Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Факультет комп'ютерних наук
Кафедра теоретичної та прикладної системотехніки

«Затверджую»
Зав. кафедри теоретичної та
прикладної системотехніки
_____ д.т.н., проф. С. І. Шматков
«___» _____ 2023 р.

Пояснювальна записка

до кваліфікаційної роботи
магістра

на тему: «**МОДЕЛЬ СТРУКТУРНОЇ ОПТИМІЗАЦІЇ ЛОКАЛЬНОЇ КОМП'ЮТЕРНОЇ МЕРЕЖІ**»

Захищено на засіданні
Атестаційної комісії № 40
протокол № __ від __.12.2023 р.
Оцінка _____ / _____
Голова Атестаційної комісії
_____ **СКОБ Ю.О.**
(підпис)

Виконав:
студент 2 курсу, групи КІ – 61
Галузь знань: 12 – Інформаційні технології
Спеціальність: 123 – «Комп'ютерна інженерія»
ЮЩЕНКО Владислав Сергійович ___

Керівник:
к.т.н., доцент кафедри теоретичної та прикладної системотехніки
СТРІЛЕЦЬ Вікторія Євгенівна

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

Харків – 2023

АНОТАЦІЯ

Пояснювальна записка до магістерської кваліфікаційної роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел і чотирьох додатків. Загальний об’єм роботи складає 76 сторінок із них 55 сторінок основної частини із 27 рисунками, 22 найменуваннями списку використаних джерел, та однією таблицею.

Метою роботи є підвищення ефективності роботи локальних комп’ютерних мереж через оптимізацію їх структури.

Під час виконання кваліфікаційної роботи були проаналізовані різні підходи до оптимізації комп’ютерних мереж, але універсального підходу не було визначено. Тому для досягнення мети у роботі було запропоновано об’єднати розглянуті шляхи оптимізації структури в одну модель.

У результаті була побудована модель оптимізації мережі, яка складалась із підходів віртуалізації мережі, для централізованого та полегшеного управління та моніторингу стану локальної мережі та з графової нейронної мережі для прогнозування змін стану локальної мережі.

Ключові слова: програмно-визначена мережа, мережа визначена знаннями, графові нейронні мережі, нейронні мережі передачі повідомлень, RouteNet.

ABSTRACT

The explanatory note to the master's thesis consists of an introduction, three sections, conclusions, a list of used sources and four appendices. The total volume of the work is 76 pages, of which 50 pages are the main part with 27 figures, 22 titles of the list of sources used, and one table. The purpose of the work is to increase the efficiency of local computer networks through the optimization of their structure.

During the qualification work, various optimization approaches were analyzed, but no universal approach was found. Therefore, in order to achieve the goal, it was proposed to combine the considered ways of optimizing the structure into one model.

As a result, a network optimization model was obtained, which consisted of: network virtualization approaches for centralized and simplified management and monitoring of the state of the local network; and from a graph neural network for predicting changes in the state of the local network.

Keywords: software-defined and knowledge-defined network, graph neural networks, message transmission neural networks, RouteNet.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

NFV – віртуалізація функцій мережі;

SDN – програмно-визначена мережа;

KDN – мережа визначена знаннями;

GNN – графова нейронна мережа;

RNN – рекурентна нейронна мережа;

MPNN – нейронна мережа передачі повідомлень.

ВСТУП

Комп'ютерні мережі стали невід’ємною частиною сучасного світу. Вони використовуються для передачі даних, спільної роботи, комунікації та забезпечення доступу до інтернету. Швидкий розвиток технологій і зростання обсягів даних призводять до того, що оптимізація комп'ютерних мереж стає більш актуальною та важливою задачею. Оптимізація мережі дозволяє забезпечити ефективну роботу, знизити витрати ресурсів і покращити якість обслуговування користувачів.

Актуальність роботи. З ростом обсягу даних та залежності від мережевих послуг стає очевидною необхідність вдосконалення структури комп'ютерних мереж. Оптимізація їх структури може призвести до зменшення витрат на обладнання, підвищення швидкості передачі даних, покращення безпеки і надійності мережі. Ця робота ставить за ***мету дослідження*** підвищення ефективності роботи локальних комп’ютерних мереж через розробку і впровадження моделі структурної оптимізації комп'ютерних мереж.

Предмет дослідження – підходи і методи структурної оптимізації локальних комп’ютерних мереж.

Об’єкт дослідження – процес оптимізації структури комп’ютерної мережі.

Для досягнення мети необхідно виконати ***задачі***:

– визначити підходи до структурної оптимізації локальної комп’ютерної мережі;

– провести аналітичний огляд існуючих методів оптимізації; аналітичний огляд існуючих концепцій архітектур мереж;

– описати моделі прогнозування стану комп’ютерної мережі;

– порівняти ефективності прогнозування моделі з реальними даними;

– провести огляд та аналіз отриманих результатів.

РОЗДІЛ 1
ОГЛЯД МОДЕЛЕЙ СТРУКТУРНОЇ ОПТИМІЗАЦІЇ

Оптимізація локальної комп'ютерної мережі може включати в себе різні аспекти, включаючи архітектурні, апаратні та програмні рішення. В даному розділі кваліфікаційної роботи будуть розглянуті моделі та методи, які можуть бути використані для структурної оптимізації локальної мережі.

1.1 Топологічна модель

Топологічна модель локальної комп'ютерної мережі (LAN) описує фізичне розташування та з'єднання пристроїв у мережі. Оптимізація LAN топології може включати в себе розгляд різних факторів, таких як продуктивність, надійність, масштабованість та безпека.

Приклади топологічної моделі оптимізації LAN:

- Зіркова топологія. У зірковій топології (рис. 1.1) всі пристрої підключаються до центрального комутатора або концентратора. Ця топологія є простою для налаштування та підтримки. Оптимізація може включати в себе вибір потужного комутатора та моніторинг навантаження на нього.

Рисунок 1.1 – Зіркова топологія

- Кільцева топологія. У кільцевій топології (рис. 1.2) кожен пристрій підключений до двох сусідніх пристроїв, створюючи замкнене кільце. Оптимізація може включати в себе маршрутизацію та механізми виявлення помилок для підтримки високої надійності.

Рисунок 1.2 – Кільцева топологія

- Деревоподібна топологія. В деревоподібній топології (рис. 1.3) є головний маршрутизатор, до якого комутатори. Оптимізація може включати в себе розподілену обробку даних та моніторинг стану комутаторів.

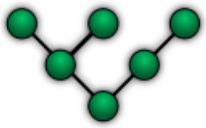

Рисунок 1.3 – Деревоподібна топологія

- Шинна топологія. У шинній топології (рис. 1.4) всі пристрої підключені до одного центрального каналу (шини). Оптимізація може включати в себе резервні канали та методи розділення конфліктуючих даних.

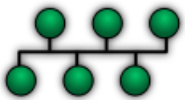

Рисунок 1.4 – Шинна топологія

- Гібридна топологія (рис. 5) поєднує різні види топологій, наприклад, з'єднує зірку та кільце. Оптимізація може включати в себе балансування навантаження та підтримку різних частин мережі.

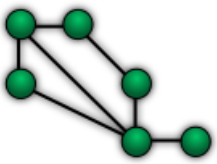

Рисунок 1.5 – Гібридна топологія

- Бездротова топологія. Бездротова LAN базується на бездротових точках доступу (Wi-Fi). Оптимізація може включати в себе вибір пристроїв з більшою пропускною здатністю та діапазоном покриття сигналу та забезпечення кращого рівня безпеки.
- Топологія сітка (рис. 6) або "Меш" (Mesh Topology). У даній топології кожен комп’ютер підключений до всіх інших комп’ютерів в мережі. Оптимізація маршрутизації та зменшення кількості зв’язків для зменшення витрат є можливим шляхом для оптимізації даної системи

Рисунок 1.6 – Топологія сітка

- Топологія "Споживач-Постачальник". В основі мережі знаходяться спеціалізовані сервери, до яких звертаються клієнти для доступу до ресурсів, через спеціальні термінали доступу або ПК. Можливим шляхом оптимізації є покращення продуктивності серверів та оптимізація пропускної здатності каналу зв’язку.

Для оптимізації топології LAN, необхідно враховувати конкретні потреби та вимоги вашої організації або дому, такі як надійність, продуктивність, масштабованість та безпека. Вибір оптимальної топології повинен бути підтриманий відповідними налаштуваннями, апаратними рішеннями та методами безпеки.

1.1.1 Оптимізаційні підходи топологічної моделі

Для вирішення завдань з оптимізації топологій, визначення оптимальних маршрутів, і тому подібне, доцільно представляти топологію мережі у вигляді графу G=(V,E) де множина вершин V відповідає вузлам

мережі, а множина ребер E – каналам. Застосування теорії графів надає можливість провести детальне дослідження мережі, охоплюючи характеристики ребер графу, за допомогою яких можна аналізувати та впливати на такі поняття як: пропускна здатність, завантаженість, чи затримки. Як тільки граф буде відповідати топології комп'ютерної мережі, можна використати метод аналізу ієрархій(MAI) (рис 1.7), для дослідження ефективності фізичної структури, і чи здатна вона забезпечувати задовільний рівень якості надання послуг.

Рисунок 1.7 – Ієрархічне зображення задачі згідно з MAI[1]

Завдяки попарному порівнянню різних структур, можна дійти до рішення, яке буде надавати максимально можливий рівень сервісу. Але цей метод має свої недоліки та обмеження. Вхідні дані про структуру мереж формуються на основі думок мережевих інженерів чи архітекторів. Тобто кількість можливих варіантів залежить від кількості інженерних ідей. Також із збільшенням кількості критеріїв, збільшується не тільки складність обчислення, а і вибору рішення, яке буде задовольнати достатній рівень сервісу. Також перед порівнянням необхідно провести уніфікацію формування потоків та пропускної спроможності ребер.

Але якщо розглядати топологію мережі з точки зору логічної структури, то теорія графів стає потужним інструментом, завдяки якому можна ефективно використовувати ресурси мережі.

При оптимізації логічної структури, спочатку виконується пошук найкоротших шляхів, за допомогою алгоритму Флойда, і формується матриця найкоротших шляхів W і в той же час формується матриця завантаженості ребер R. Так як через одне ребро може проходити декілька різних найкоротших шляхів, матриця R відображає скільки найкоротших шляхів проходить через кожне ребро. Значення завантаженості може приймати значення від 0 до E_{max}.

Чим однорідніша матриця завантаженості тим рівномірніше розподілене навантаження між вузлами мережі. Тому в загальному оптимізацію логічної структури продовжують до тих пір, поки різниця між мінімальним та максимальним значенням навантаження не буде мінімальною.

Коли структура мережі має велику зв’язність, тим простіше буде розподілити навантаження в мережі, завдяки наявності різних обхідних шляхів. Застосовуючи алгоритм Minimum Spanning Tree MST, до матриці навантаженості R, можна визначати потенцій обхідні шляхи. В результаті роботи алгоритму буде сформований граф $G_{MST}(V,E_{min})$ тобто усі вузли графу будуть з’єднанні мінімальною кількістю ребер, які в свою чергу відносяться до найменш навантажених ребер.

1.2 Математична модель оптимізації мережі

Математична модель оптимізації локальної комп'ютерної мережі може бути досить складною і залежить від конкретних цілей та обмежень. Однак основна мета такої моделі - мінімізувати певний параметр чи функцію, такі як: вартість, час затримки, або споживану енергію, з урахуванням різних змінних і обмежень мережі. Загальна структура математичної моделі оптимізації локальної комп'ютерної мережі:

- Параметри (П). Це набір змінних, що описують параметри мережі. Параметри можуть включати розміщення пристроїв, пропускну спроможність

ліній передачі даних, характеристики апаратного забезпечення, інші характеристики мережі.

- Цільова функція (F). Це функція, яка повинна бути оптимізована. Наприклад, це може бути функція вартості, що включає в себе витрати на обладнання та експлуатацію, час затримки передачі даних, споживану енергію тощо.

- Обмеження (C). Обмеження визначаються для врахування різних факторів, які повинні бути задоволені в мережі. Обмеження можуть включати обмеження на пропускну спроможність, максимальну кількість підключених пристроїв, стійкість до відмов тощо.

- Функція оптимізації. Метою оптимізації є знаходження значень параметрів (П), які мінімізують цільову функцію (F) за умов обмежень (C). Оптимізаційні методи можуть бути застосовані для пошуку найкращого рішення.

- Математичні рівняння та нерівності. Математичні співвідношення формалізують зв'язки між параметрами (П), цільовою функцією (F) та обмеженнями (C). Ці рівняння можуть включати лінійні, нелінійні, дискретні чи неперервні залежності.

- Алгоритм оптимізації. Для вирішення математичної моделі використовуються конкретні оптимізаційні алгоритми, такі як методи лінійного програмування, генетичні алгоритми, методи градієнта тощо.

- Розв'язок задачі. Результатом розв'язку задачі оптимізації є набір оптимальних значень параметрів (П), які задовольняють усі обмеження (C) і мінімізують цільову функцію (F).

Математична модель допомагає визначити оптимальну конфігурацію та налаштування для локальної комп'ютерної мережі відповідно до поставлених цілей та обмежень.

