільовому вузлі.

Це дозволило оцінити:

– частоту втрат пакетів під час масових запитів;

– затримки у відповіді на запити;

– пропускну здатність комутаторів при великих обсягах атакуючого трафіку.

Інструмент дозволяє задавати параметри частоти відправки пакетів, вибирати порти й IP-адреси джерела, а також підробляти IP-адреси, що дозволяє моделювати реальні атаки. Завдяки цим функціям hping3 використовується як основний засіб для тестування мереж у симуляціях.

3.2.2 Методи збору інформації про мережу

Для підтримки актуального стану мережевої топології та збору даних про її продуктивність використовуються спеціалізовані методи, які забезпечують постійний моніторинг зв’язків, портів комутаторів і мережевого трафіку. Описані нижче методи реалізують різні аспекти цього процесу.

Метод build_edges (рис 3.2) відповідає за періодичне побудування та оновлення зв'язків між комутаторами та хостами в мережевій топології.

Він використовується для збору даних про зв'язки, їх збереження в CSV-файл та додавання додаткових показників, таких як пропускна здатність, затримка та рівень втрат пакетів.

Використовується таймер для періодичного виклику методу кожну секунду, що забезпечує актуальність зібраної інформації.

Метод збирає інформацію про зв'язки між комутаторами та хостами.

Для кожного порту комутатора перевіряється, чи є з’єднання з іншим пристроєм (комутатором чи хостом). Якщо таке з’єднання існує, створюється запис про зв'язок.

Визначається максимальна пропускна здатність між зв'язаними портами, а також затримка і рівень втрат пакетів.

Зібрані дані про зв'язки додаються до списку edges.

```
@set_ev_cls(ofp_event.EventOFPPortDescStatsReply, MAIN_DISPATCHER)
def port_desc_stats_reply_handler(self, ev):
    msg = ev.msg
    datapath = msg.datapath
    dpid = datapath.id
    por = msg.body
    ports = []

    for p in ev.msg.body:
        ports.append('port_no=%d hw_addr=%s name=%s config=0x%08x '
                     'state=0x%08x curr=0x%08x advertised=0x%08x '
                     'supported=0x%08x peer=0x%08x curr_speed=%d '
                     'max_speed=%d' %
                     (p.port_no, p.hw_addr,
                     p.name, p.config,
                     p.state, p.curr, p.advertised,
                     p.supported, p.peer, p.curr_speed,
                     p.max_speed))
    for port in por:
        port_no = port.port_no
        if 4294967294 != port_no:
            port_info = {
                    'hw_addr': port.hw_addr,
                    'name': port.name.decode('utf-8').strip('\x00'),
                    'curr_speed': port.curr_speed,
                }
            self.graph[dpid][port_no] = str(port.curr_speed)
            self.port_desc[dpid][port_no] = port_info

    self.logger.debug(f'Порт {port_no} на DPID={dpid}: {port_info}')
```

Рисунок 3.2 – Метод build_edges

Дані записуються у файл ‘Link.csv’, який містить інформацію про кожен зв'язок: часову мітку, ідентифікатори комутаторів/хостів, пропускну здатність, затримку та рівень втрат пакетів (рис. 3.3).

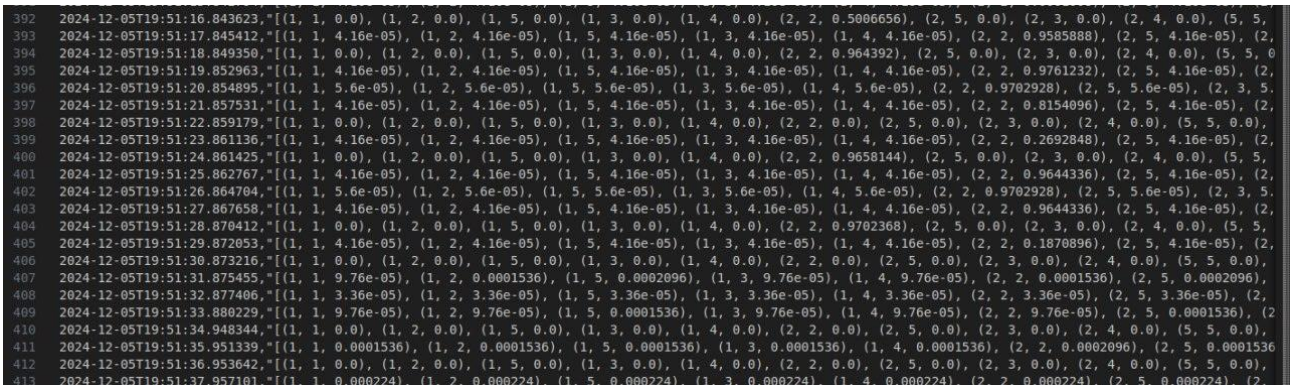


Рисунок 3.3 – Структура файлу ‘Link.csv’

Метод switch_features_handler (Рис 3.4) обробляє подію, коли комутатор приєднується до контролера. Цей метод відсилає запит на отримання статистики портів, а також додає початкове правило для обробки пакетів. Таким чином, контролер збирає первинну інформацію про комутатор та його порти.

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    msg = ev.msg
    datapath = ev.msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    match = parser.OFPMatch()

    actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                      ofproto.OFPCML_NO_BUFFER)]
    self.add_flow(datapath, 0, match, actions)
    self.send_port_desc_stats_request(datapath)
    self.logger.debug('OFPSwitchFeatures received: '
                      'datapath_id=0x%016x n_buffers=%d '
                      'n_tables=%d auxiliary_id=%d '
                      'capabilities=0x%08x',
                      msg.datapath_id, msg.n_buffers, msg.n_tables,
                      msg.auxiliary_id, msg.capabilities)
```

Рисунок 3.4 – Метод switch_features_handler

Метод port_desc_stats_reply_handler (рис 3.5) збирає інформацію про всі порти комутатора, таку як MAC-адреса, номер порту, поточна швидкість тощо. А також обробляє відповідь на запит статистики портів (EventOFPPortDescStatsReply). Ця інформація додається до атрибуту self.port_desc і використовується для оновлення топології.

```python
def port_desc_stats_reply_handler(self, ev):
    msg = ev.msg
    datapath = msg.datapath
    dpid = datapath.id
    por = msg.body
    ports = []

    for p in ev.msg.body:
        ports.append('port_no=%d hw_addr=%s name=%s config=0x%08x '
                     'state=0x%08x curr=0x%08x advertised=0x%08x '
                     'supported=0x%08x peer=0x%08x curr_speed=%d '
                     'max_speed=%d' %
                     (p.port_no, p.hw_addr,
                     p.name, p.config,
                     p.state, p.curr, p.advertised,
                     p.supported, p.peer, p.curr_speed,
                     p.max_speed))
    for port in por:
        port_no = port.port_no
        if 4294967294 != port_no:
            port_info = {
                    'hw_addr': port.hw_addr,
                    'name': port.name.decode('utf-8').strip('\x00'),
                    'curr_speed': port.curr_speed,
                }
            self.graph[dpid][port_no] = str(port.curr_speed)
            self.port_desc[dpid][port_no] = port_info
    self.logger.debug(f'Порт {port_no} на DPID={dpid}: {port_info}')
    if dpid not in self.topology['switches']:
        self.topology['switches'][dpid] = {
            'dpid': dpid,
            'ports': {}
        }
    for port_no, port_info in self.port_desc[dpid].items():
        self.topology['switches'][dpid]['ports'][port_no] = port_info
    self.save_top()
```

Рисунок 3.5 – Метод port_desc_stats_reply_handler

Функція _port_stats_reply_handler (рис 3.6) обробляє статистику по портах після отримання відповіді на запит (EventOFPPortStatsReply). Вона зберігає дані про трафік, такий як кількість переданих і отриманих байтів (tx_bytes, rx_bytes) для кожного порту комутатора. Зібрані дані використовуються для аналізу пропускної здатності мережі, а також зберігаються в структуру self.current_cycle_stats.

```python
@set_ev_cls(ofp_event.EventOFPPortStatsReply, MAIN_DISPATCHER)
def _port_stats_reply_handler(self, ev):
    switch_dpid = ev.msg.datapath.id
    self.tmp = {switch_dpid}
    for p in ev.msg.body:
        port_no = p.port_no
        if str(4294967294) != str(port_no):
            self.bw[switch_dpid][p.port_no] = (p.tx_bytes - self.prev_bytes[switch_dpid][p.port_no])*8.0/10000000
            self.prev_bytes[switch_dpid][p.port_no] = p.tx_bytes
            self.current_cycle_stats[switch_dpid][port_no] = {
                'tx_bytes': p.tx_bytes,
                'rx_bytes': p.rx_bytes,
                'bandwidth_mbps': self.bw[switch_dpid][p.port_no]
            }

    if switch_dpid not in self.received_dpid:
        self.received_dpid.append(switch_dpid)
    if len(self.received_dpid) == len(self.switches):
        self.save_port_stats()
        self.current_cycle_stats = defaultdict(dict)
        self.received_dpid = []
```

Рисунок 3.6 – Метод port_desc_stats_reply_handler

Загальний збір даних про порти, топологію та трафік відбувається через запити і обробку відповідей на статистику комутаторів.

Дані про комутатори, порти, підключення та трафік зберігаються у структури типу defaultdict, а також записуються в JSON і CSV файли для подальшого використання.

Використання подій та періодичних запитів дозволяє контролеру постійно отримувати актуальну інформацію про мережу та підтримувати її в актуальному стані.

Завдяки використанню цих методів контролер має змогу постійно отримувати актуальну інформацію про стан мережі, її продуктивність і стабільність. Це забезпечує можливість швидкого реагування на зміни, виявлення проблем і оптимізації роботи мережі.

3.3 Архітектура моделі GCN

Модель базується на використанні GNN для ефективного навчання та обробки даних про топологію мережі. Основна мета цієї моделі полягає у класифікації ребер графа, що визначає, чи входить конкретне ребро до складу оптимального маршруту між парою вузлів. Це дозволяє не лише оптимізувати

маршрутизацію, але й забезпечити більш точне розуміння структури мережі та її поведінки.

Для досягнення цієї мети використовується підхід, який включає кілька ключових етапів: підготовку даних, побудову графової архітектури моделі, а також тренування та валідацію моделі. Кожен із цих етапів відіграє важливу роль у досягненні високої точності та продуктивності під час виконання завдань маршрутизації.

3.3.1 Попередня обробка даних

Для ефективного прогнозування маршруту в мережевій топології необхідно мати повну інформацію про те, які ребра входять до складу оптимального маршруту між заданими парами вузлів. Це означає, що для кожної пари вузлів (джерело-призначення) повинні бути визначені маршрути та відповідно позначені ребра, які до них належать. Цей підхід забезпечує формування чітких даних для тренування моделі класифікації.

На етапі підготовки даних створюється граф для кожного рядка вхідного набору даних (DataFrame). Кожен рядок описує множину ребер (edges), а також їх параметри, такі як пропускна здатність (bandwidth) (рис. 3.7). Завдання цього етапу полягає у формуванні структурованої інформації про топологію мережі та її характеристики, яка стане основою для навчання моделі.

```python
def prepare_training_data_with_edge_labels(df, node_id_map, bandwidth_min, data):
    edge_labels = []

    for index, row in df.iterrows():
        edges = row['edges']
        G = nx.Graph()
        for edge in edges:
            src, dst, bandwidth = edge
            if src != dst and bandwidth > 0:
                G.add_edge(src, dst, bandwidth=bandwidth)

        nodes = list(G.nodes())
        for i in range(len(nodes)):
            for j in range(i+1, len(nodes)):
                src = nodes[i]
                tgt = nodes[j]
                path, _ = minimum_bandwidth_path(G, src, tgt, capacity='bandwidth')
                if path:

                    path_edges = list(zip(path[:-1], path[1:]))
                    for edge in path_edges:
                        src_id = node_id_map[edge[0]]
                        dst_id = node_id_map[edge[1]]

                        if src_id < dst_id:
                            edge_tuple = (src_id, dst_id)
                        else:
                            edge_tuple = (dst_id, src_id)
                        edge_labels.append(edge_tuple)
    from collections import Counter
    edge_count = Counter(edge_labels)
    total_routes = len(edge_count)
```

Рисунок 3.7 – Підготовка міток ребер на основі оптимальних маршрутів

Процес містить створення списку всіх ребер з мітками, що вказують на їхню приналежність до маршруту (рис. 3.8). Для кожного ребра визначається, чи є воно частиною оптимального маршруту, і присвоюється відповідна мітка: 1 для ребер, які входять до маршруту, та 0 для тих, які не входять. Це дозволяє формувати набір даних, необхідний для навчання моделі класифікації.

```
all_edges = []
all_labels = []
for idx in range(data.edge_index.size(1)):
    src = data.edge_index[0, idx].item()
    dst = data.edge_index[1, idx].item()
    if src < dst:
        edge_tuple = (src, dst)
    else:
        edge_tuple = (dst, src)
    all_edges.append(edge_tuple)
    label = 1 if edge_tuple in edge_count else 0
    all_labels.append(label)

return all_edges, all_labels
```

Рисунок 3.8 – Список всіх ребер з мітками

Важливим кроком є визначення оптимальних маршрутів між вузлами. Для цього використовується функція minimum_bandwidth_path (рис. 3.9), яка знаходить маршрут із найбільшою мінімальною пропускною здатністю, що відповідає реальним вимогам до якості обслуговування трафіку.

Після цього ребра маршруту позначаються відповідними мітками, що дозволяє створити узгоджену і зрозумілу структуру для подальшого аналізу та навчання.

```python
def minimum_bandwidth_path(G, source, target):

    import heapq
    max_bandwidth = {node: 0 for node in G.nodes()}
    max_bandwidth[source] = float('inf')
    heap = [(-max_bandwidth[source], source, [source])]

    while heap:
        current_bandwidth, u, path = heapq.heappop(heap)
        current_bandwidth = -current_bandwidth

        if u == target:
            return path, current_bandwidth

        for v, attrs in G[u].items():
            edge_bandwidth = attrs.get(capacity, 0)
            path_bandwidth = min(current_bandwidth, edge_bandwidth)
            if path_bandwidth > max_bandwidth[v]:
                max_bandwidth[v] = path_bandwidth
                heapq.heappush(heap, (-path_bandwidth, v, path + [v]))

    return None, 0
```

Рисунок 3.9 – Визначення оптимальних маршрутів між вузлами

3.3.2 Побудова архітектури

Архітектура моделі GNNRouteSelector складається з кількох основних компонентів (рис. 3.10):

– GCN Шари використовуються для генерації ембедінгів вузлів, що описують їх характеристики та зв’язки у графі.

– Для кожного ребра графа обчислюються ембедінги на основі комбінації ембедінгів його кінцевих вузлів. Це забезпечує врахування взаємозв'язків між вузлами.

– Повнозв'язний Шар відповідає за класифікацію ребер, визначаючи, чи є вони частиною маршруту.

```python
class GNNRoutePredictor(nn.Module):
    def __init__(self, num_node_features, hidden_dim, dropout=0.5):
        super(GNNRoutePredictor, self).__init__()
        self.conv1 = GCNConv(num_node_features, hidden_dim)
        self.conv2 = GCNConv(hidden_dim, hidden_dim)
        self.dropout = nn.Dropout(p=dropout)
        self.fc = nn.Linear(hidden_dim * 2, 1)

    def forward(self, data, edge_index):
        x, edge_indices = data.x, data.edge_index
        x = self.conv1(x, edge_indices)
        x = F.relu(x)
        x = self.dropout(x)
        x = self.conv2(x, edge_indices)
        x = F.relu(x)
        x = self.dropout(x)
        src = edge_index[:, 0]
        dst = edge_index[:, 1]
        src_emb = x[src]
        dst_emb = x[dst]
        combined = torch.cat([src_emb, dst_emb], dim=1)
        out = self.fc(combined).squeeze()
        return out
```

Рисунок 3.10 – Реалізація моделі GNNRoutePredictor

Ця архітектура забезпечує здатність моделі враховувати як локальні властивості вузлів, так і глобальну структуру графа (рис. 3.11).