1.2.1 Приклад побудови математичної моделі

Нехай для математичної моделі мережі у нас задані:

- Корінь дерева $X_0 = \{x_1^0, \dots, x_k^0\}$
- Множина вузлів мережі АП $\{x_1, x_2, \dots, x_n\}$
- Координати АП $\{\delta_i, \omega_i\}$
- Обсяг генерованої інформації в вузлах АП $h_i, i = 1, n$
- Пропускна здатність каналів зв'язку $D = \{d_1, \dots, d_m\}$
- Вартість каналів зв'язку $C = \{c_1, \dots, c_m\}$
- Множина комутаторів SW = $\{SW_1, SW_2, \dots, SW_j\}$, при чому кожен SW_i включає в себе три змінні: вартість c_i, к-сть портів n_i продуктивність $П_i$

Необхідно знайти місця розміщення комутаторів $Z=\{z_k\}$ і термінальні вузли x_z, що підключаються до них, а також множину вузлів АП, що підключаються прямо до X_0, структуру мережі D^0 і пропускні здатності усіх каналів зв'язку, при яких мінімізується сумарна вартість мережі C_Σ , включаючи вартість усіх каналів зв'язку і всіх комутаторів.

Після того як визначили вхідні дані і ціль оптимізації, першим етапом є визначення змінних. Нехай у нас є п'ять двійкових змінних:

- $\delta_{iZ} = 1$, якщо x_i, підключено до SW який знаходиться в Z
- $\delta_Z = 1$, якщо в Z встановлено SW
- $\delta_i = 1$, якщо x_i підключений до X_0
- $\delta_{Z0} = 1$, якщо SW_z підключений до X_0
- $\delta_{ZZ} = 1$, якщо SW_z підключений до SW в зоні z'

Тобто нам необхідно буде знайти такі параметри, а також структуру мережі D^0 , які будуть мінімізувати цільову функцію:

$$C_\Sigma = \sum_{i=1}^{n} C_{i0}^{\text{пер}}(h_i, l_{i0})\delta_i + \sum_{i=1}^{n}\sum_{z\in Z} C_{iz}^{\text{пер}}(h_i, l_{iz})\delta_{iZ} + \sum_{z\in Z} C_{SW}^{(S)}(H_z)\delta_z$$
$$+ \sum_{z\in Z} C_{z0}^{\text{пер}}(H_z, l_{z0})\delta_{z0} + \sum_{z\in Z}\sum_{z\in Z} C_{zz}^{\text{пер}}(H_z, l_{zz})\delta_{zz} \to min$$

При наступних обмеженнях:

- Обмеження продуктивності комутатора у пункті z: $H_z \leq \Pi_{SWz}^{(S)}$
- Обмеження по числу підключених до SW_Z АП і та інших комутаторів $\sum_{i=1}^{n} \delta_{iz} + \sum_{z \in Z} \delta_{ZZ} \leq n_{SWz}^{(s)}$
- Будь-який АП x_i повине бути підключений або до деякого SW_Z або до кореня X_0 $\sum_{z \in Z} \delta_{iz} + \delta_{i0} = 1$

Наступним етапом стане вирішення даної системи, за допомогою будь-якого підходящого засобу, від простого перебору параметрів, до методів машинного навчання

1.3 Моделі машинного навчання

Модель машинного навчання для оптимізації локальної комп'ютерної мережі може бути корисною для автоматизації процесів моніторингу та управління мережею з метою підвищення її продуктивності та надійності.

Приблизний опис подібної моделі:

- Збір та аналіз даних. Збір даних про стан мережі, включаючи швидкість передачі даних, завантаженість, кількість пакетів, рівень сигналу (для бездротових мереж), стан маршрутизаторів і комутаторів та інші параметри. А також збереження журналів подій, які містять інформацію про помилки та події, які відбуваються в мережі.
- Підготовка та очищення даних. Оброблення та очищення даних, включаючи видалення дублікатів, заповнення пропущених значень, нормалізацію тощо.
- Визначення цільової функції. Визначення цільової функції для оптимізації, наприклад, мінімізація витрат на мережеве обладнання, максимізація швидкості передачі даних або мінімізація відмов.

- Вибір алгоритму машинного навчання. Вибір відповідного алгоритму машинного навчання для завдання оптимізації, такі як: лінійна регресія, дерева рішень, нейронні мережі або інші.
- Тренування моделі. Використання історичні дані для тренування моделі машинного навчання.
- Оцінка та перевірка моделі. Використання метрик, такі як: середньоквадратична помилка (RMSE) чи коефіцієнт детермінації (R-squared), для оцінки точності та ефективності моделі.
- Реалізація дій. Наприклад, на основі прогнозів моделі можна змінювати налаштування маршрутизаторів для підвищення швидкості передачі даних або перерозподіляти навантаження між серверами.
- Моніторинг та навчання моделі. Моніторинг стану мережі та збір даних для навчання моделі на постійній основі.
- Резервне копіювання та відновлення. Забезпечення резервного копіювання моделі та налаштувань для відновлення у випадку відмови або помилки.

Модель машинного навчання може бути використана для автоматизації управління та оптимізації локальної комп'ютерної мережі з метою забезпечення кращої продуктивності та надійності мережі.

1.4 Віртуалізація мереж

Віртуалізація локальної комп'ютерної мережі (LAN) може бути важливим кроком для оптимізації та управління мережею у великих організаціях або компаніях, де важливо забезпечити високий рівень ефективності, безпеки та доступності. Віртуалізаційна модель оптимізації LAN має набір складових (рис.1.8).

- Віртуалізація комутації (Network Virtualization). Використання віртуальних інтерфейсів на комутаторах, які можуть бути програмно

налаштовані та управлятися. Це дозволяє легко масштабувати мережу та налаштовувати маршрутизацію для оптимізації трафіку. Розділення LAN на віртуальні мережі (VLAN) ізолює різні групи пристроїв та зменшить затори.

- Віртуалізація ресурсів (Resource Virtualization). Використання віртуальних серверів та обчислювальних ресурсів для оптимізації використання обладнання. Впровадження віртуальних машин (VM) дозволяє ефективніше розподіляти обчислювальні завдання та зменшити витрати на обладнання.
- Віртуалізація безпеки (Security Virtualization). Можна використовувати віртуальні файрволи, IDS/IPS системи та інші засоби безпеки для захисту мережі та даних. При цьому доступ до ресурсів мережі може бути керованим та моніторитися на рівні віртуальних мереж та сегментів.
- Віртуалізація моніторингу (Monitoring Virtualization). Включає в себе використання засобів моніторингу мережі, які дозволяють відстежувати стан та продуктивність віртуальних ресурсів. Автоматизована система моніторингу дозволяє вчасно виявляти проблеми та вживати заходи для їх вирішення.
- Віртуалізація керування (Management Virtualization). Використання централізованого програмного забезпечення для керування всією віртуальною LAN. При цьому забезпечтується можливість автоматизованого розгортання, конфігурації та моніторингу віртуальних ресурсів.

Рисунок 1.8 – Структура віртуалізаційної моделі оптимізації

Віртуалізація LAN дозволяє оптимізувати використання ресурсів, покращити безпеку, спростити керування та забезпечити високий рівень доступності. Ця модель може бути адаптована до конкретних вимог та обставин вашої організації.

Висновки за розділом 1

Існує безліч підходів до оптимізації локальних мереж, але не має одного єдиного методу. Мережевий адміністратор може використати як один підхід так і об’єднати декілька, в залежності від ситуації.

У даній кваліфікаційній роботі буде розглядатись об’єднання методів віртуалізації та методів машинного навчання, у зв’язку з наступними причинами. Віртуалізація мережі дозволить абстрагуватися від поняття топології мережі та при цьому облегшить роботу мережевого адміністратора, так як управління мережею буде відбуватись із одного місця. А методи машинного навчання дозволять прогнозувати можливі зміни в структурі

мережі для покращення її ефективності, що в свою чергу можна автоматизувати і перекласти керування мережею на штучний інтелект.

РОЗДІЛ 2
АНАЛІЗ ПІДХОДІВ ВІРТУАЛІЗАЦІЇ МЕРЕЖІ

Будь-яка комп’ютерна мережа складається з таких апаратних елементів, як комутатори, маршрутизатори та брандмауери. Організація, що має відділення у різних географічних регіонах, може використовувати декілька різних мережевих технологій, які разом утворюють корпоративну мережу. Віртуалізація мережі – це процес поєднання всіх цих мережевих ресурсів задля забезпечення централізованого виконання адміністративних завдань. Адміністратори можуть налаштовувати та контролювати ці елементи віртуально, не торкаючись фізичних компонентів, що дозволяє суттєво спростити керування мережею.

2.1 Віртуалізація функцій мережі NFV

Віртуалізація функцій мережі [2] (network function virtualization – NFV) є концепцією мережевої архітектури, в якій за допомогою віртуалізації замінюються функції мережевих пристроїв. Технологія віртуалізації мережевих функцій поєднує функції таких мережевих пристроїв, як брандмауери, балансувальники навантаження і аналізатори трафіку, що працюють разом для підвищення продуктивності мережі.

Архітектура NFV складається з трьох частин:

- Централізована інфраструктура віртуальної мережі (NFVI): інфраструктура NFV може базуватися або на платформі керування контейнером, або на гіпервізорі, який абстрагує сховище, обчислювальні, та мережеві ресурси.
- Програмні додатки або функції віртуалізованої мережі (VNF): програмне забезпечення замінює апаратні компоненти традиційної мережевої

архітектури для надання різних типів мережевих функцій (віртуалізованих мережевих функцій).

- Фреймворк (часто відомий як MANO – управління, автоматизація та мережева оркестровка) необхідний для керування інфраструктурою та надання мережевих функцій.

Переваги NFV:

- дозволяє гнучко та динамічно надавати нові послуги, скоротивши при цьому капітальні та операційні витрати;
- заміна обладнання на стандартизовані сервери та мережеві компоненти не прив’язує операторів до постачальників обладнання;
- знижує операційні витрати за рахунок спрощення моніторингу та адміністрування операцій (всі мережеві функції переносяться в єдину віртуалізовану інфраструктуру);
- швидке підключення нових користувачів до мережі;
- окупає інфраструктуру телекомунікаційних компаній.

Коли провайдер послуг створює з’єднання з новим місцем, є кілька пристроїв, які повинні бути обов’язково встановлені в мережі, це, в першу чергу, керований маршрутизатор і демаркаційний пристрій Carrier Ethernet, які мають вирішальне значення, оскільки розділяють мережу клієнта та мережу оператора. Крім цього, до них додається обладнання, яке має бути встановлене для контролю та управління з’єднаннями та трафіком. Крім набору стандартного обладнання, вже перерахованого раніше, провайдери послуг часто змушені встановлювати кілька нестандартних пристроїв, які необхідні для забезпечення бізнес-завдань клієнта. І таким чином, для сервіс провайдера закупівля та сервісна підтримка такого обладнання стають дуже трудомістким, тим більше, що заздалегідь планувати рівень завантаження цього обладнання навряд чи можливо. NFV допомагає вирішити це завдання за рахунок віртуалізації мережевих функцій у програмних додатках, які можна запустити

як на звичайних серверах, так і на віртуальних машинах, що працюють на цих серверах.

На рисунку 2.1 зображено порівняння традиційної мережі із NFV мережею. Основна відмінність в мережі NFV полягає в тому що на стороні сервіс провайдеру, знаходиться не один маршрутизатор, а віртуальні та фізичні пристрої, які можуть динамічно визначати шлях призначення пакету, тому на стороні користувача відсутні маршрутизатори, адже пакети надходять напряму до кінцевих пристроїв.

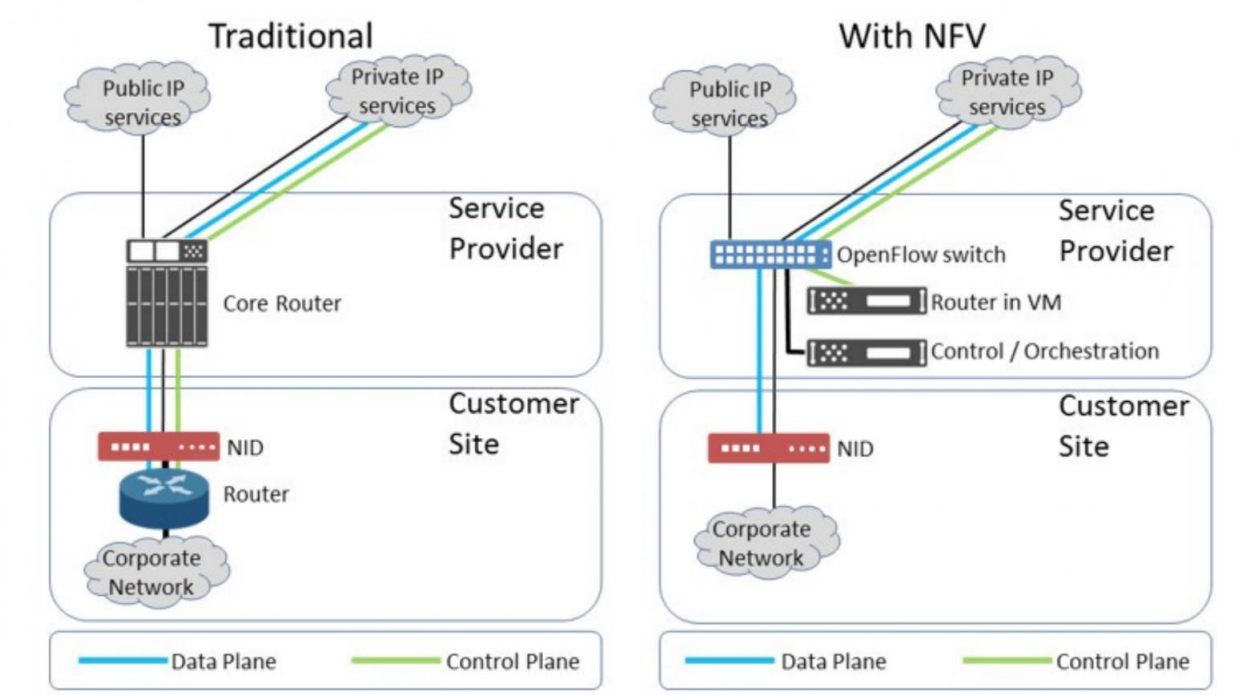

Рисунок 2.1 – Порівняння традиційної мережі із NFV мережею

2.2 Програмно-конфігурована мережа SDN

Програмно-конфігурована мережа (Software-defined Networking – SDN) – це підхід до мереж, який використовує програмні контролери, які можуть керуватися інтерфейсами прикладного програмування (API) для зв’язку з апаратною інфраструктурою для спрямування мережевого трафіку. Використовуючи програмне забезпечення, можна створити та керувати

низкою віртуальних накладених мереж, які працюють у поєднанні з фізичною базовою мережею. Мережі SDN пропонують потенціал для доставки середовищ програмних застосунків у вигляді коду та мінімізації практичного часу, необхідного для керування мережею [3].