```
{
    "switches": {
        "1": {
            "dpid": 1,
            "ports": {
                "1": {
                    "hw_addr": "02:d4:0c:36:ac:5e",
                    "link": "h1",
                    "name": "s1-eth1",
                    "config": "0x0",
                    "state": "0x4",
                    "curr": "0x840",
                    "advertised": "0x0",
                    "supported": "0x0",
                    "peer": "0x0",
                    "curr_speed": 10000000,
                    "max_speed": 0
                },
                "2": {
                    "hw_addr": "ce:bb:35:f6:03:db",
                    "link": "0000000000000003",
                    "name": "s1-eth2",
                    "config": "0x0",
                    "state": "0x4",
                    "curr": "0x840",
                    "advertised": "0x0",
                    "supported": "0x0",
                    "peer": "0x0",
                    "curr_speed": 10000000,
                    "max_speed": 0
                },
                "3": {
                    "hw_addr": "1a:8a:e9:09:71:87",
                    "link": "0000000000000004",
                    "name": "s1-eth3",
                    "config": "0x0",
                    "state": "0x4",
                    "curr": "0x840",
                    "advertised": "0x0",
                    "supported": "0x0",
                    "peer": "0x0",
                    "curr_speed": 10000000,
                    "max_speed": 0
                }
            }
        },
        "2": {
            "dpid": 2,
```


Рисунок 3.11 – Структура побудованого графа

3.3.3 Тренування моделі

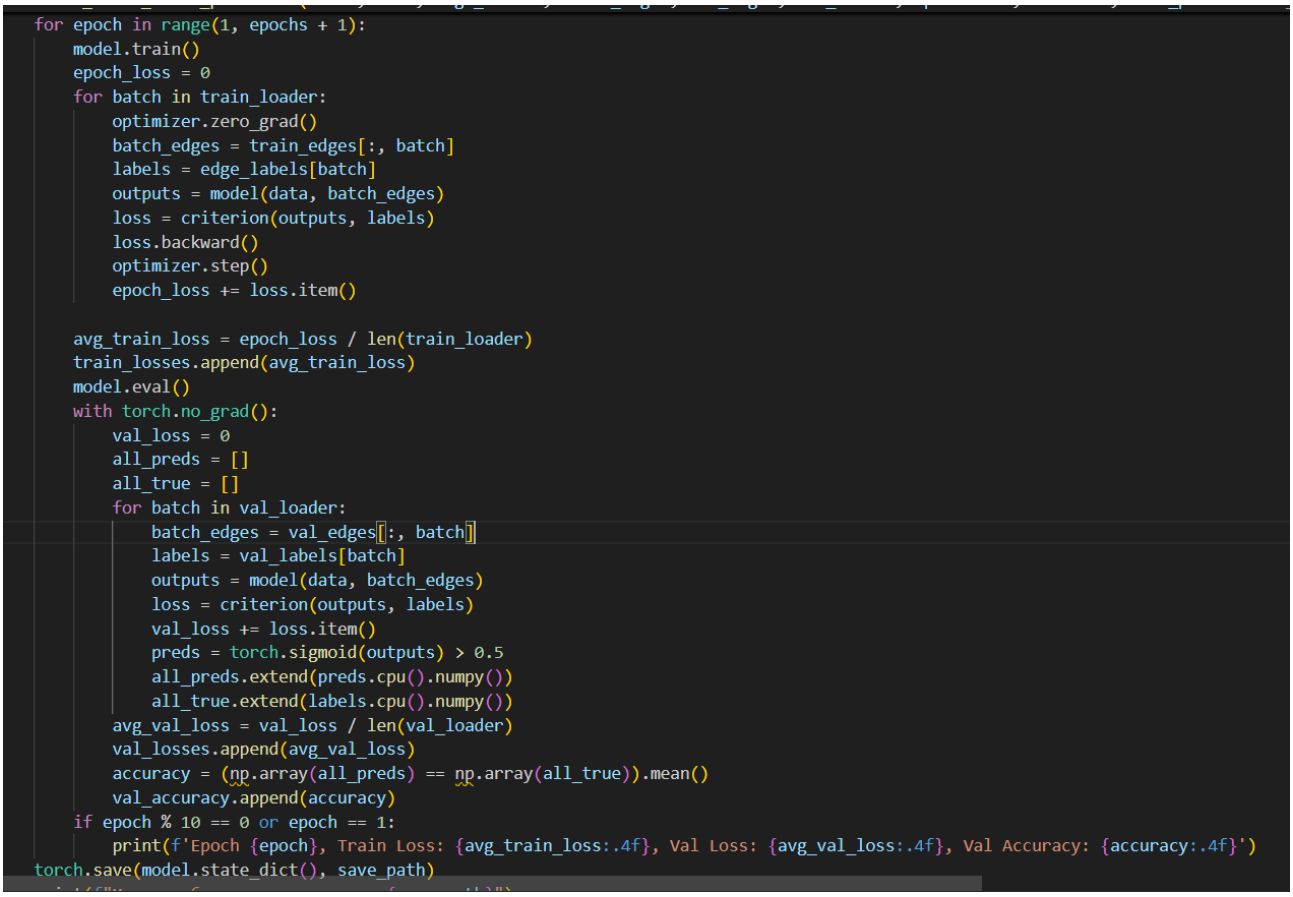
Для навчання моделі використовується задача класифікації ребер. Основним критерієм оптимізації є бінарна крос-ентропійна функція втрат, яка добре підходить для двокласових задач. Оптимізація параметрів здійснюється за допомогою алгоритму Adam, який забезпечує швидке та стабільне навчання моделі.

```python
def train_model_route_predictor(model, data, edge_labels, train_edges, val_edges, val_labels, epochs=100, lr=0.001, save_path='model_route_predictor
    optimizer = optim.Adam(model.parameters(), lr=lr)
    criterion = nn.BCEWithLogitsLoss()

    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model = model.to(device)

    data = data.to(device)
    train_edges = torch.tensor(train_edges, dtype=torch.long).t().contiguous().to(device)
    val_edges = torch.tensor(val_edges, dtype=torch.long).t().contiguous().to(device)
    edge_labels = torch.tensor(edge_labels, dtype=torch.float).to(device)
    val_labels = torch.tensor(val_labels, dtype=torch.float).to(device)

    # Поділ на батчі
    batch_size = 1024
    num_train = train_edges.size(1)
    num_val = val_edges.size(1)

    train_indices = torch.randperm(num_train)
    val_indices = torch.randperm(num_val)

    train_loader = torch.utils.data.DataLoader(train_indices, batch_size=batch_size, shuffle=True)
    val_loader = torch.utils.data.DataLoader(val_indices, batch_size=batch_size, shuffle=False)

    train_losses = []
    val_losses = []
    val_accuracy = []
```

Рисунок 3.12 – Навчання та валідація моделі

Процес навчання включає етапи обчислення втрати, оновлення параметрів моделі та регулярну валідацію (рис. 3.13). Це дозволяє оцінювати продуктивність моделі на тестових даних і запобігати перенавчанню. Валідація також дає змогу порівнювати результати моделі з фактичними маршрутами, що підтверджує її точність та ефективність.

Етап підготовки даних та архітектурні рішення, закладені у моделі GNNRouteSelector, дозволяють точно прогнозувати оптимальні маршрути в складних мережевих топологіях. Поєднання даних про структуру графа, пропускну здатність ребер та їхню класифікацію створює основу для ефективного управління трафіком і покращення загальної продуктивності мережі.



```
for epoch in range(1, epochs + 1):
    model.train()
    epoch_loss = 0
    for batch in train_loader:
        optimizer.zero_grad()
        batch_edges = train_edges[:, batch]
        labels = edge_labels[batch]
        outputs = model(data, batch_edges)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        epoch_loss += loss.item()

    avg_train_loss = epoch_loss / len(train_loader)
    train_losses.append(avg_train_loss)
    model.eval()
    with torch.no_grad():
        val_loss = 0
        all_preds = []
        all_true = []
        for batch in val_loader:
            batch_edges = val_edges[:, batch]
            labels = val_labels[batch]
            outputs = model(data, batch_edges)
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            preds = torch.sigmoid(outputs) > 0.5
            all_preds.extend(preds.cpu().numpy())
            all_true.extend(labels.cpu().numpy())
        avg_val_loss = val_loss / len(val_loader)
        val_losses.append(avg_val_loss)
        accuracy = (np.array(all_preds) == np.array(all_true)).mean()
        val_accuracy.append(accuracy)
    if epoch % 10 == 0 or epoch == 1:
        print(f'Epoch {epoch}, Train Loss: {avg_train_loss:.4f}, Val Loss: {avg_val_loss:.4f}, Val Accuracy: {accuracy:.4f}')
torch.save(model.state_dict(), save_path)
```

Рисунок 3.13 – Оновлення параметрів моделі

3.4 Аналіз отриманих результатів

На основі даних, наведених у таблиці 3.1, можна оцінити ефективність навчання моделі та її здатність прогнозувати оптимальні маршрути в мережі. Оцінка базується на показниках втрати (Loss), точності (Accuracy), а також метрик Precision, Recall і F1 Score.

Зменшення втрати є одним із ключових показників успішного навчання моделі. На початкових етапах (епоха 10) втрата складає 0.3091, що вказує на значні помилки моделі під час прогнозування. Протягом тренування значення втрати поступово зменшується, досягаючи найнижчого значення 0.0215 на 180-й епосі. Це свідчить про те, що модель успішно навчається, а її прогнози стають дедалі точнішими.

Таблиця 3.1

Відображення отриманих результатів

Епоха	Втрата	Точність	Precision	Recall	F1 Score
10	0.3091	0.6909	0.7564	0.6870	0.6720
20	0.2873	0.7127	0.8010	0.7770	0.7778
30	0.1028	0.8972	0.8050	0.7780	0.7796
40	0.0878	0.9122	0.8020	0.7820	0.7832
50	0.0451	0.9549	0.8074	0.7910	0.7921
60	0.0338	0.9662	0.8120	0.7950	0.7961
70	0.0391	0.9609	0.8147	0.8000	0.8018
80	0.0364	0.9636	0.8130	0.7970	0.7981
90	0.0367	0.9633	0.8135	0.7980	0.7998
100	0.0404	0.9596	0.8138	0.7980	0.7998
110	0.0240	0.9760	0.8118	0.7960	0.7983
120	0.0425	0.9575	0.8125	0.8000	0.8014
130	0.0374	0.9626	0.8155	0.8020	0.8040
140	0.0257	0.9743	0.8108	0.7960	0.7979
150	0.0385	0.9615	0.8129	0.8010	0.8021
160	0.0413	0.9587	0.8135	0.7990	0.8008
170	0.0280	0.9720	0.8160	0.8040	0.8055
180	0.0215	0.9785	0.8117	0.7990	0.8004
190	0.0225	0.9775	0.8176	0.7990	0.8013
200	0.0336	0.9664	0.8115	0.8020	0.8026

Точність значно зростає протягом навчання. З 19.09% на 10-й епосі вона підвищується до 97.85% на 180-й епосі, що є показником високої якості прогнозів моделі. Відсутність значного зниження точності на останніх етапах (після 180-ї епохи) свідчить про стабільність моделі та відсутність перенавчання.

Precision залишається високим протягом усього навчання, поступово зростаючи від 0.7564 (епоха 10) до 0.8176 (епоха 190). Це свідчить про те, що модель рідко визначає неправильні ребра як частину маршруту.

Recall також демонструє позитивну динаміку, підвищуючись від 0.6870 на початку навчання до 0.8040 на 170-й епосі. Високі значення Recall показують, що модель майже не пропускає правильні ребра в маршрутах.

F1 Score, як збалансований показник між Precision і Recall, зростає з 0.6720 до 0.8055, що відображає загальну якість класифікації моделі.

На останніх епохах (після 180-ї) спостерігається стабілізація результатів. Метрики Precision, Recall і F1 Score змінюються незначно, що свідчить про досягнення моделлю свого потенціалу навчання.

Модель демонструє ефективне навчання з високими показниками точності, Precision, Recall і F1 Score. Вона добре справляється з класифікацією ребер навіть у складних умовах. Низька втрата та висока точність після 180-ї епохи вказують на те, що подальше навчання може бути неефективним і що модель готова до застосування у практичних задачах прогнозування маршрутів у мережах.

Висновки за розділом 3

Побудована модель здатна точно прогнозувати оптимальні маршрути в складних мережевих топологіях.

Використання отриманих результатів дозволяє оптимізувати маршрутизацію трафіку, підвищити продуктивність і надійність мережі, а також ефективно управляти ресурсами в програмно-конфігурованих мережах (SDN).

Модель може бути використана для моніторингу та управління трафіком у реальних мережах, особливо за умов великих навантажень або потенційних атак.

ВИСНОВКИ

У результаті виконаного дослідження було розроблено програмну модель для оптимізації маршрутизації в комп'ютерних мережах з використанням графових нейронних мереж (GNN). Робота довела, що GNN здатні ефективно вирішувати складні задачі маршрутизації, враховуючи динамічні зміни в мережі та складну топологію.

Порівняння традиційних методів маршрутизації, таких як алгоритми Дейкстри і Беллмана-Форда, з новітніми підходами на основі GNN показало значні переваги останніх у контексті адаптивності до змін, масштабованості та обробки великих і складних графів. Використання GNN дозволило значно підвищити ефективність мережі, знижуючи затримки, покращуючи пропускну здатність і забезпечуючи рівномірний розподіл навантаження серед вузлів.

Особливо важливим є те, що розроблена модель здатна адаптувати маршрути до змін в умовах реального часу, що є критично важливим для сучасних комп'ютерних мереж, де умови можуть змінюватися швидко і непередбачувано. Інтеграція цієї моделі з програмно-визначеними мережами (SDN) та використання інструментів, таких як Mininet і PyTorch Geometric, дозволили провести ефективне тестування та валідацію розробленої системи.

У підсумку, дослідження показує, що використання графових нейронних мереж в задачах оптимізації маршрутизації відкриває нові можливості для підвищення ефективності та надійності комп'ютерних мереж, що особливо важливо для масштабних і складних мережевих систем сучасності.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Карапетян А. Р. Особливості адаптивної маршрутизації в порівнянні із статичною або динамічною // Вісник Черкаського державного технологічного університету. – Черкаси, 2020. – № 2. – С. 45–52.

2 Методи маршрутизації для забезпечення якості мережного сервісу // Науковий вісник Національного університету «Полтавська політехніка». – Полтава, 2019. – № 3. – С. 33–40.

3 Радіотехніка та телекомунікації // Журнал «Вісник Національного технічного університету України «Київський політехнічний інститут». Серія: Радіотехніка». – Київ, 2018. – № 4. – С. 12–19.

4 Вікіпедія. Маршрутизація. URL: https://uk.wikipedia.org/wiki/Маршрутизація (дата звернення: 01.09.2024).

5 EITCA. Які переваги динамічної маршрутизації над статичною в комп’ютерних мережах? URL: https://eitca.org/uk/які-переваги-динамічної-маршрутизації-над-статичною-в-компютерних-мережах/ (дата звернення: 01.09.2024).

6 VPN Unlimited. Що таке Маршрутизація - Термінологія та визначення у сфері кібербезпеки. URL: https://www.vpnunlimited.com/uk/help/glossary/routing (дата звернення: 01.09.2024).