2.2.1 Будова SDN

У SDN програмне забезпечення відокремлено від апаратного забезпечення. Мережа SDN складається із двох площин: площини керування та площини даних. Площина керування визначає куди надсилати трафік програмному забезпеченню, в той час коли площина даних фактично пересилає трафік в апаратне забезпечення. Це дозволяє мережевим адміністраторам програмувати та керувати всією мережею з однієї точки, а не від пристрою до пристрою.

Типова архітектура SDN (рис. 2.2) складається із трьох частин, які можуть бути розташовані в різних фізичних місцях, та двох інтерфейсів:

- Програми SDN – це програми, які явно, прямо та програмно передають свої вимоги до мережі та бажану мережеву поведінку контролеру SDN через північний інтерфейс (NBI). Крім того, вони можуть використовувати абстрактне уявлення про мережу для своїх внутрішніх цілей прийняття рішень. Програма SDN складається з однієї логіки програми SDN і одного або кількох драйверів NBI. Програми SDN можуть самі надавати інший рівень абстрактного мережевого контролю, таким чином пропонуючи один або більше NBI вищого рівня через відповідних агентів NBI.
- Контролер SDN – це логічно-централізований об’єкт, який відповідає за переклад вимог із прикладного рівня SDN до шляхів даних SDN і надання додаткам SDN абстрактного представлення мережі (яке може включати статистику та події) . Контролер SDN складається з одного або кількох агентів NBI, логіки керування SDN і драйвера інтерфейсу керування до площини даних (CDPI). Хоча визначення, як логічно-централізованого

об’єкта не передбачає і не виключає таких деталей реалізації як: об’єднання кількох контролерів; ієрархічне з’єднання контролерів; інтерфейси зв’язку між контролерами, а також віртуалізація чи нарізка мережевих ресурсів.

- Мережеві пристрої, які отримують інформацію від контролерів про те, куди перемістити дані.
- Інтерфейс Control to Data-Plane Interface (CDPI) – це інтерфейс, визначений між контролером і мережевими пристроями, який забезпечує: програмне керування всіма операціями пересилання, оголошення про можливості, статистичні звіти та повідомлення про події.
- Північний інтерфейс (NBI) – це інтерфейс, визначений між додатками і контролерами, які зазвичай надають абстрактні мережеві представлення та забезпечують пряме вираження мережевої поведінки та вимог.

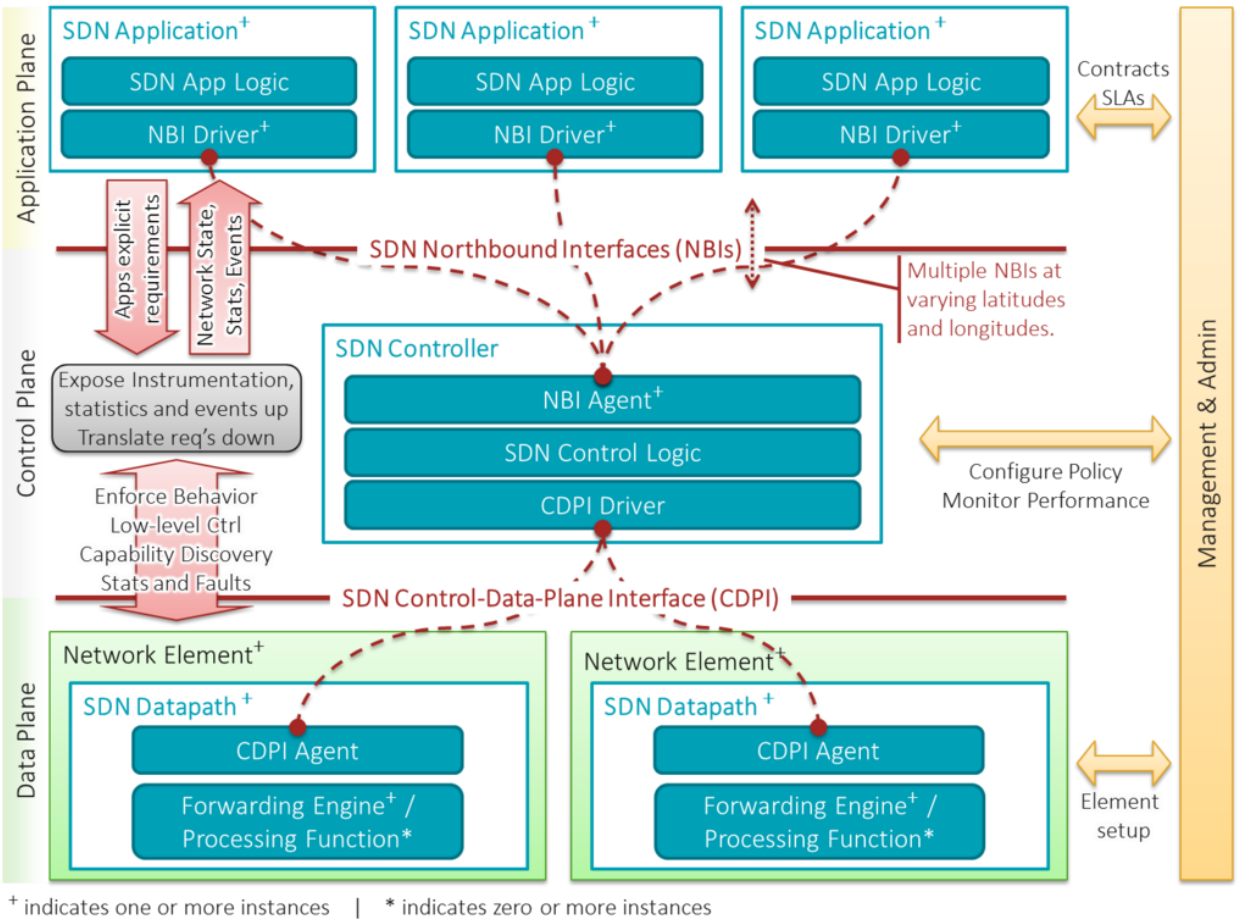

Рисунок 2.2 – Загальний огляд архітектури SDN

Фізичні або віртуальні мережеві пристрої фактично переміщують дані через мережу. У деяких випадках віртуальні комутатори, які можуть бути вбудовані в програмне або апаратне забезпечення, беруть на себе обов'язки фізичних комутаторів і об'єднують їхні функції в єдиний інтелектуальний комутатор. Комутатор перевіряє цілісність як пакетів даних, так і місць призначення віртуальної машини та переміщує пакети.

2.2.2 Переваги SDN

Архітектура SDN має багато переваг, здебільшого завдяки централізації контролю та управління мережею. Деякі з переваг включають:

- Простота керування мережею – відокремлення функцій пересилання пакетів від площини даних забезпечує пряме програмування та простіше керування мережею. Це може включати налаштування мережевих служб у режимі реального часу, таких як Ethernet або брандмауери, або швидкий розподіл ресурсів віртуальної мережі для зміни мережевої інфраструктури в одному централізованому місці.
- Рухливість – оскільки SDN забезпечує динамічне балансування навантаження для керування потоком трафіку відповідно до коливань потреби та використання, це зменшує затримку, підвищуючи ефективність мережі.
- Гнучкість – за допомогою рівня керування на основі програмного забезпечення мережеві оператори мають більше можливостей контролювати мережу, змінювати налаштування конфігурації, надавати ресурси та збільшувати пропускну здатність мережі.
- Покращений контроль над мережевою безпекою – SDN дозволяє мережевим адміністраторам встановлювати політики з одного центрального місця, щоб визначати контроль доступу та заходи безпеки в мережі за типом робочого навантаження або за сегментами мережі. Ви також можете використовувати мікросегментацію, щоб зменшити складність і встановити

узгодженість у будь-якій мережевій архітектурі – публічній хмарі , приватній хмарі , гібридній хмарі чи мультихмарі .

- Спрощена конструкція та робота мережі – адміністратори можуть використовувати єдиний протокол для зв’язку з широким спектром апаратних пристроїв через центральний контролер. Він також забезпечує більшу гнучкість у виборі мережевого обладнання, оскільки організації часто віддають перевагу використанню відкритих контролерів, а не пристроїв і протоколів, що відповідають постачальникам.
- Модернізація телекомунікацій – технологія SDN у поєднанні з віртуальними машинами та віртуалізацією мереж дозволяє постачальникам послуг забезпечувати чітке розділення мережі та контроль для клієнтів. Це допомагає постачальникам послуг покращити свою масштабованість і надавати пропускну здатність на вимогу клієнтам, яким потрібна більша гнучкість і використання змінної пропускної здатності.

2.2.3 Моделі SDN

Хоча передумова централізованого програмного забезпечення, що контролює потік даних у комутаторах і маршрутизаторах, застосовується до всіх програмно-визначених мереж, існують різні моделі SDN.

- Open SDN: мережеві адміністратори використовують такий протокол, як OpenFlow, щоб керувати поведінкою віртуальних і фізичних комутаторів на рівні даних.
- SDN через API: Замість використання відкритого протоколу інтерфейси прикладного програмування контролюють, як дані переміщуються через мережу на кожному пристрої.
- Модель SDN Overlay. Інший тип програмно визначеної мережі запускає віртуальну мережу поверх існуючої апаратної інфраструктури, створюючи динамічні тунелі до різних локальних і віддалених центрів обробки даних. Віртуальна мережа розподіляє пропускну здатність для різних

каналів і призначає пристрої для кожного каналу, залишаючи фізичну мережу недоторканою.

- Hybrid SDN: ця модель поєднує програмно визначену мережу з традиційними мережевими протоколами в одному середовищі для підтримки різних функцій у мережі. Стандартні мережеві протоколи продовжують спрямовувати частину трафіку, тоді як SDN бере на себе відповідальність за інший трафік, дозволяючи мережевим адміністраторам поетапно вводити SDN у застаріле середовище.

2.2.4 Відмінність SDN від традиційної мережі

Ключова відмінність між SDN і традиційною мережею полягає в інфраструктурі: SDN базується на програмному забезпеченні, тоді як традиційна мережа базується на апаратному забезпеченні. Оскільки площина керування базується на програмному забезпеченні, SDN є набагато гнучкішим, ніж традиційна мережа. Це дозволяє адміністраторам контролювати мережу, змінювати параметри конфігурації, надавати ресурси та збільшувати пропускну здатність мережі – і все це через централізований інтерфейс користувача без додавання додаткового обладнання.

Існують також відмінності в безпеці між SDN і традиційними мережами. Завдяки більшій видимості та можливості визначати безпечні шляхи, SDN пропонує кращу безпеку багатьма способами. Однак, оскільки програмно визначені мережі використовують централізований контролер, безпека контролера має вирішальне значення для підтримки безпечної мережі, і це єдиний вузол що представляє потенційну вразливість SDN.

2.3 Мережа, визначена знаннями

Мережа, визначена знаннями (Knowledge-Defined Networking – KDN) [4] – це розширена версія SDN, яка робить крок вперед, відокремлюючи площину

керування від логіки керування та вводячи нову площину, яка називається площиною знань, відокремленою від логіки керування для генерування знань на основі даних, зібраних із мережі.

Мережа, визначена знаннями (KDN) – це концепція використання інформації для генерування знань за допомогою моделей машинного навчання або моделей на основі правил, і відповідно до цих знань приймаються мережеві рішення

2.3.1 Архітектура мережі KDN

KDN складається із п’яти основних площин. Блок схема високого рівня зображена на рис. 2.3.

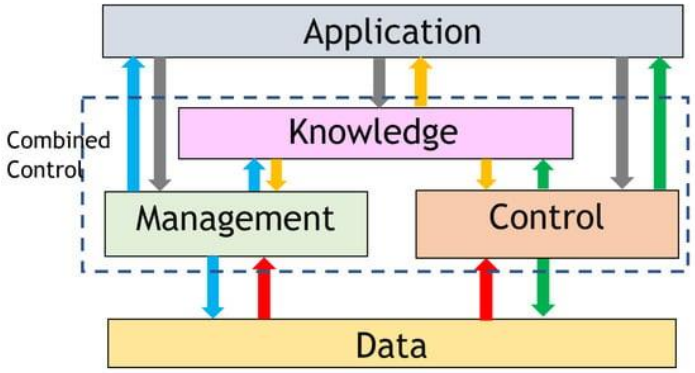

Рисунок 2.3 – Блок-схема високого рівня архітектури KDN

- Площина знань складається з трьох підрівнів:
  - площина генерації знань – генерує знання з використанням даних/інформації за допомогою методів на основі евристичної моделі або методів машинного навчання;
  - площина композиції знань – компонує згенеровані знання та універсальні знання, за допомогою редактора онтології, для створення складених знань, які можна використовувати для створення правил шляхом узгодження з намірами користувача;

- площина розподілу та управління знаннями – зберігає знання та правила з використанням бази знань.