7 Studfiles. Лекція 7. Методи маршрутизації. URL: https://studfile.net/preview/5200774/page:8/ (дата звернення 04.09.2024)

8 Санчес-Ленгелінг Б., Речардс Б., Вілсон Д., Досовіцкій А., Гіршманн М., Картер Ш. Легке введення в графові нейронні мережі// Distill. 2021. URL: https://distill.pub/2021/gnn-intro/ (дата звернення: 05.09.2024).

9 Кіпф Т. Н., Веллінг М. Напівконтрольована класифікація з використанням графових згорткових мереж. 2017. URL: https://arxiv.org/abs/1609.02907 (дата звернення: 05.09.2024).

10 Величкович П. та ін. Графові мережі уваги. 2018. URL: https://arxiv.org/abs/1710.10903 (дата звернення: 05.09.2024).

11 Гамільтон В., Їнг З., Лесковец Дж. Ідуктивне навчання представлень на великих графах. 2017. URL: https://arxiv.org/abs/1706.02216 (дата звернення: 06.09.2024).

12 Riba P., Dellen B., Ponsa D., Villanueva J. Learning Graph Distances with Message Passing Neural Networks // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2018. Vol. 40, № 9. P. 2135–2146.

13 Li Y., Gu M., Dullien T., Vinyals O., Kohli P. Graph Matching Networks for Learning the Similarity of Graph Structured Objects // Proceedings of the 36th International Conference on Machine Learning. 2019. P. 3835–3845.

14 Zhang X., Zhu S., Zhang W., Zhang J. Attributed Graph Clustering via Adaptive Graph Convolution // Proceedings of the 28th International Joint Conference on Artificial Intelligence. California: International Joint Conferences on Artificial Intelligence Organization, 2019. P. 4327–4333.

15 Ying R., You J., Morris C., Ren X., Hamilton W., Leskovec J. Hierarchical Graph Representation Learning with Differentiable Pooling // Advances in Neural Information Processing Systems. 2018. Vol. 31. P. 4800–4810.

16 Tsitsulin A., Mottin D., Karras P., Bronstein A., Müller E. Graph Clustering with Graph Neural Networks // Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2020. P. 100–110.

17 Bojchevski A., Shchur O., Zügner D., Günnemann S. NetGAN: Generating Graphs via Random Walks // Proceedings of the 35th International Conference on Machine Learning. 2018. P. 610–619.

18 Nowak A., Miłkowski P., Stroiński M. Revised Note on Learning Quadratic Assignment with Graph Neural Networks // IEEE Access. 2018. Vol. 6. P. 35333–35341.

19 Zhou J., Cui G., Zhang Z., Yang C., Liu Z., Wang L., Li C., Sun M. Graph Neural Networks: A Review of Methods and Applications // AI Open. 2020. Vol. 1. P. 57–81.

20 Battaglia P., Pascanu R., Lai M., Rezende D.J., Kavukcuoglu K. Interaction Networks for Learning about Objects, Relations and Physics // Advances in Neural Information Processing Systems. 2016. Vol. 29. P. 4502–4510.

21 Duvenaud D., Maclaurin D., Aguilera-Iparraguirre J., Gómez-Bombarelli R., Hirzel T., Aspuru-Guzik A., Adams R.P. Convolutional Networks on Graphs for Learning Molecular Fingerprints // Advances in Neural Information Processing Systems. 2015. Vol. 28. P. 2224–2232.

22 Do K., Tran T., Venkatesh S. Graph Transformation Policy Network for Chemical Reaction Prediction // Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. New York, NY, USA: ACM, 2019. P. 750–760.

23 Fout A., Byrd J., Shariat B., Ben-Hur A. Protein Interface Prediction using Graph Convolutional Networks // Advances in Neural Information Processing Systems. 2017. Vol. 30. P. 6530–6539.

24 Zitnik M., Agrawal M., Leskovec J. Modeling Polypharmacy Side Effects with Graph Convolutional Networks // Bioinformatics. 2018. Vol. 34, № 13. P. i457–i466.

25 Rhee S., Seo S., Kim S. Hybrid Approach of Relation Network and Localized Graph Convolutional Filtering for Breast Cancer Subtype Classification // Proceedings of the 27th International Joint Conference on Artificial Intelligence. California: International Joint Conferences on Artificial Intelligence Organization, 2018. P. 3527–3534.

26 Cui Z., Henrickson K., Ke R., Wang Y. Traffic Graph Convolutional Recurrent Neural Network: A Deep Learning Framework for Network-Scale Traffic Learning and Forecasting // IEEE Transactions on Intelligent Transportation Systems. 2020. Vol. 21, № 11. P. 4883–4894.

27 Berg R., Kipf T.N., Welling M. Graph Convolutional Matrix Completion // arXiv preprint arXiv:1706.02263. 2017.

28 Wu Q., Sun L., Fu Y., Hong R., Wang M. Dual Graph Attention Networks for Deep Latent Representation of Multifaceted Social Effects in Recommender Systems // Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval. New York, NY, USA: ACM, 2019. P. 209–218.

29 Peng H., He X., Liu W., Gao M., Liu W., Li X. Large-Scale Hierarchical Text Classification with Recursively Regularized Deep Graph-CNN // Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. New York, New York, USA: ACM Press, 2018. P. 2357–2366.

30 Zhang Y., Liu Q., Song L. Sentence-State LSTM for Text Representation // Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics. Stroudsburg, PA, USA: Association for Computational Linguistics, 2018. P. 317–327.

31 Bastings J., Titov I., Aziz W., Marcheggiani D., Sima'an K. Graph Convolutional Encoders for Syntax-Aware Neural Machine Translation // Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing. Stroudsburg, PA, USA: Association for Computational Linguistics, 2017. P. 1957–1967.

32 Miwa M., Bansal M. End-to-End Relation Extraction using LSTMs on Sequences and Tree Structures // Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics. Stroudsburg, PA, USA: Association for Computational Linguistics, 2016. P. 1105–1116.

33 Nguyen T.H., Grishman R. Graph Convolutional Networks with Argument-Aware Pooling for Event Detection // Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence. New Orleans, 2018. C. 5900–5907

34 Zhou J., Han X., Yang C., Liu Z., Wang L., Li C., Sun M. GEAR: Graph-Based Evidence Aggregating and Reasoning for Fact Verification // Proceedings of the

57th Annual Meeting of the Association for Computational Linguistics. Florence, 2019. С. 892–901.

35 Song L., Wang Z., Hamza W., Zhang Y., Gildea D. Exploring Graph-Structured Passage Representation for Multi-hop Reading Comprehension with Graph Neural Networks // arXiv preprint arXiv:1809.02040. 2018.

36 Wang Z., Chen T., Ren J., Yu W., Cheng H., Lin L. Deep Reasoning with Knowledge Graph for Social Relationship Understanding // Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence. California: International Joint Conferences on Artificial Intelligence Organization, 2018. С. 1021–1028.

37 Garcia V., Bruna J. Few-Shot Learning with Graph Neural Networks // arXiv preprint arXiv:1711.04043. 2018.

38 Teney D., Liu L., Van Den Hengel A. Graph-Structured Representations for Visual Question Answering // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017. С. 1–9.

39 Hu H., Gu J., Zhang Z., Dai J., Wei Y. Relation Networks for Object Detection // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018. С. 3588–3597.

40 Qi S., Wang W., Jia B., Shen J., Zhu S. Learning Human-Object Interactions by Graph Parsing Neural Networks // arXiv preprint arXiv:1808.07962. 2018.

41 Chen X., Shrivastava A., Gupta A. Iterative Visual Reasoning Beyond Convolutions // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018. С. 7239–7248.

42 Liang X., Xu C., Shen X., Yang J., Liu S., Lin L. Semantic Object Parsing with Graph LSTM // European Conference on Computer Vision. 2016. С. 125–143.

43 Matsunaga D., Suzumura T., Takahashi T. Exploring Graph Neural Networks for Stock Market Predictions with Rolling Window Analysis // arXiv preprint arXiv:1909.10660. 2019.

44 Rusek K., Suárez-Varela J., Mestres A., Barlet-Ros P., Cabellos-Aparicio A. Unveiling the Potential of Graph Neural Networks for Network Modeling and Optimization in SDN // Proceedings of the 2019 ACM Symposium on SDN Research. New York, NY, USA: ACM, 2019. С. 140–151.

45 Khalil E., Dai H., Zhang Y., Dilkina B., Song L. Learning Combinatorial Optimization Algorithms over Graphs // arXiv preprint arXiv:1704.01665. 2017.

46 Allamanis M., Brockschmidt M., Khademi M. Learning to Represent Programs with Graphs // arXiv preprint arXiv:1711.00740. 2017.

47 PyG Documentation // PyTorch Geometric. URL: https://pytorch-geometric.readthedocs.io/en/latest/?utm_source=chatgpt.com. (дата звернення: 09.09.2024)

48 Ryu SDN Framework: Getting Started. Ryu Documentation. URL: https://ryu.readthedocs.io/en/latest/getting_started.html (дата звернення: 13.09.0 2024)

49 Приклад знаходження дерева мінімальної вартості для орієнтованого графа за алгоритмом Дейкстри / Ростислав Верещак // Оптимальні каркаси та шляхи. 2013. 22 жовтня. URL: https://www.mathros.net.ua/pryklad-znahodzhennja-dereva-minimalnoi-vartosti-dlja-orijentovanogo-grafa-za-algorytmom-dejkstry.html (дата звернення: 02.10.2024).

50 Kipf, T. N., & Welling, M., Semi-Supervised Classification with Graph Convolutional Networks, ICLR (2017); doi:10.48550/arXiv.1609.02907

51 Veličković P., Cucurull G., Casanova A., Romero A., Liò P., Bengio Y. Graph Attention Networks // arXiv preprint arXiv:1710.10903. 2017. URL: https://arxiv.org/abs/1710.10903 (дата звернення: 04.10.2024).

ДОДАТКИ

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) ***Магістр***
Галузь знань: 12 – Інформаційні технології
Спеціальність: 123 – Комп'ютерна інженерія
Освітня програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
в.о. завідувача кафедри комп'ютерних
систем та робототехніки
к. ф.-м. н., доц. ХРУСЛОВ М. М.
«12» вересня 2024року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Літвінов Григорій Віталійович
(прізвище, ім'я, по батькові студента)

1. Тема роботи **Модель передачі даних у комп'ютерних мережах на основі графових нейронних мереж**

керівник роботи Стрілець Вікторія Євгенівна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від ***№ 4101-5/3657 від 12 листопада 2024 року***

2. Строк подання студентом роботи ***30 листопада 2024 року***

3. Перелік питань, які потрібно розробити
1. Аналіз сучасних методів оптимізації маршрутизації в мережах передачі даних.
2. Розробка алгоритму побудови графової моделі мережі та обробки даних для навчання GNN.
3. Аналіз можливостей застосування моделі.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Проведення літературного огляду з проблематики оптимізації маршрутизації в мережах передачі даних.	30.10.2023 — 06.03.2024
2	Огляд методів маршрутизації в мережах передачі даних та застосування графових нейронних мереж.	06.03.2024 — 01.04.2024
3	Підбір та аналіз даних для навчання моделей оптимізації маршрутизації.	01.04.2024 — 27.06.2024
4	Розробка та налаштування моделі на основі графових нейронних мереж для оптимізації маршрутизації.	20.06.2024 — 24.08.2024
5	Аналіз продуктивності моделі на тестових даних, порівняння з традиційними методами за ключовими метриками.	15.08.2024 — 17.09.2024
6	Розробка рекомендацій щодо впровадження моделі оптимізації маршрутизації в мережі передачі даних.	17.08.2024 — 20.09.2024
7	Оформлення пояснювальної записки	01.09.2024 — 27.10.2024
8	Представлення кваліфікаційної роботи керівнику та рецензенту	28.10.2024 - 28.11.2024

5. Дата видачі завдання ***05 вересня 2024року.***

Студент Літвінов Г.В. _____
ініціали, прізвище підпис

Керівник роботи Стрілець В.Є. _____
ініціали, прізвище підпис

Додаток Б

Затверджую

_____

«_____»_____2024 р.

Технічне завдання

на розробку програмного виробу «МОДЕЛЬ ПЕРЕДАЧІ ДАНИХ У КОМП'ЮТЕРНИХ МЕРЕЖАХ НА ОСНОВІ ГРАФОВИХ НЕЙРОННИХ МЕРЕЖ»

1.	Введення	1.1. Назва: МОДЕЛЬ ПЕРЕДАЧІ ДАНИХ У КОМП'ЮТЕРНИХ МЕРЕЖАХ НА ОСНОВІ ГРАФОВИХ НЕЙРОННИХ МЕРЕЖ 1.2. Галузь застосування: програмно-конфігуровані мережі (SDN), телекомунікаційні системи, центри обробки даних, мережі Інтернет-провайдерів.
2.	Підстава для розробки	2.1. Навчальний план за спеціальністю 123 – Комп’ютерна інженерія. 2.2. Завдання на кваліфікаційну роботу магістра № ***4101-5/3657*** від ***12 листопада 2024 року*** (представити як Додаток А до пояснювальної записки до кваліфікаційної роботи).
3.	Призначення розробки	3.1. Мета розробки: розробка програмної моделі для оптимізації маршрутизації у мережах передачі даних за допомогою графових нейронних мереж. 3.2. Надає можливість ефективно прогнозувати маршрути в комп'ютерних мережах, враховуючи поточну топологію та статистику трафіку.

		3.3. Вихідні дані розробки: є точні прогнози маршрутів між вузлами мережі, засновані на обробленій топології та статистиці трафіку.
4.	Технічні вимоги до програмного виробу	4.1. Вимоги до функціональних характеристик: модель повинна надавати можливість прогнозувати оптимальні маршрути між вузлами мережі на основі поточної топології та статистики трафіку, враховуючи пропускну здатність каналів і навантаження. 4.2. Вимоги до надійності: Прогнозування маршрутів та обробка статистики повинні виконуватися в реальному часі без значних затримок. 4.3.Вимоги до умов експлуатації: немає 4.4. Вимоги до складу і параметрів технічних засобів: Процесор: Мульти-ядровий процесор (мінімум 4 ядра, рекомендовано 8 ядер) з тактовою частотою 2.5 GHz або вище для ефективної обробки запитів і моделювання маршрутизації в реальному часі. Оперативна пам'ять (RAM): Мінімум 16 GB для забезпечення безперебійної роботи при обробці великих обсягів даних, з можливістю масштабування для великих мереж. Дискова система: SSD з ємністю не менше 512 GB для швидкого зчитування/запису даних і забезпечення високої продуктивності при роботі з базами даних та логами. Мережевий адаптер: Швидкість з'єднання не менше 1 Gbps для забезпечення стабільного потоку даних між компонентами мережі.

		4.5. Вимоги до інформаційної та програмної сумісності: сумісність з Ubuntu 20.04 Python, PyTorch, а також з іншими бібліотеками для обробки природної мови. 4.6. Вимоги до маркування та упаковки: немає 4.7. Вимоги до транспортування і зберігання: на цифрових носіях інформації або у вигляді хмарного програмного продукту. 4.8. Спеціальні вимоги: не передбачені.