- Площина управління працює паралельно з контролером KDN і відповідає за: збір процесів і даних/інформації з мережевих пристроїв, моніторинг стану мережевого пристрою та налаштування мережевого пристрою.

- Площина даних складається з пристроїв пересилання, які можуть зберігати, пересилати або обробляти дані відповідно до правил потоку, надісланих площиною керування. У KDN площина даних потрібна для надсилання даних, запитуваних площинами управління та контролю.

- Площина контролю складається з одного або декількох контролерів SDN на основі архітектури та відповідає за надсилання правил потоку, правил контролю доступу, правил пріоритезації трафіку на основі QoS тощо до площини даних.

- Площина додатків – ця площина забезпечує платформу для мережевих додатків для передачі вимог базовій мережевій інфраструктурі. Це також дозволяє мережевим адміністраторам централізовано визначати мережеві політики, специфічні для додатків, і визначати політики конфігурації мережі, які більш узгоджені з бізнес-потребами та цілями високого рівня, де логіка програми відокремлена від апаратного забезпечення.

2.3.2 Порівняння KDN з SDN та традиційними мережами

Традиційна мережа є найстарішим підходом до роботи в мережі та передбачає ручне налаштування та керування пристроями. Цей метод мережевих зв’язків був поширеним із самого початку створення мереж і досі переважає в сучасних мережах зв’язку.

SDN — це новітній підхід, який відокремлює площину керування від площини даних і забезпечує більшу гнучкість у проектуванні мережі. Площина керування переміщується в централізоване розташування, а

мережеві адміністратори використовують програмне забезпечення для керування мережею. Ця парадигма дозволяє мережевим адміністраторам легше та швидше керувати мережами завдяки підвищеній гнучкості та можливості програмування.

KDN створює мережу, що самонавчається, самооптимізується та самовідновлюється, інтегруючи технології AI та ML. Аналіз даних — це інструмент, який використовується системами KDN для автоматичного покращення продуктивності мережі, адаптації до мінливих мережевих обставин, а також виявлення та вирішення потенційних мережевих проблем до того, як вони стануть серйознішими.

Висновки за розділом 2

Розглянувши підходи віртуалізації мереж можна зробити висновок, що NFV, SDN та KDN є важливими інноваціями, які значно підвищують ефективність мережевих інфраструктур порівняно з традиційними методами.

Вони мають перевагу над традиційними методами в гнучкості, масштабованості, в швидкості запровадженні змін, в економії ресурсів, легке керування та можливість автоматизації.

Отже, віртуалізація мережі ефективніша за традиційні мережі, що є ключовим чинником у подоланні викликів, пов’язаних із зростанням обсягів даних та змінами в бізнес-вимогах.

РОЗДІЛ 3
МОДЕЛЬ ОПТИМІЗАЦІЇ НА ОСНОВІ НЕЙРОНОЇ МЕРЕЖІ

На рис. 3.1 показана архітектура прецеденту, яка виконує оптимізацію мережі в контексті парадигми мережі, визначеною знаннями (KDN – knowledge-Defined Networking). У цьому випадку припускається що площина керування отримує своєчасні оновлення стану мережі (наприклад, матриця трафіку, вимірювання затримки).

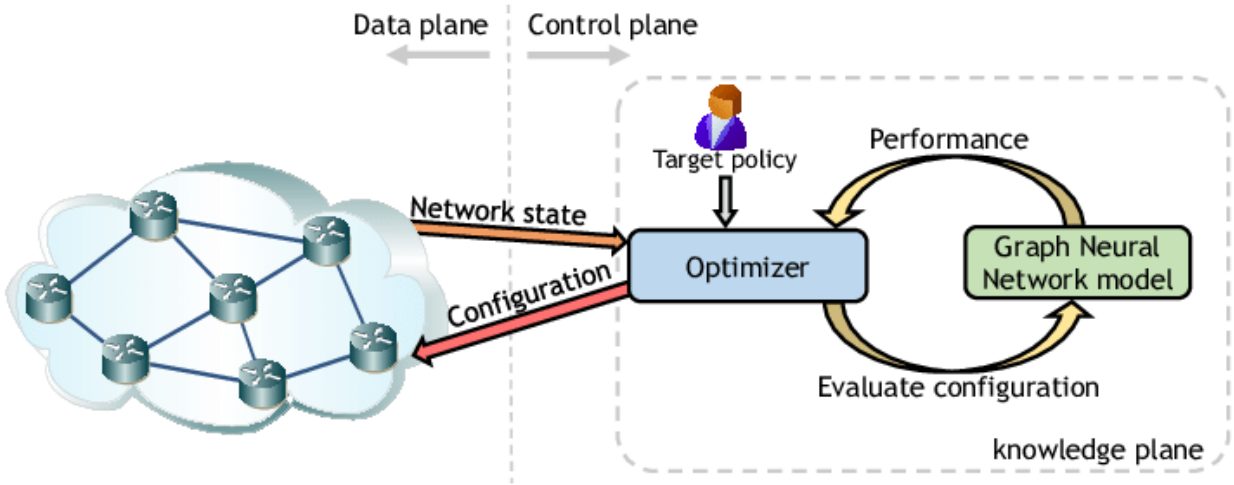

Рисунок 3.1 – Архітектура оптимізації мережі

Цього можна досягти за допомогою «звичайних» методів вимірювання на основі SDN (наприклад, OpenFlow [5], OpenSketch [6]) або більш нових пропозицій телеметрії, таких як INT для P4 [7] або iOAM [8]. Так само в площині знань є оптимізатор, поведінка якого визначається заданою цільовою політикою. Ця політика, відповідно до мереж на основі намірів, може бути визначена декларативною мовою, такою як NEMO [9], і, нарешті, перекладена на (багатоцільову) проблему оптимізації мережі. На цьому етапі точна мережева модель може відігравати вирішальну роль у процесі оптимізації, використовуючи її для запуску алгоритмів, які ітеративно досліджують

продуктивність потенційних рішень, щоб знайти оптимальну конфігурацію. Навмисно залишають за межами цієї архітектури етап навчання. Щоб досягти успіху в таких сценаріях, як запропонований вище, мережева модель повинна відповідати двом основним вимогам забезпечувати точні результати та мати низьку обчислювальну вартість, щоб дозволити оптимізаторам мережі працювати в коротких проміжках часу. Крім того, оптимізаторам важливо мати достатню гнучкість для моделювання сценаріїв «що-якщо», що включають різні схеми маршрутизації, зміни в топології та варіації в матриці трафіку. З цією метою можна покластись на здатність моделей Graph Neural Network (GNN) ефективно працювати та узагальнювати середовища, представлені у вигляді графів.

Було обрано модель на основі GNN, RouteNet [10], побудована на основі нейронних мереж передачі повідомлень, яка використовується в галузі хімії [11], здатна поширювати будь-яку схему маршрутизації по топології мережі та абстрагувати значущу інформацію про поточний стан мережі. На рис. 3.2 показано схематичне зображення моделі.

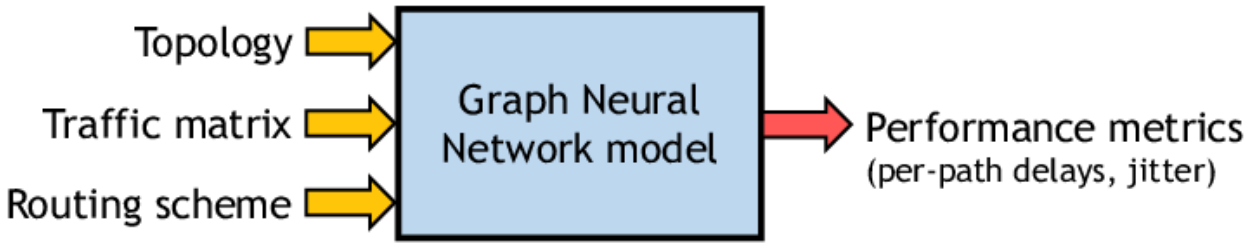

Рисунок 3.2 – Схема RouteNet

Більш докладно, RouteNet приймає як вхідні дані задану топологію, схему маршрутизації джерело-призначення (тобто зв’язки між кінцевими точками, шляхами та посиланнями) і матрицю трафіку (визначену як пропускна здатність між кожною парою вузлів у мережі) і, нарешті, створює показники продуктивності відповідно до поточного стану мережі (наприклад, затримки на шлях або тремтіння сигналу). Для цього RouteNet використовує

вектори з фіксованою розмірністю, які кодують стани шляхів і посилань і передають інформацію між ними відповідно до схеми маршрутизації.

3.1 Моделювання мережі за допомогою графових нейронних мереж

Комп'ютерна мережа може бути представлена як набір з'єднань $N = \{l_i\}, i \in (0, 1, \dots, n_l)$, а схема маршрутизації в мережі – як набір шляхів $\mathcal{R} = \{p_k\}$, $k \in (0, 1, \dots, n_p)$. Кожний шлях визначений як послідовність з'єднань $p_k = (l_{k(0)}, \dots, l_{k(|p_k|)})$, де $k(i)$ – це індекс і-ого з'єднання на шляху k. Де властивості (особливості) кожного шляху і з'єднання позначені як x_{l_i} та x_{p_i}. Вимірювані ключові показники ефективності моделюються як випадкові змінні W_i і L_i, де перша є наскрізною затримкою, а друга — загальна кількість відкинутих пакетів за період часу для кожної пари джерело-призначення в мережі.

Для створення RouteNet було використано архітектуру нейронних мереж передачі повідомлень (надалі MPNN – Message Passing Neural Network). Основне припущення MPNN полягає в тому, що інформація, що відноситься до вузлів, ребер або всьому графу, може бути закодована у векторах фіксованої розмірності, також званих вкладеннями. Прямий прохід в MPNN є комбінацією трьох простих функцій:

1) Message (Повідомлення);
2) Update (Оновлення);
3) Readout (Зчитування).

Функція Message приймає як вхідні вкладення вузла/ребра і виводить інформаційний вектор (повідомлення), який буде надіслано всім сусідам у графі. Функція Update збирає (підсумовує) повідомлення від усіх сусідів графа та оновлює вкладення вузлів/ребер. Цей обмін повідомленнями повторюється T разів, і, нарешті, функція Readout приймає отримані вкладення вузлів/ребер для отримання вихідних даних GNN.

RouteNet обробляє вхідні мережеві топології змінного розміру та довільні схеми маршрутизації джерело-одержувач і виробляє наскрізні прогнози продуктивності на виході. Основне припущення, що лежить в основі RouteNet, полягає в тому, що інформація на рівні шляху (наприклад, наскрізні показники, такі як затримки або втрата пакетів) і на рівні зв'язку (наприклад, затримка з'єднання, швидкість втрат пакетів, використання з'єднання) може бути закодована в навчальні вектори дійсних чисел (вектори стану шляху та посилання відповідно). Зауважимо, що абстракція шляху може не обов'язково відповідати фізичному шляху. Це може бути загальний наскрізний потік трафіку. Наприклад, тунель MPLS. Виходячи з цього припущення, RouteNet побудовано на таких принципах:

1) стан шляху залежить від стану всіх ланок, які лежать на шляху;

2) стан посилання залежить від стану всіх шляхів, які проходять через посилання.

Нехай, стан з'єднання буде h_{li} – невідомий вектор. Таким же чином вектор шляху – h_{Pi} . Ці принципи можуть бути математично сформульовані наступними виразами:

$$h_{l_i} = f\left(h_{p_i}, \dots, h_{p_j}\right),\ l_i \in p_k, k = 1, \dots, j \quad (1)$$

$$h_{p_k} = g\left(h_{l_{k(0)}}, \dots, h_{l_{k(|p_k|)}}\right) \quad (2)$$

де f і g – деякі невідомі функції. Добре відомо, що нейронні мережі можуть працювати як універсальні апроксиматори функцій. Однак пряма апроксимація функцій f і g неможлива в цьому випадку, оскільки:

1) рівняння (1) і (2) визначають неявну функцію (нелінійну систему рівнянь зі станами, які є прихованими змінними);

2) ці функції залежать від схеми маршрутизації вхідних даних;

3) розмірність кожної функції дуже велика. Для цього знадобиться великий набір навчальних зразків.

RouteNet представляє архітектуру GNN, яка ефективно вивчає f і g. Він інваріантний до топології та схеми маршрутизації та робить можливим наближення нейронної функції.