5.	Вимоги до програмної документації	Програмною документацією до виробу «Модель передачі даних у комп'ютерних мережах на основі графових нейронних мереж» вважати: 1) Справжнє Технічне завдання на розробку виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи). 2) Методику дослідження мережевих даних і навчання моделі (представити в розділах 2 та 3 пояснювальної записки до кваліфікаційної роботи). 3) Опис моделі та архітектури (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи).
6.	Вимоги до техніко-економічних показників	Програмною документацією до виробу «Модель передачі даних у комп'ютерних мережах на основі графових нейронних мереж» вважати: 1) Справжнє Технічне завдання на розробку виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи). 2) Методику розробки та налаштування архітектури (у вигляді глав 3.3 та 3.4 пояснювальної записки до кваліфікаційної роботи).

		3) Опис виробу (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи)	
7.	Стадії і етапи розробки	Дата	Назва етапу
		від 2 жовтня 2024 до 16 жовтня 2024	Аналіз практики предметної області.
		від 2 жовтня 2024 до 2 листопада 2024	Аналіз існуючих моделей та алгоритмів.
		від 3 листопада 2024 до 30 листопада 2024	Підготовка даних для навчання моделі
		від 10 листопада 2024 до 17 листопада 2024	Розробка архітектури моделі.
		від 12 листопада 2024 до 21 листопада 2024	Навчання моделі.
		від 1 вересня 2024 до 21 вересня 2024	Тестування та оптимізація моделі.
		від 22 вересня 2024 до 15 жовтня 2024	Оформлення результатів. Написання пояснювальної записки.
		від 16 листопада 2024 до 21 листопада 2024	Представлення кваліфікаційного проекту керівнику кваліфікаційної роботи та рецензенту.

8.	Порядок контролю і приймання програмного продукту (моделі)	**1.** Перевірку ходу розробки програми виконувати раз в 3 тижні. **2.** Захист розробленої моделі провести на засіданні Атестаційної комісії. Пояснювальну записку подати на паперових носіях в 1 примірнику і в електронному вигляді в 1 примірнику.

Виконавець
студент групи КІ- 61
Літвінов Г. В.

Замовник
к. т. н.,
Стрілець В. Є . _____

Додаток В

Програма і методика випробувань програмного виробу

«МОДЕЛЬ ПЕРЕДАЧІ ДАНИХ У КОМП'ЮТЕРНИХ МЕРЕЖАХ НА ОСНОВІ ГРАФОВИХ НЕЙРОННИХ МЕРЕЖ»

1 Об'єкт випробувань

Назва програмного виробу : «МОДЕЛЬ ПЕРЕДАЧІ ДАНИХ У КОМП'ЮТЕРНИХ МЕРЕЖАХ НА ОСНОВІ ГРАФОВИХ НЕЙРОННИХ МЕРЕЖ»

Галузь застосування : Автоматизація та приладобудування

Перераховані відомості запозичуються з відповідних розділів Технічного завдання.

2. Мета випробувань

Перевірка відповідності функціональності програмної реалізації системи заявленим функціональним можливостям в технічному завданні (Додаток Б до пояснювальної записки до кваліфікаційної роботи).

3. Загальні положення

Підстави для проведення випробувань

Підставою для проведення випробувань є наказ про призначення атестаційної комісії.

Місце і тривалість випробувань

Приймальні (приймально-здавальні) випробування проводяться на базі комп'ютерного класу кафедри в період роботи атестаційної комісії.

Обсяг випробувань

Приймальні випробування програмного вироби проводяться в обсязі відповідному цієї програми і методики випробувань.

Організації, які беруть участь у випробуваннях

Приймальні випробування проводяться атестаційною комісією напередодні засідання (або в процесі засідання) за участю Замовника, Виконавця та інших осіб, присутніх на засіданні.

4. Вимоги до програми або програмного виробу

4. Вимоги до програми або програмного виробу

- Модель повинна задовольняти наступним вимогам:
- працювати на основних операційних системах: Linux Ubuntu 20.04;
- вимоги до надійності;
- передбачити захист від некоректних дій користувача;
- сумісність з іншими програмними продуктами;
- бути легко розширюваною;
- елементи програми повинні бути ізольовані одне від одного для зменшення їх впливу на роботи програми під час редагування програмного коду;
- вимоги до складу і параметрів технічних засобів;
- вимоги до маркування та упаковки (не висуваються);
- вимоги до транспортування і зберігання (не висуваються).
- Спеціальні вимоги (не пред’являються).

5. Вимоги до програмної документації

Програмною документацією до виробу «» вважати:

Програмною документацією щодо розроблюваного програмного продукту вважати:

справжнє технічне завдання на розробку програми (представити як Додаток Б до пояснювальної записки до кваліфікаційної роботи);

Програму і методику випробувань розробленої програми (представити як Додаток В до пояснювальної записки до кваліфікаційної роботи);

рекомендацій щодо застосування створеної програмної стандартизації у проектах (представити в Розділі 3 пояснювальної записки до кваліфікаційної роботи).

Текст програми(представити як Додаток Г до пояснювальної записки до кваліфікаційної роботи).

6. Засоби і порядок випробувань

6.1 Засоби випробувань

Для проведення випробувань необхідний проект для розробки моделі на мові програмування python.

6.2 Порядок проведення випробувань

Як правило, випробування проводяться в два етапи:

-ознайомчий (1-й етап);

-випробування програмного виробу (2-й етап).

Перелік перевірок, що проводяться на 1 етапі випробувань, включає в себе:

Перевірку комплектності програмної документації.

Перевірка комплектності складу програмної документації здійснюється за критерієм наявності зазначеної в ТЗ документації.

Перевірку комплектності складу технічних і програмних засобів.

Методику проведення перевірок на 1 етапі випробувань.

Якість програмної документації перевіряється на відповідність вимогам стандартів ЕСПД.

Перелік перевірок, що проводяться на 2 етапі випробувань, включає в себе:

перевірку відповідності технічних характеристик програми вимогам технічного завдання;

перевірку ступеня виконання функціональних вимог до програми;

методику проведення перевірок, що входять до переліку по 2 етапу випробувань.

Програма працює відповідно до умов експлуатації операційних систем Linux Ubuntu 20.04.

Для роботи необхідний компілятор мови програмування python, версії не нижчої ніж 8.0

Порядок проведення випробувань:

Запуск програми здійснюється за допомогою через консоль за допомогою команди python: “python ryu-manager <ім’я файлу>.py”;

Для перевірки функціональності програми пропонується виконати такі тести.

Тест 1: Прогнозування маршрутів

Передумови: середнє навантаження мережі

- **Дії:**

1. Відкрити консоль в папці де знаходиться програма.

2. Система починає змінювати маршрути виходячи з завантаженості мережі.

- **Очікуваний результат:**

1. Зменшення навантаження мережі та вивід прогнозованих оптимальних маршрутів.

2. Збереження топології та поточного стану ребр.

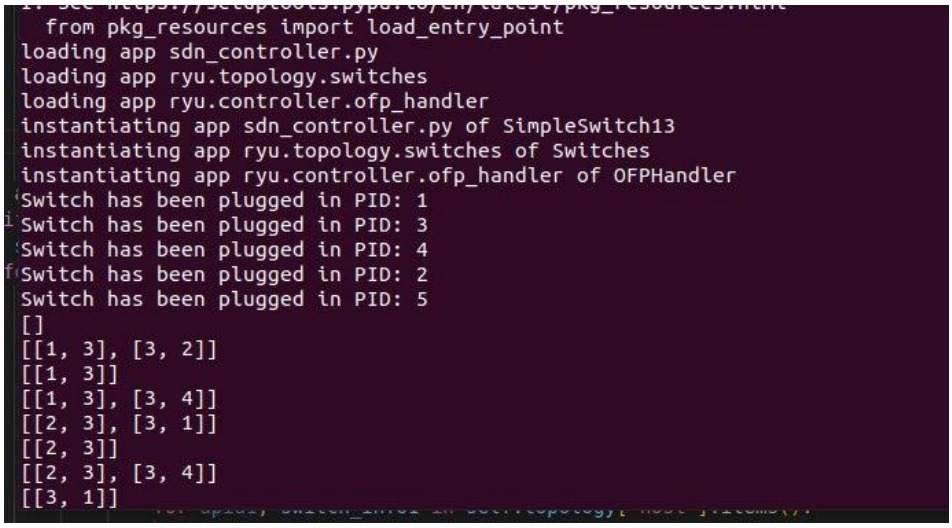


Рисунок В. 1 – Відображення зміни маршруту

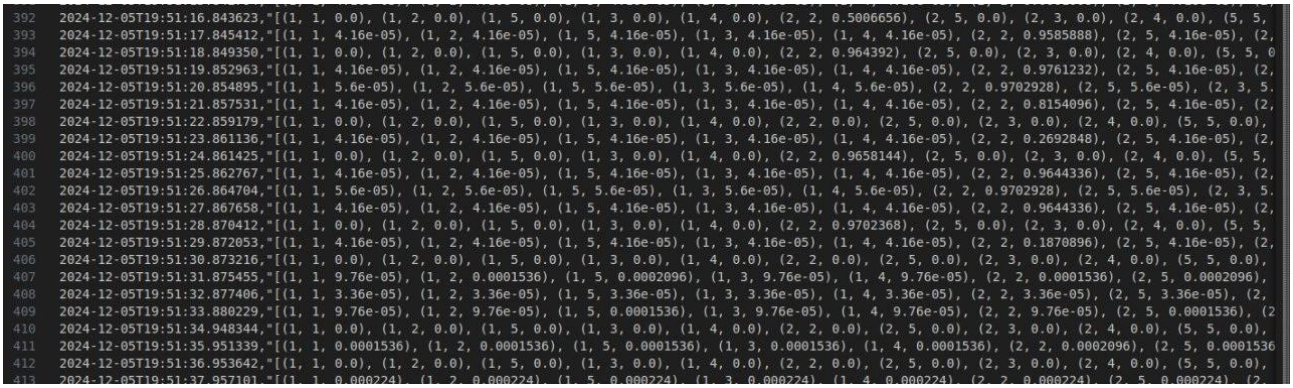


Рисунок В. 2 – Відображення запису в файлі стану ребр

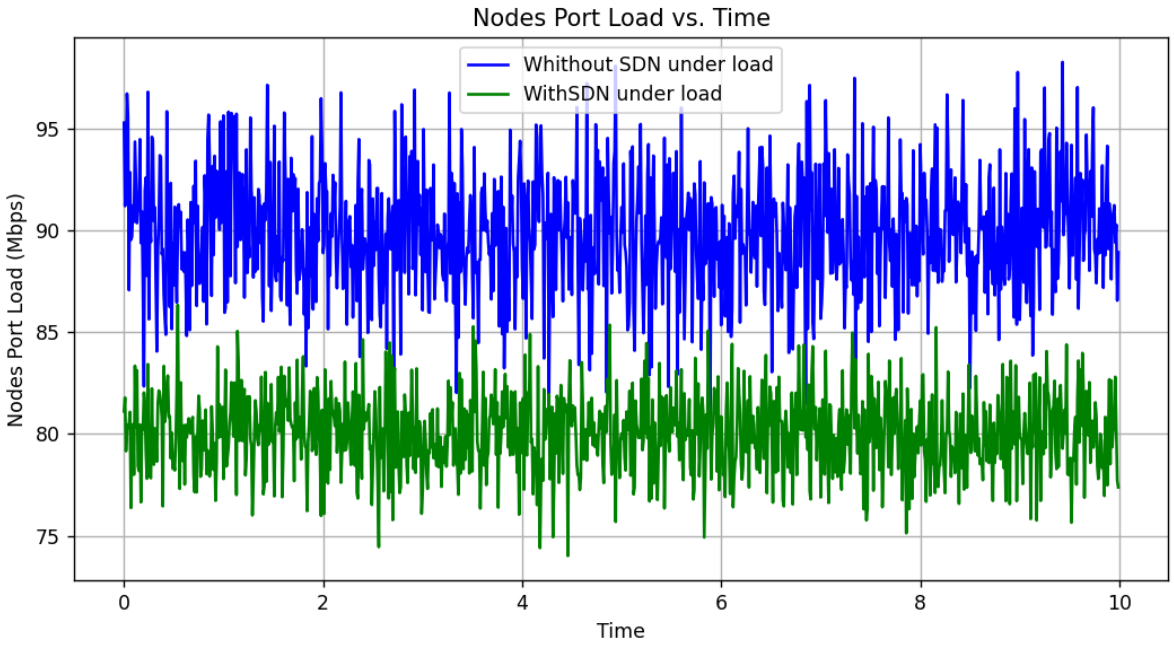


Рисунок В. 3 – Тест на навантаження протягом 10 хвилин

Висновки: тест виконано успішно

Виконавець: студентка групи КІ-61, Літвінов Г. В. _____

Додаток Г

Лістинг коду «Модель знаходження оптимального маршруту»

```
import os
import pandas as pd
import ast
import torch
from torch_geometric.data import Data
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.nn import GCNConv
import torch.optim as optim
import networkx as nx
import heapq
import json
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from collections import Counter
import numpy as np

# 1. Підготовка Даних

def load_data(file_path):
    """
    Завантаження даних з CSV файлу з timestamp та ребрами.

    Параметри:
    - file_path: Шлях до CSV файлу.

    Повертає:
    - DataFrame з розпарсеними ребрами.
    """
    df = pd.read_csv(file_path, header=None, names=['timestamp', 'edges'])
    df['edges'] = df['edges'].apply(ast.literal_eval)
    return df

# 2. Побудова Графу

def build_graph(latest_snapshot, bandwidth_max):
    """
    Побудова PyTorch Geometric Data об'єкту з останнього знімку.

    Параметри:
    - latest_snapshot: Рядок з DataFrame, що містить ребра.
    - bandwidth_max: Максимальна пропускна здатність для нормалізації.
```

```
    Повертає:
    - data: PyTorch Geometric Data об'єкт.
    - node_list: Список ідентифікаторів вузлів.
    - node_id_map: Відображення вузлів на індекси.
    """
    edges = latest_snapshot['edges']

    # Виділення унікальних вузлів
    nodes = set()
    for edge in edges:
        nodes.add(edge[0])
        nodes.add(edge[1])

    node_list = sorted(list(nodes))
    node_id_map = {node: idx for idx, node in enumerate(node_list)}

    # Підготовка edge_index та edge_attr
    edge_index = []
    edge_attr = []

    for edge in edges:
        src, dst, bandwidth = edge
        if src != dst and bandwidth > 0:
            edge_index.append([node_id_map[src], node_id_map[dst]])
            # Нормалізація пропускної здатності
            normalized_bandwidth = bandwidth / bandwidth_max
            edge_attr.append([normalized_bandwidth])

    # Конвертація в тензори
    edge_index = torch.tensor(edge_index, dtype=torch.long).t().contiguous()
    edge_attr = torch.tensor(edge_attr, dtype=torch.float)

    # Створення one-hot ознак для вузлів
    x = torch.eye(len(node_list))

    # Створення Data об'єкту
    data = Data(x=x, edge_index=edge_index, edge_attr=edge_attr)

    return data, node_list, node_id_map

# 3. Модель GNNRoutePredictor

class GNNRoutePredictor(nn.Module):
    def __init__(self, num_node_features, hidden_dim, dropout=0.5):
        """
        Ініціалізація моделі GNNRoutePredictor.
```

```
        Параметри:
        - num_node_features: Кількість вхідних ознак для кожного вузла.
        - hidden_dim: Розмірність прихованих шарів.
        - dropout: Ймовірність Dropout.
        """
        super(GNNRoutePredictor, self).__init__()
        self.conv1 = GCNConv(num_node_features, hidden_dim)
        self.conv2 = GCNConv(hidden_dim, hidden_dim)
        self.dropout = nn.Dropout(p=dropout)
        self.fc = nn.Linear(hidden_dim * 2, 1)  # Для класифікації ребер

    def forward(self, data, edge_index):
        """
        Прямий прохід моделі.

        Параметри:
        - data: Об'єкт PyTorch Geometric Data, що містить граф.
        - edge_index: Індекси ребер для яких прогнозуються мітки.

        Повертає:
        - Прогнози для ребер.
        """
        x, edge_indices = data.x, data.edge_index
        x = self.conv1(x, edge_indices)
        x = F.relu(x)
        x = self.dropout(x)
        x = self.conv2(x, edge_indices)
        x = F.relu(x)
        x = self.dropout(x)

        # Для кожного ребра в edge_index, комбінуємо ембедінги вузлів
        src = edge_index[:, 0]
        dst = edge_index[:, 1]
        src_emb = x[src]
        dst_emb = x[dst]
        combined = torch.cat([src_emb, dst_emb], dim=1)
        out = self.fc(combined).squeeze()
        return out

# 4. Підготовка Тренувальних Даних з Мітками Ребер

def prepare_training_data_with_edge_labels(df, node_id_map, bandwidth_max, data):
    """
    Підготовка тренувальних даних з мітками ребер, що вказують, чи є ребро частиною
оптимального маршруту.
```

```
    Параметри:
    - df: DataFrame з усіма знімками.
    - node_id_map: Відображення вузлів на індекси.
    - bandwidth_max: Максимальна пропускна здатність для нормалізації.
    - data: PyTorch Geometric Data об'єкт.

    Повертає:
    - all_edges: Список всіх ребер у графі.
    - all_labels: Список міток для кожного ребра (1 - частина маршруту, 0 - ні).
    """
    edge_labels = []  # Список для зберігання ребер, що входять до маршрутів

    for index, row in df.iterrows():
        edges = row['edges']
        G = nx.Graph()
        for edge in edges:
            src, dst, bandwidth = edge
            if src != dst and bandwidth > 0:
                G.add_edge(src, dst, bandwidth=bandwidth)

        nodes = list(G.nodes())
        for i in range(len(nodes)):
            for j in range(i+1, len(nodes)):
                src = nodes[i]
                tgt = nodes[j]
                path, _ = maximum_bandwidth_path(G, src, tgt, capacity='bandwidth')
                if path:
                    # Отримуємо ребра на шляху
                    path_edges = list(zip(path[:-1], path[1:]))
                    for edge in path_edges:
                        src_id = node_id_map[edge[0]]
                        dst_id = node_id_map[edge[1]]

                        # Упорядковуємо ребра для уникнення дублювання
                        if src_id < dst_id:
                            edge_tuple = (src_id, dst_id)
                        else:
                            edge_tuple = (dst_id, src_id)
                        edge_labels.append(edge_tuple)

    from collections import Counter
    edge_count = Counter(edge_labels)  # Підрахунок кількості входжень кожного ребра
у маршрути
    total_routes = len(edge_count)  # Загальна кількість унікальних ребер у маршрутах

    all_edges = []
    all_labels = []
```

```
    for idx in range(data.edge_index.size(1)):
        src = data.edge_index[0, idx].item()
        dst = data.edge_index[1, idx].item()
        if src < dst:
            edge_tuple = (src, dst)
        else:
            edge_tuple = (dst, src)
        all_edges.append(edge_tuple)
        label = 1 if edge_tuple in edge_count else 0  # Мітка: 1 - частина маршруту, 0 - ні
        all_labels.append(label)

    return all_edges, all_labels

# 5. Тренування Моделі

def train_model_route_predictor(model, data, edge_labels, train_edges, val_edges,
val_labels, epochs=100, lr=0.001, save_path='model_route_predictor.pth'):
    """
    Тренування моделі GNNRoutePredictor.

    Параметри:
    - model: Екземпляр GNNRoutePredictor.
    - data: PyTorch Geometric Data об'єкт.
    - edge_labels: Список міток ребер.
    - train_edges: Індекси ребер для тренування.
    - val_edges: Індекси ребер для валідації.
    - val_labels: Список міток ребер для валідації.
    - epochs: Кількість епох тренування.
    - lr: Коефіцієнт навчання.
    - save_path: Шлях для збереження моделі.
    """
    optimizer = optim.Adam(model.parameters(), lr=lr)
    criterion = nn.BCEWithLogitsLoss()

    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model = model.to(device)

    data = data.to(device)
    train_edges = torch.tensor(train_edges, dtype=torch.long).t().contiguous().to(device)
    val_edges = torch.tensor(val_edges, dtype=torch.long).t().contiguous().to(device)
    edge_labels = torch.tensor(edge_labels, dtype=torch.float).to(device)
    val_labels = torch.tensor(val_labels, dtype=torch.float).to(device)

    # Поділ на батчі
    batch_size = 1024
    num_train = train_edges.size(1)
    num_val = val_edges.size(1)
```

```
    train_indices = torch.randperm(num_train)
    val_indices = torch.randperm(num_val)

    train_loader = torch.utils.data.DataLoader(train_indices, batch_size=batch_size,
shuffle=True)
    val_loader = torch.utils.data.DataLoader(val_indices, batch_size=batch_size,
shuffle=False)

    train_losses = []
    val_losses = []
    val_accuracy = []

    for epoch in range(1, epochs + 1):
        model.train()
        epoch_loss = 0
        for batch in train_loader:
            optimizer.zero_grad()
            batch_edges = train_edges[:, batch]
            labels = edge_labels[batch]
            outputs = model(data, batch_edges)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            epoch_loss += loss.item()

        avg_train_loss = epoch_loss / len(train_loader)
        train_losses.append(avg_train_loss)

        # Валідація
        model.eval()
        with torch.no_grad():
            val_loss = 0
            all_preds = []
            all_true = []
            for batch in val_loader:
                batch_edges = val_edges[:, batch]
                labels = val_labels[batch]
                outputs = model(data, batch_edges)
                loss = criterion(outputs, labels)
                val_loss += loss.item()
                preds = torch.sigmoid(outputs) > 0.5
                all_preds.extend(preds.cpu().numpy())
                all_true.extend(labels.cpu().numpy())

            avg_val_loss = val_loss / len(val_loader)
            val_losses.append(avg_val_loss)
```

```
            accuracy = accuracy_score(all_true, all_preds)
            val_accuracy.append(accuracy)

        if epoch % 10 == 0 or epoch == 1:
            print(f'Epoch {epoch}, Train Loss: {avg_train_loss:.4f}, Val Loss:
{avg_val_loss:.4f}, Val Accuracy: {accuracy:.4f}')

    print("Тренування завершено.")

    # Збереження моделі
    torch.save(model.state_dict(), save_path)
    print(f"Модель збережена за шляхом: {save_path}")

    # Побудова графіків
    plt.figure(figsize=(12, 6))

    # Втрата
    plt.subplot(1, 2, 1)
    plt.plot(range(1, epochs + 1), train_losses, label='Train Loss')
    plt.plot(range(1, epochs + 1), val_losses, label='Val Loss')
    plt.xlabel('Епоха')
    plt.ylabel('Втрата')
    plt.title('Втрата Тренування та Валідації')
    plt.legend()

    # Точність
    plt.subplot(1, 2, 2)
    plt.plot(range(1, epochs + 1), val_accuracy, label='Val Accuracy')
    plt.xlabel('Епоха')
    plt.ylabel('Точність')
    plt.title('Точність Валідації')
    plt.legend()

    # Збереження графіків
    os.makedirs('plots', exist_ok=True)
    plt.tight_layout()
    plt.savefig('plots/training_metrics_route_predictor.png')
    plt.show()
print("Графік метрик збережено за шляхом 'plots/training_metrics_route_predictor.png'")
```