Input: $\mathbf{x}_p$, $\mathbf{x}_l$,$\mathcal{R}$
Output: $\mathbf{h}_p^T$, $\mathbf{h}_l^T$, $\hat{\mathbf{y}}_p$
1 **foreach** $p \in \mathcal{R}$ **do** $\mathbf{h}_p^0 \leftarrow [\mathbf{x}_p, 0 \ldots, 0]$;
2 **foreach** $l \in \mathcal{N}$ **do** $\mathbf{h}_l^0 \leftarrow [\mathbf{x}_l, 0 \ldots, 0]$;
3 **for** $t = 0$ *to* $T - 1$ **do**
4 **foreach** $p \in \mathcal{R}$ **do**
5 **foreach** $l \in p$ **do**
6 $\mathbf{h}_p^t \leftarrow RNN_t(\mathbf{h}_p^t, \mathbf{h}_l^t)$
7 $\tilde{\mathbf{m}}_{p,l}^{t+1} \leftarrow \mathbf{h}_p^t$
8 **end**
9 $\mathbf{h}_p^{t+1} \leftarrow \mathbf{h}_p^t$
10 **end**
11 **foreach** $l \in \mathcal{N}$ **do**
12 $\mathbf{h}_l^{t+1} \leftarrow U_t\left(\mathbf{h}_l^t, \sum_{p:k \in p} \tilde{\mathbf{m}}_{p,k}^{t+1}\right)$
13 **end**
14 **end**
15 $\hat{y}_p \leftarrow F_p(\mathbf{h}_p)$

Рисунок 3.3 – Алгоритм прямого розповсюдження RouteNet

Алгоритм (рис. 3.3) описує пряме розповсюдження (і внутрішню архітектуру) RouteNet. У цьому процесі ця модель GNN отримує як вхідні дані початковий шлях і характеристики каналу x_p, x_l і опис маршруту $\mathscr{R}$, а також виводить метрику для кожного шляху ($\hat{y}_p$). Зауважте, що ми спростили позначення, вилучивши підіндекси шляхів і посилань.

Архітектура RouteNet дозволяє мати справу з циклічними залежностями, описаними в рівняннях (1) і (2), і підтримувати довільні схеми маршрутизації (які за своєю суттю представлені в архітектурі). Щоб усунути циклічні залежності, RouteNet повторює ті самі операції передачі повідомлень над векторами стану посилань і шляхів T разів (цикл з рядка 3). Ці кроки представляють процес збіжності до фіксованої точки функції з початкових станів h_p^0 і h_l^0.

Щодо питання інваріантності маршрутизації (більш загально відомої як інваріантність топології в контексті проблем, пов’язаних із графами), RouteNet вимагає використання структури, здатної представляти графи різних топологій змінного розміру. У розглядуваному випадку прагнемо представити різні схеми маршрутизації єдиним способом. Одне сучасне рішення цієї проблеми [12] пропонує використання нейронних архітектур передачі повідомлень, які поєднують обидва: подання топології у вигляді графа та вектори для кодування станів зв’язку. У цьому контексті RouteNet можна інтерпретувати як розширення нейронної мережі передачі повідомлень, яка спеціально підходить для представлення залежностей між посиланнями та шляхами, заданими схемою маршрутизації (рівняння (1) і (2)).

В алгоритмі цикл від рядка 3 до рядка 14 представляє операції передачі повідомлень, які взаємно обмінюються закодованою інформацією (прихованими станами) між посиланнями та шляхами. Подібним чином, рядки 9 і 12 є функціями оновлення, які кодують нову зібрану інформацію в приховані стани відповідно для шляхів і посилань. Оновлення станів шляхів (рядок 9) є простим присвоєнням значення, тоді як оновлення посилань (рядок 12) є навченою нейронною мережею. Загалом, оновлення шляху також може бути тренованою нейронною мережею.

Ця архітектура забезпечує гнучкість представлення будь-якої схеми маршрутизації джерело-одержувач. Це досягається прямим відображенням R (тобто набору наскрізних шляхів) на певні операції передачі повідомлень між об’єктами посилань і шляхів, які визначають архітектуру RouteNet. Таким чином, кожен шлях збирає повідомлення з усіх посилань, що входять до нього (цикл з рядка 5), і, аналогічно, кожне посилання отримує повідомлення з усіх шляхів, які його містять (рядок 12). Враховуючи, що порядок шляхів, що проходять через те саме посилання, не має значення, було використано просте підсумовування для агрегації повідомлень на рівні шляху. Однак у випадку з посиланнями наявність втрати пакетів може означати послідовну залежність у посиланнях, які утворюють кожен шлях. Тому для агрегування станів зв’язків

на шляхах було використано повторювану нейронну мережу (RNN). RNN добре підходять для фіксації залежності в послідовностях змінного розміру (наприклад, обробка тексту). Це дозволяє моделювати послідовну залежність посилань і поширювати цю інформацію по всіх шляхах.

Крім того, використання цих функцій агрегації повідомлень (RNN і сумування) дозволяє значно обмежити розмірність проблеми. Метою цих функцій є збір довільної кількості повідомлень, отриманих у кожній сутності (посилання або шлях), і стиснення цієї інформації в масиви фіксованих розмірів (тобто приховані стани). Розмір прихованих станів посилань і шляхів є настроюваними гіперпараметрами. Зрештою, усі приховані стани в RouteNet представляють явну функцію, що містить інформацію про стани посилання та шляху. Це дозволяє використовувати їх для визначення різних функцій одночасно. Маючи набір прихованих станів h_p^T і h_l^T, можна підключити нейронні мережі зчитування для оцінки деяких показників шляху та/або показники рівня зв'язку. Зазвичай цього можна досягти, використовуючи звичайні повнозв'язані нейронні мережі з деякими рівнями та відповідними функціями активації. В алгоритмі функція F_p (рядок 15) представляє функцію зчитування, яка передбачає деякі характеристики рівня шляху ($\hat{y}_p$), використовуючи як вхідні дані приховані стани шляху h_p. Подібним чином можна було б зробити висновок про деякі глобальні властивості та функції рівня зв'язку ($\hat{y}_p$), використовуючи також інформацію в прихованих станах посилання h_l.

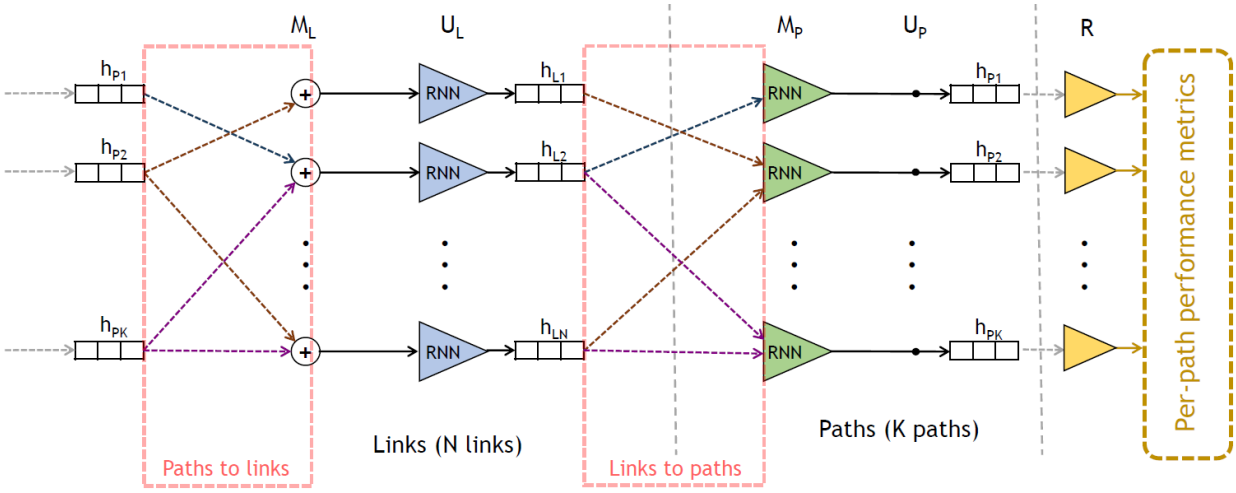

Рисунок 3.4 – Графічне зображення архітектури RouteNet

3.2 Моделі затримки, джиттера та падінь

Формально затримка та тремтіння i-го шляху визначаються як $\mathbb{E}W_i$ та $\mathbb{D}^2W_i$ відповідно. З моделювання отримуємо вибіркове середнє $\overline{w}_i$ та дисперсію $s^2(w_i)$ які є їхніми оцінками. Замість моделювання $\overline{w}_i$ та $s^2(w_i)$ незалежно, було апроксимовано весь розподіл W_i (граничний розподіл з урахуванням вхідних характеристик) розподілом ймовірностей, параметризованим виходом нейронної мережі RouteNet $\hat{y}_p$, який є двоелементним вектором, що представляє затримку та тремтіння. Прямим узагальненням моделі є:

$$W_i \sim Norm(\mu_i, \sigma_i),\ \mu_i = \hat{y}_{i0},\ \sigma_i = softplus(\hat{y}_{i1}). \quad (3)$$

Таку модель можна навчити шляхом максимізації логарифмічної функції правдоподібності нормального розподілу (функція втрат є негативною):

$$\boldsymbol{\ell}(\mu_i, \sigma_i) = -n_i \left(\frac{s^2(w_i)}{2\sigma_i^2} + \frac{(\overline{w}_i - \mu_i)^2}{2\sigma_i^2} + \log(\sigma_i) \right). \quad (4)$$

де n_i – загальна кількість прийнятих пакетів. Зауважимо, що ця функція втрат є просто масштабованою квадратичною помилкою затримки плюс додаткові члени, що представляють помилку джиттера, і вона використовується в гетероскедастичній регресії. Така проста форма функції втрат (негативний логарифм правдоподібності) можлива, оскільки вибіркове середнє значення та дисперсія є достатньою статистикою нормального розподілу та використовується апроксимація середнього поля. Припускається, що затримки на пакет і на шлях є незалежними й однаково розподіленими випадковими змінними, а залежність між шляхами виходить лише з очікуваних значень. Термін середнє поле використовується через подібність такої моделі до апостеріорного середнього поля у варіаційному висновку.

У випадку гамма-розподілу функція втрат буде задана як:

$$\ell(\alpha_i, \beta_i) = n_i \log \Gamma(\alpha_i) + n_i \beta_i \overline{w_i} + n_i (1 - \alpha_i) \overline{\log w_i} - n_i \alpha_i \log \beta_i,$$

де α і β будуть виходами RouteNet.

Цей підхід не обмежується моделюванням лише затримки. Модель можна налаштувати на різні характеристики продуктивності, змінюючи розподіл. Зокрема, вибір дискретного розподілу, як-от Біноміальний (пуассонівський — ще один варіант), дозволяє моделювати втрату пакетів на кожному шляху:

$$L_i \sim Binomial(p_i, n_i + l_i),\ p_i = sigmoid(\hat{y}_i),$$

де p_i — коефіцієнт втрат пакетів на шляху i (тобто $l_i/(n_i + l_i)$).

Функція логарифму правдоподібності в цьому випадку визначається як:

$$\ell(p_i) = \ell_i \log(p_i) + n_i \log(1 - p_i)\ ,$$

де ℓ_i — кількість спостережуваних втрат, що є достатньою статистикою для біноміального розподілу. Така функція втрат також поширена в задачах двійкової класифікації.

3.3 Фреймворки машинного навчання

Створення штучних нейронних мереж (ШНМ) вручну, може бути досить складним завданням, враховуючи можливу складність структури ШНМ. У результаті були створені такі фреймворки, як TensorFlow[13] і PyTorch[14], щоб спростити створення та використання ШНМ. Ці структури забезпечують блоки нейронної мережі, функції вартості та оптимізатори для складання та навчання моделей нейронної мережі.

3.3.1 Огляд TensorFlow

TensorFlow — це популярна платформа з відкритим кодом для машинного навчання. Перш ніж стати фреймворком із відкритим кодом, її розробляли дослідники і інженери із команди GoogleBrain.

Програмна бібліотека TensorFlow замінила фреймворк Google DistBelief і працює майже на всіх доступних платформах виконання (CPU, GPU, TPU, Mobile тощо). Фреймворк надає математичну бібліотеку, яка включає основні арифметичні оператори та тригонометричні функції.

Зараз TensorFlow використовується різними міжнародними компаніями, такими як Google, Uber, Microsoft, а також багато університетів використовують TensorFlow для проведення своїх досліджень в галузях машинного та глубокого навчання.

TensorFlow Lite – полегшена версія основного фреймворку створена для периферійного машинного навчання. Він був оптимізований для запуску полегшених алгоритмів на різних периферійних пристроях, які мають обмеження по обчислюваних ресурсах, такі як смартфони, мікроконтролери, тощо.

Переваги TensorFlow:

– Підтримка та керування бібліотекою: TensorFlow підтримується компанією Google і часто випускає нові версії із новими особливостями.

– Відкритий код: TensorFlow — це платформа з відкритим кодом, доступна широкому колу користувачів і дуже популярна.

– Візуалізація даних: TensorFlow надає інструмент під назвою TensorBoard для графічної візуалізації даних. Це також дозволяє легко налагоджувати вузли, зменшує зусилля, пов’язані з переглядом усього коду, і ефективно вирішує нейронну мережу.

– Сумісність з Keras: TensorFlow сумісний із Keras, що дозволяє користувачам кодувати деякі високорівневі функціональні розділи та надає системні функції для TensorFlow (конвеєрне формування, оцінювачі тощо).

– Велика масштабованість: властивість TensorFlow розгортатися на кожній машині дозволяє користувачам розробляти будь-які системи.

– Сумісність: TensorFlow сумісний з багатьма мовами, такими як C++, JavaScript, Python, C#, Ruby та Swift. Це дозволяє користувачам працювати із звичною для них мовою програмування.

– Архітектурна підтримка: TensorFlow знаходить своє використання як бібліотека апаратного прискорення завдяки паралелізму робочих моделей. Він використовує різні стратегії розподілу в системах GPU і CPU. TensorFlow також має свою архітектуру TPU, яка виконує обчислення швидше, ніж GPU та CPU. Таким чином, моделі, створені з використанням TPU, можна легко розгортати в хмарі за нижчою ціною та виконувати швидше. Однак архітектура TensorFlow TPU дозволяє лише виконувати модель, а не навчати її.

Недоліки TensorFlow:

– Швидкість: TensorFlow відстає від своїх конкурентів у швидкості обчислень.

– Залежності: Хоча TensorFlow зменшує довжину коду та полегшує доступ до нього для користувача, він додає рівень складності його використанню. Код який можна виконувати з використанням будь-якої платформи для його підтримки, що збільшує залежність для виконання.

– Символічні цикли: TensorFlow затримується у забезпеченні символічних циклів для невизначених послідовностей. Вони використовуються для певних послідовностей, що робить його зручною системою. Тому його називають низькорівневим API.

– Підтримка графічного процесора: спочатку TensorFlow мав лише підтримку NVIDIA для графічного процесора та підтримку Python для програмування на графічному процесорі, що є недоліком, оскільки існує різке збільшення кількості інших мов у глибокому навчанні. TensorFlow Distribution Strategies — це API TensorFlow для розподілу навчання між кількома GPU, кількома машинами або TPU. Використання цього API, дає змогу поширювати існуючі моделі та навчальний код із мінімальними змінами коду.

3.3.2 Огляд PyTorch

PyTorch вперше був представлений у 2016 році. До PyTorch фреймворки глибокого навчання часто зосереджувалися на швидкості чи зручності використання, але не на обох. PyTorch став популярним інструментом у дослідницькій спільноті глибокого навчання завдяки поєднанню уваги на зручності використання з ретельним розглядом продуктивності. Він забезпечує імперативний стиль програмування Pythonic, який підтримує код як модель, полегшує налагодження та сумісний з іншими популярними науково-обчислювальними бібліотеками, залишаючись при цьому ефективним і підтримуючи апаратні прискорювачі, такі як графічні процесори.

PyTorch — це бібліотека Python, яка виконує миттєве виконання динамічних тензорних обчислень із автоматичним диференціюванням і прискоренням GPU, зберігаючи при цьому продуктивність, порівнянну з найшвидшими поточними бібліотеками для глибокого навчання. Сьогодні більша частина його ядра написана на C++, що є однією з основних причин, чому PyTorch може значно зменшити накладні витрати порівняно з іншими фреймворками.

Кілька популярних програм глибокого навчання створено на основі PyTorch, зокрема Tesla Autopilot або Uber's Pyro.

Переваги PyTorch:

– PyTorch заснований на Python: PyTorch орієнтований на Python або «pythonic», розроблений для глибокої інтеграції в код Python замість того, щоб бути інтерфейсом до бібліотеки, написаної іншою мовою. Python є однією з найпопулярніших мов, що використовуються науковцями з обробки даних, а також є однією з найпопулярніших мов, які використовуються для створення моделей машинного навчання та досліджень ML.

– Простий для вивчення: оскільки його синтаксис подібний до звичайних мов програмування, таких як Python, PyTorch порівняно легший у вивченні, ніж інші фреймворки глибокого навчання.

– Відлагодження: PyTorch можна відлагодити за допомогою одного з багатьох широко доступних інструментів відлагодження Python (наприклад, інструменти Python pdb та ipdb).

– Динамічні обчислювальні графи: PyTorch підтримує динамічні обчислювальні графи, що означає, що поведінку мережі можна змінювати програмно під час виконання.

– Паралелізм даних: функція паралелізму даних дозволяє PyTorch розподіляти обчислювальну роботу між кількома ядрами CPU або GPU.

Хоча інші фреймворки також надають такі можливості, однак в PyTorch це простіше реалізовано.

Недоліки PyTorch:

– Відсутність обслуговування моделей на виробництві: хоча це зміниться в майбутньому, інші фреймворки ширше використовуються для реальної виробничої роботи.

– Обмежені інтерфейси моніторингу та візуалізації: хоча TensorFlow також постачається з високопродуктивним інструментом візуалізації для побудови графа моделі (TensorBoard), PyTorch поки що не має нічого подібного. Таким чином, розробники можуть використовувати один із багатьох існуючих інструментів візуалізації даних Python або зовні підключатися до TensorBoard.

– Не такий обширний: PyTorch не є наскрізним інструментом розробки машинного навчання; розробка фактичних додатків вимагає перетворення коду PyTorch в інший фреймворк, такий як Caffe2, для розгортання додатків на серверах, робочих станціях і мобільних пристроях.

3.3.3 Порівняння PyTorch і TensorFlow

Основними характеристиками даних фреймворків це точність, швидкість та використання пам’яті. Порівнюючи рисунки 3.5 та 3.6, можна зробити висновок що обидві моделі мають приблизно однакову точність. Для обох моделей точність перевірки моделей в обох системах становила в середньому близько 78% після 20 епох. Таким чином, обидва фреймворки здатні точно реалізувати нейронну мережу та здатні давати однакові результати за однакової моделі та набору даних для навчання.

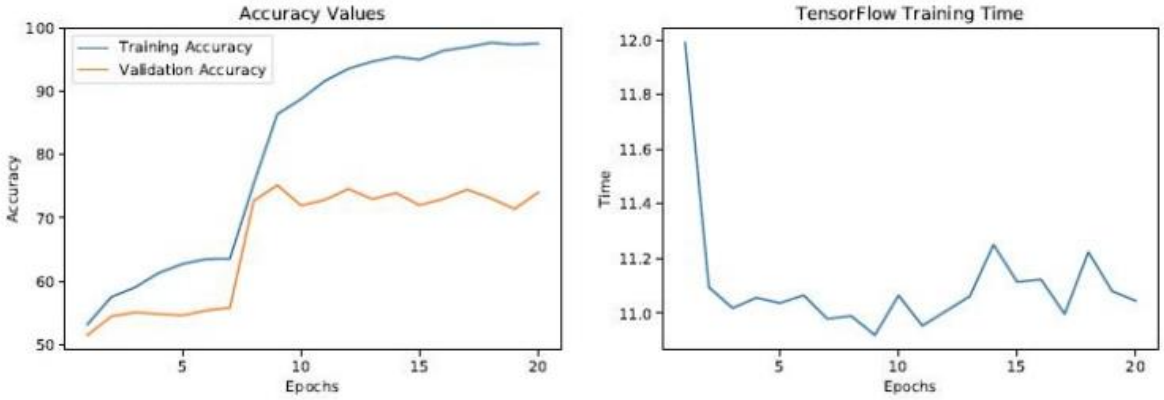

Рисунок 3.5 – Час навчання та точність Tensorflow[15]

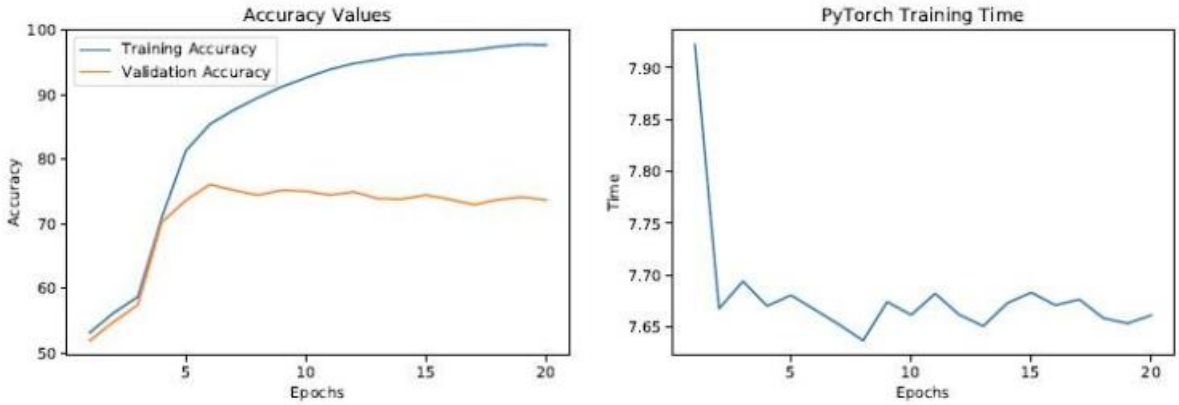

Рисунок 3.6 – Час навчання та точність PyTorch[15]

З рисунків 3.5 та 3.6 можна побачити, що PyTorch швидше навчається із середнім значенням в 7.67 секунд на епоху, в той час як у TensorFlow – 11.19 секунд на епоху.

При відносно однаковій точності і незначному програшу в швидкості навчання TensorFlow в даному експерименті[15] використовував 1.7 Гб оперативної пам’яті, в той час як PyTorch – 3.5 Гб оперативної пам’яті. Хоча при початковій загрузці даних, споживання оперативної пам’яті було приблизно однаковим: 4.8 ГБ в TensorFlow та 5 Гб в PyTorch.

В загальному реалізації Tensorflow і PyTorch демонструють однакову точність. Однак час навчання TensorFlow значно вищий, але використання пам’яті менше.

3.4 Опис моделі нейронної мережі для оптимізації

Для реалізації моделі нейронної мережі було обрано мову програмування Python і фреймворк iGNNition [16], побудований на основі TensorFlow, який дозволяє описати модель роботи в графовій нейронній мережі, а також описати сутності графа.

Щоб визначити архітектуру GNN, потрібно визначити файл model_description.yaml[17]. Цей файл міститиме кілька розділів, які визначатимуть різні аспекти розглядуваної GNN. Точніше, розділи, які повинні бути заповнені:

- Крок 1. Визначення сутності;
- Крок 2. Визначення передачі повідомлення;
- Крок 3. Визначення функції зчитування;
- Крок 4. Визначення внутрішніх нейронних мереж.

3.4.1 Визначення сутності

Під час проектування GNN ми можемо виявити ситуації, в яких не всі вузли в графі поводяться/представляють той самий об’єкт. Для цього нам може знадобитися розглянути різну поведінку залежно від типу вузла, про який йдеться. В розглядуваному випадку це дві сутності «шлях» та «з’єднання»

Для того щоб визначити сутність потрібно вказати ім’я сутності, її розмірність та вхідні дані які вона приймає (рис. 3.6).

```
entities:
- name: link
  state_dimension: 32
  initial_state:
    - type: build_state
      input: [$capacity]

- name: path
  state_dimension: 32
  initial_state:
    - type: build_state
      input: [$traffic]
```

Рисунок 3.6 – Опис сутностей графу

У ключовому полі entities, що зображено на рис. 3.5, вказується список всіх сутностей, де кожна з них описується. Потім для кожної з сутностей ми спочатку вказуємо її назву (name), яку ми будемо використовувати в решті визначення GNN для посилання на цей тип вузлів. Крім того, ми надаємо розмірність станів (state_dimension), які матиме кожен із цих вузлів. Нарешті, ми повинні вказати, як обчислюється початковий стан. Для цього визначення ми повинні надати список «операцій», які все більше визначають кінцевий початковий стан. Так для сутності «з'єднання» вхідними даними є ємність з'єднання, а для «шляху» відповідно трафік мережі на даному шляху.

3.4.2 Визначення передачі повідомлень

На цьому етапі ми повинні визначити основну частину алгоритму GNN, яка є фазою передачі нейронних повідомлень. На цьому етапі ми визначаємо, як різні вузли в графі обмінюються повідомленнями один з одним, щоб створювати вбудовування вузлів, які належним чином враховують структурну інформацію графа.

Передача повідомлення відбувається за допомогою трьох функції:

- функція повідомлення визначає як вихідні вузли будуть формувати свої повідомлення, які вони будуть надсилати відповідним вузлам призначення;
- функція агрегації визначає як кожен із вузлів призначення прийматиме всі повідомлення, і створює єдиний вхід. Наприклад можна підсумувати всі повідомлення в єдиний остаточний вхід;
- функція оновлення описує як сутність використовуватиме агреговану вхідну інформацію для оновлення свого поточного прихованого стану.

```
message_passing:
  num_iterations: 8
  stages:
    # STAGE 1:
    - stage_message_passings:
      # links to paths
      - destination_entity: path
        source_entities:
          - name: link
            message:
              - type: direct_assignment
        aggregation:
          - type: ordered
        update:
          type: neural_network
          nn_name: recurrent1
    # STAGE 2:
    - stage_message_passings:
      # paths to links
      - destination_entity: link
        source_entities:
        - name: path
          message:
          - type: direct_assignment
        aggregation:
          - type: sum
        update:
          type: neural_network
          nn_name: recurrent1
```

Рисунок 3.7 – Опис фази передачі повідомлення

Як можна побачити з рис. 3.7 перш за все потрібно визначити кількість ітерацій передачі повідомлення, а далі вже самі етапи передачі. Першим етапом є повідомлення від «з'єднаннь до шляхів» в цій фазі визначається сутність одержувача, сутність джерела з якого буде надсилатись саме повідомлення, та тип цього повідомлення. Потім функція агрегації, і оновлення. Аналогічно для другої фази повідомлення. На рис. 3.8 показано за якою схемою проходить обмін повідомленнями в мережі RouteNet.

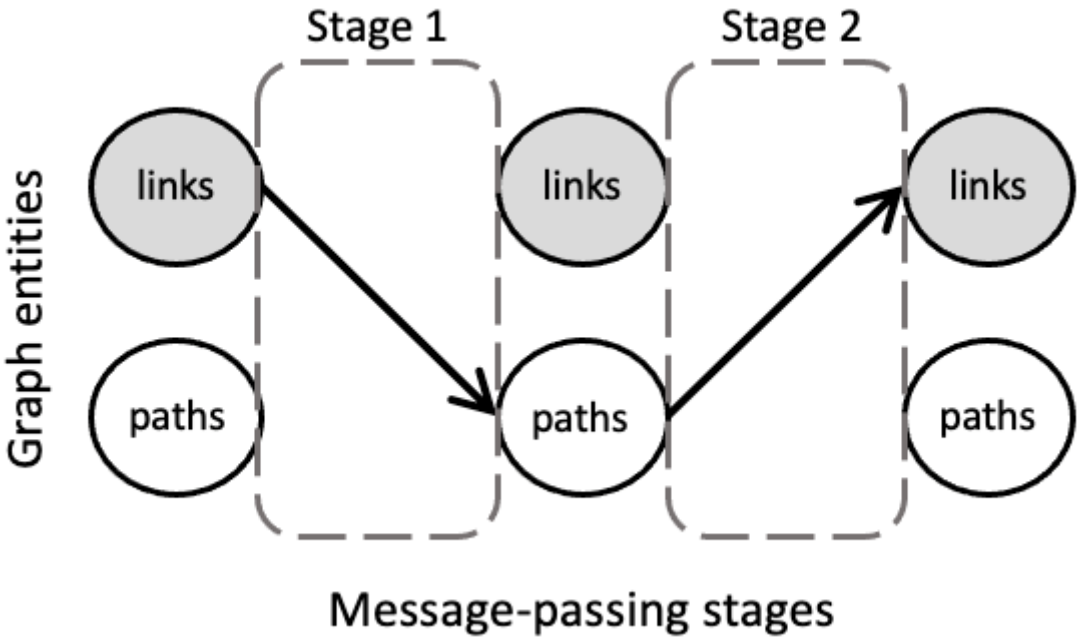

Рисунок 3.8 – Схема передачі повідомлень в RouteNet

3.4.3 Визначення зчитування

Після того як ми визначили передачу повідомлення, залишається визначити зчитування. Функція зчитування (рис. 3.9) відповідає за отримання деяких/усіх кінцевих станів, обчислених під час передачі повідомлення, та використання їх належним чином для прогнозування кінцевої мітки.

```
readout:
- type: neural_network
  input: [path]
  nn_name: readout_model
  output_label: [$delay]
```

Рисунок 3.9 – Функція зчитування

Як можемо побачити, що в даній функції в якості вхідних даних передаються шляхи які надалі входять до нейронної мережі. На основі роботи нейронної мережі буде сформована затримка.

3.4.4 Визначення внутрішніх нейронних мереж

Нарешті, залишається тільки визначити нейронні мережі. Зауважте, що в усіх попередніх розділах ми явно не визначали фактичну архітектуру нейронної мережі, а лише посилалися на неї за назвою. У цьому розділі ми повинні вказати фактичну архітектуру кожного з них.

З попереднього розділу ми в функції зчитування посилались на нейронну мережу з назвою readout_model, опис якої зображений на рис. 3.10.

```
neural_networks:
- nn_name: readout_model
  nn_architecture:
  - type_layer: Dense
    units: 256
    kernel_regularizer: 0.1
    activation: selu
  - type_layer: Dense
    units: 256
  - type_layer: Dense
    units: 1
    kernel_regularizer: 0.01
    activation: None
```

Рисунок 3.10 – Опис архітектури нейронної мережі

Ця нейронна мережа складається з трьох рівнів Dense. Важливо зауважити, що IGNNTION дозволяє використовувати всі типи шарів, представлені в бібліотеці Keras[18]. Крім того, кожен із цих шарів може мати численні параметри, які налаштовують його властивості. Для цього, знову ж таки, ми підтримуємо всі параметри, прийняті Keras для кожного шару відповідно. Це робиться простим додаванням їх до властивостей кожного шару (наприклад, функція активації в першому щільному шарі). Якщо параметр не визначено (якщо в документації Keras зазначено, що це необов’язковий параметр), тоді IGNNITION використовуватиме параметр за замовчуванням, який використовується Keras.

Також модель використовує ще одну нейронну мережу з назвою recurrent_1 в функції оновлення повідомлень вона складається з одного шару GRU [19], і використовує параметри за умовчуванням. Стробований рекурентний блок (GRU) — це механізм стробування в рекурентних нейронних мережах (RNN), схожий на блок довгострокової пам’яті (LSTM), але без вихідного вентиля. ГРУ намагається вирішити проблему зникаючого градієнта, яка може виникнути за допомогою стандартних рекурентних нейронних мереж.

3.5 Підготовка датасету

Для того щоб можна було проводити навчання нейронної мережі потрібно підготувати датасет. Розробники нейронної мережі надають API [20], за допомогою якого можна створити власний датасет. Але для виконання кваліфікаційної роботи було датасет NSFNET [21], топологія якого зображена на рис. 3.11.

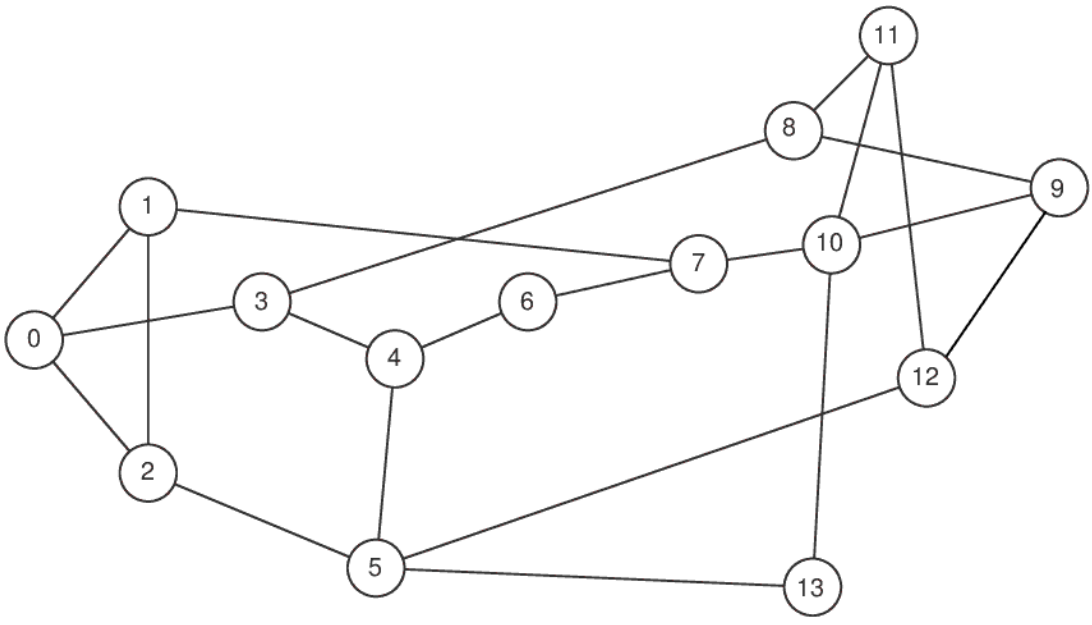

Рисунок 3.11 – Топологія nsfnet

Змінні, які використані в датасеті:

- global_packets: кількість пакетів, переданих у мережі за одиницю часу (пакети/одиницю часу);
- global_losses: пакети, втрачені в мережі за одиницю часу (пакети/одиницю часу);
- global_delay: середня затримка на пакет для всіх пакетів, переданих у мережі (одиниці часу);
- maxAvgLambda: ця змінна використовується для визначення загальної інтенсивності трафіку мережевого сценарію. Зокрема, це максимальна середня швидкість трафіку (біт/с). Потім середня швидкість трафіку кожного шляху src-dst (лямбда) обчислюється як «maxAvgLambda», помножена на випадкове значення від 0 до 1. Зауважте, що цю швидкість трафіку можна розділити на кілька потоків, які надсилають трафік через той самий src- шлях dst;
- performance_matrix: матриця із сукупними вимірюваннями продуктивності src-dst і рівня потоку (затримка, тремтіння та втрати), виміряними на кожній парі src-dst.
- traffic_matrix: матриця з розподілом часу та розміру, що використовується для генерування трафіку для кожної пари src-dst;
- routing_matrix: матриця зі шляхами для з’єднання кожної пари src-dst;
- topology_object: він використовує об’єкт Graph з бібліотеки Networkx, включаючи інформацію, пов’язану з топологією, на рівні вузла та зв’язку.

3.6 Результати роботи нейронної мережі

Після проведення навчання нейронної мережі, отримані результати середньої абсолютної відсоткової помилки в розмірі менше 10% (рис. 3.12). Цей результат є досить точним.

```
Epoch 96/100
1000/1000 [==============================] - 355s 355ms/step - loss: 0.0144 - mean_absolute_percentage_error: 7.0828 - denorm_mean_absolute_percentage_error: 7.1580 -
Epoch 97/100
1000/1000 [==============================] - 325s 325ms/step - loss: 0.0261 - mean_absolute_percentage_error: 10.9980 - denorm_mean_absolute_percentage_error: 10.0310
Epoch 98/100
1000/1000 [==============================] - 349s 349ms/step - loss: 0.0200 - mean_absolute_percentage_error: 9.1064 - denorm_mean_absolute_percentage_error: 10.3559
Epoch 99/100
1000/1000 [==============================] - 350s 350ms/step - loss: 0.0182 - mean_absolute_percentage_error: 8.1419 - denorm_mean_absolute_percentage_error: 6.4669 -
Epoch 100/100
1000/1000 [==============================] - 359s 359ms/step - loss: 0.0910 - mean_absolute_percentage_error: 28.4587 - denorm_mean_absolute_percentage_error: 33.3237
```

Рисунок 3.12 – Навчання нейронної мережі

Також було отримано графік залежності між кількістю епох і втратами зображеного на рис. 3.13, а на рис. 3.14 зображений графік точності передбачень

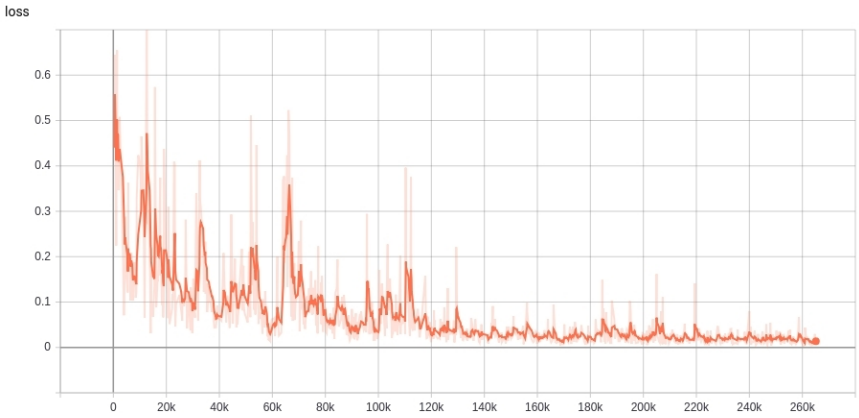

Рисунок 3.13 – Графік залежності к-сті епох до втрат

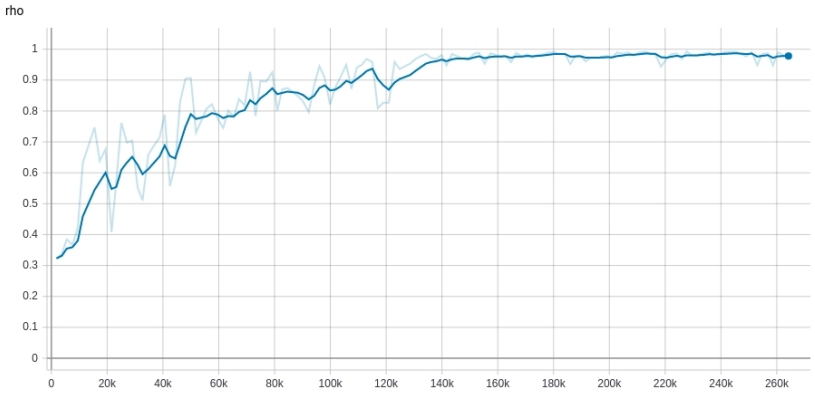

Рисунок 3.14 – Графік точності передбачень

Також було передбачено різні затримки для різних шляхів та порівняно із реальними затримками (рис. 3.15) які були надані в датасеті.

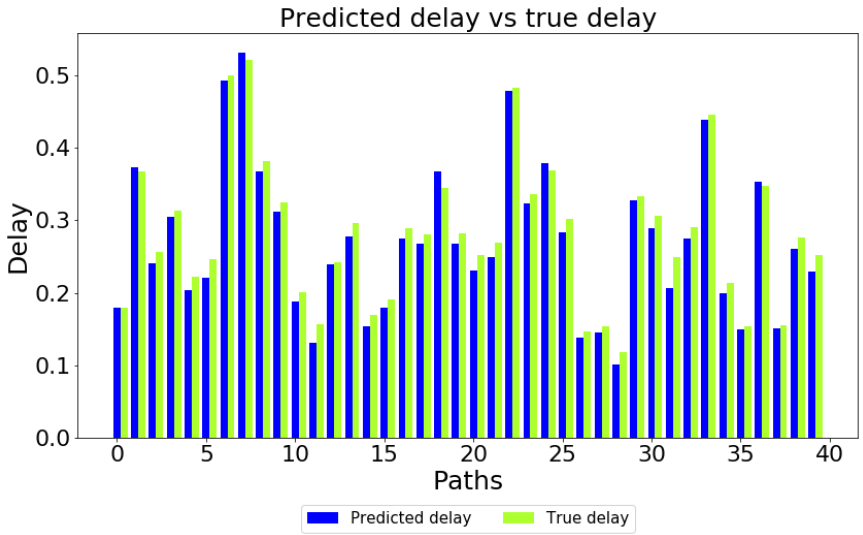

Рисунок 3.15 – Гістограма реальної та передбаченої затримки

У статті [22] було проведено експеримент, метою якого було оптимальне оновлення мережі, після додавання додаткових з’єднань. Результати експерименту зображені у таблиці 3.1.

Таблиця 3.1

Результати обчислень оптимального розташування з'єднання

EVALUATION RESULTS OF THE OPTIMAL LINK PLACEMENT USE CASE

Traffic matrix	Original scenario		RouteNet-based optimizer			Baseline			Relative reduction RouteNet-based vs Baseline	
	Delay	Jitter/delay	Opt. link placement	New delay	New jitter/delay	Most loaded link	New delay	New jitter/delay	Rel. Delay (%)	Rel. jitter/delay (%)
TM_1	0.964	0.345	(11-1)	0.715	0.222	(3-8)	1.377	0.352	48.09%	36.77%
TM_2	0.909	0.335	(9-2)	0.498	0.223	(12-5)	1.065	0.310	53.22%	28.12%
TM_3	0.949	0.374	(11-2)	0.752	0.236	(3-8)	1.312	0.320	42.66%	26.10%
TM_4	1.130	0.339	(12-2)	0.684	0.205	(5-12)	1.387	0.293	50.70%	29.95%
TM_5	0.967	0.314	(10-0)	0.630	0.214	(5-12)	1.321	0.311	52.33%	31.08%
TM_6	1.007	0.362	(11-1)	0.764	0.206	(12-5)	1.192	0.298	35.90%	30.97%
TM_7	1.055	0.379	(9-2)	0.743	0.223	(12-5)	1.087	0.345	31.67%	35.41%
TM_8	0.955	0.360	(11-1)	0.749	0.208	(3-8)	1.281	0.310	41.52%	32.90%

У даному експерименті використовувались 8 матриць маршрутизації в які мали незадовільну втрату пакетів більше 0.1% та високу інтенсивність трафіку. Другий стовпець таблиці вказує на початкову середню затримку в мережі та співвідношення тремтіння/затримка. Третій стовпець вказує де краще додати з'єднання і нові середні показники, отримані за допомогою RouteNet. В четвертому стовпці таблиці була використана стратегія заміни найнавантаженішого з'єднання, на з'єднання із більшою пропускною здатністю. І в останньому стовпці, показана різниця в відсотковому співвідношенні, між цими підходами. RouteNet в середньому на 44% зменшує середню затримку в мережі, ніж підхід збільшення пропускної здатності каналу, і приблизно на 30% зменшує співвідношення тремтіння/затримка сигналу.

Висновки за розділом 3

У даному розділі основну увагу було приділено графовим нейронним мережам, та реалізацією моделі комп'ютерної мережі, за допомогою шляхів та з'єднань. Після реалізації моделі RouteNet, її навчання на тренувальному наборі даних, були отримані досить високі показники прогнозування затримок, тремтіння чи втрат сигналу, що в свою чергу доводить, можливість використання графових нейронних мереж для передбачення поведінки локальної комп'ютерною мережі.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи був проведений аналіз сучасних методів оптимізації локальних комп'ютерних мереж. Особлива увага була приділена двом ключовим аспектам в сфері мережевих технологій: концепції програмованого керування мережею (SDN) та використанню нейронних мереж для прогнозування роботи мережі.

Модель SDN виявилася потужним інструментом для покращення ефективності та управління мережею. Її здатність централізованого керування та програмованості дозволяє забезпечити більш гнучке управління ресурсами мережі, швидку реакцію на зміни та зниження загальних витрат на утримання мережевої інфраструктури.

Нейронні мережі, використані для прогнозування роботи мережі, виявилися ефективними в передбаченні навантаження. Вони забезпечують можливість врахування різноманітних факторів, що впливають на мережевий трафік, та автоматичного вдосконалення свого прогностичного потенціалу з часом.

Сполучення методів SDN та нейронних мереж у цій роботі створює потужний інструментарій для оптимізації локальної комп'ютерної мережі. Це дозволяє досягти балансу між гнучкістю управління та точністю передбачення, що в свою чергу призводить до підвищення продуктивності та ефективності використання ресурсів.

За результатами досліджень можна зробити висновок, що розроблені моделі оптимізації мережі є перспективними для впровадження в реальних умовах функціонування. Дані технології мають потенціал значно покращити якість обслуговування та знизити витрати на утримання мережевої інфраструктури, що робить їх актуальними та важливими для подальшого розвитку сучасних інформаційних технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Б. А. Бугиль, О. А. Лаврів, М. І. Бешлей, В. В. Червенець Методи оптимізації фізичної та логічної структур телекомунікаційних мереж. *Вісник Національного університету "Львівська політехніка". Радіоелектроніка та телекомунікації.* 2013. № 766. С. 78-83. URL: https://science.lpnu.ua/sites/default/files/journal-paper/2017/jun/5107/12bugillavrivbeshleychervenec.pdf (дата звернення: 01.11.2023).

2. Віртуалізація функцій мережі – Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Віртуалізація_функцій_мережі (дата звернення: 10.11.2023).

3. What is Software-Defined Networking?. *IBM*. URL: https://www.ibm.com/topics/sdn (дата звернення: 10.11.2023).

4. A Comprehensive Survey on Knowledge-Defined Networking. *MDPI*. URL: https://www.mdpi.com/2673-4001/4/3/25 (дата звернення: 10.11.2023).

5. Luo T., Tan H.-P., Quek T. Q. S. Sensor OpenFlow: Enabling Software-Defined Wireless Sensor Networks. *IEEE Communications Letters*. 2012. Т. 16, № 11. С. 1896–1899. URL: https://doi.org/10.1109/lcomm.2012.092812.121712 (дата звернення: 14.11.2023).

6. Yu M., Jose J., Miao R. Software defined traffic measurement with opensketch. *USENIX Symposium on Networked Systems Design and Implementation, (NSDI)*. 2013. Т. 13. С. 29–42.

7. P4 / P. Bosshart та ін. *ACM SIGCOMM computer communication review*. 2014. Т. 44, № 3. С. 87–95. URL: https://doi.org/10.1145/2656877.2656890 (дата звернення: 15.11.2023).

8. In-band OAM (iOAM). *GitHub*. URL: https://github.com/CiscoDevNet/iOAM (дата звернення: 15.11.2023).

9. NeMo: an application's interface to intent-based networks". URL: http://nemo-project.net/ (дата звернення: 15.11.2023).

10. RouteNet: Leveraging Graph Neural Networks for Network Modeling and Optimization in SDN / K. Rusek та ін. Conference: the 2019 ACM Symposium. URL: http://dx.doi.org/10.1145/3314148.3314357 (дата звернення: 15.11.2023).

11. Neural Message Passing for Quantum Chemistry / J. Gilmer та ін. *e 34 th International Conference on Machine Learning*. URL: https://doi.org/10.48550/arXiv.1704.01212.

12. Rusek K., Cholda P. Message-Passing Neural Networks Learn Little's Law. *IEEE Communications Letters*. 2019. Т. 23, № 2. С. 274–277. URL: https://doi.org/10.1109/lcomm.2018.2886259 (дата звернення: 19.11.2023).

13. TensorFlow Core | Machine Learning for Beginners and Experts. *TensorFlow*. URL: https://www.tensorflow.org/overview (дата звернення: 17.11.2023).

14. Features. *PyTorch*. URL: https://pytorch.org/features/ (дата звернення: 17.11.2023).

15. Simmons C., Holliday M. A. A Comparison of Two Popular Machine Learning Frameworks. The Journal of Computing Sciences in Colleges. 2019. Т. 35, № 4. С. 20–25. URL: https://www.ccsc.org/publications/journals/SE2019.pdf#page=20 (дата звернення: 17.11.2023).

16. IGNNITION: Bridging the Gap between Graph Neural Networks and Networking Systems / D. Pujol-Perich та ін. *IEEE Network*. 2021. Т. 35, № 6. С. 171–177. URL: https://doi.org/10.1109/mnet.001.2100266 (дата звернення: 20.11.2023).

17. Barcelona Neural Networking Center, 2021. IGNNITION – Documentation. URL: https://ignnition.org/doc/ (дата звернення: 20.11.2023).

18. Chollet, F., та інші, 2023. Keras. URL: https://github.com/keras-team/keras (дата звернення: 20.11.2023).

19. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation/ K. Cho та ін. *Computer Science - Computation and Language* 2014. С. 1–10. URL: https://doi.org/10.48550/arXiv.1406.1078 (дата звернення: 15.11.2023).

20. GitHub - BNN-UPC/datanetAPI at dataset_v0. *GitHub*. URL: https://github.com/BNN-UPC/datanetAPI/tree/dataset_v0 (дата звернення: 20.11.2023).

21. Nsfnet dataset. *Knowledge-Defined networking*. URL: https://knowledgedefinednetworking.org/data/datasets_v0/nsfnet.tar.gz (дата звернення: 20.11.2023).

22. RouteNet: Leveraging Graph Neural Networks for Network Modeling and Optimization in SDN / K. Rusek та ін. *IEEE Journal on Selected Areas in Communications*. 2020. Т. 38, № 10. С. 2260–2270. URL: https://doi.org/10.1109/jsac.2020.3000405 (дата звернення: 15.11.2023).

ДОДАТКИ

ДОДАТОК А

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

Факультет комп'ютерних наук
Кафедра теоретичної та прикладної системотехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) **Магістр**
Галузь знань: **12 – Інформаційні технології**
Спеціальність: **123 «Комп'ютерна інженерія»**

ЗАТВЕРДЖУЮ
Завідувач кафедри теоретичної та прикладної системотехніки
д.т.н., проф. Шматков С. І.

«08 » грудня 2022 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Ющенка Владислава Сергійовича
(прізвище, ім'я, по батькові студента)

1. Тема роботи: «Модель структурної оптимізації локальної комп'ютерної мережі»

керівник роботи: Стрілець Вікторія Євгенівна, кандидат технічних наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «10» листопада 2023 року № 4101-5/3197

2. Строк подання студентом роботи 28 .11.2023

3. Перелік питань, які потрібно розробити
1) Аналіз методів структурної оптимізації комп'ютерних мереж.
2) Створення моделі структурної оптимізації комп'ютерної мережі на основі нейронної мережі.
3) Моделювання структури комп'ютерної мережі з використанням запропонованої моделі.
4) Аналіз результатів моделювання.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Огляд моделей і методів структурної оптимізації комп'ютерних мереж	08.12.22-30.06.23
2	Дослідження параметрів локальної комп'ютерної мережі	01.07.23-15.07.23
3	Пошук наборів даних із інформацією про локальні комп'ютерні мережі	01.05.2023
4	Побудова моделі структурної оптимізації комп'ютерної мережі на основі нейронної мережі та її навчання	15.07.23-30.07.23
5	Оптимізація роботи локальної комп'ютерної мережі за допомогою нейронної мережі	30.07.2023
6	Симуляція роботи локальної мережі до і після оптимізації	01.09.2023
7	Порівняння та аналіз результатів	01.10.2023
8	Формування звітної документації:	15.11.23-28.11.23
9	Оформлення пояснювальної записки	28.11.2023

5. Дата видачі завдання 08.12.2022 р.

Студент __В.С. Ющенко____ _____
ініціали, прізвище підпис

Керівник роботи __В.Є. Стрілець____ _____
ініціали, прізвище підпис

ДОДАТОК Б

ІНДИВІДУАЛЬНЕ ТЕХНІЧНЕ ЗАВДАННЯ

Технічне завдання

на розробку програмного виробу

«Модель оптимізації комп’ютерної мережі на основі нейромережі»

Назва розділу	Назва і зміст підрозділу
1. Введення	1.1. Назва програмного виробу: модель структурної оптимізації локальної комп’ютерної мережі. 1.2. Галузь застосування: оцінювання структури комп’ютерних мереж та прогнозування їх продуктивності і ефективності
2. Підстава для розробки	2.1. Навчальний план за спеціальністю 123 - «Комп’ютерна інженерія» 2.2. Завдання на кваліфікаційну роботу магістра затверджено наказом ректора № _____ від _____.
3. Призначення розробки	3.1. Мета розробки програмного виробу: можливість прогнозувати затримки та тремтіння сигналу під час роботи локальної комп’ютерної мережі. 3.2. Призначення програмного виробу: використання виробу для підвищення рівня роботи комп’ютерних мереж за допомогою використанням нейронних мереж. 3.3. Вихідні дані для розробки: модель затримки, тремтіння і втрати трафіку в роботі комп’ютерної мережі

4. Технічні вимоги до програмного виробу	4.1. Вимоги до функціональних характеристик: можливість прогнозувати значення затримок та тремтіння сигналу в комп'ютерних мереж з використанням графових нейронних мереж. 4.2. Вимоги до надійності: забезпечення працездатності моделі прогнозування затримок тремтіння та втрат сигналу комп'ютерних мережі. 4.3. Вимоги до умов експлуатації: встановлення мови програмування Python та необхідних бібліотек чи фреймворків для роботи моделі прогнозування. 4.4. Вимоги до складу і параметрів технічних засобів: комп'ютер або ноутбук із 8 ГБ оперативної пам'яті та процесором не нижче Intel Core i3 9-го покоління. 4.5. Вимоги до інформаційної та програмної сумісності: ОС Linux або Windows 8/10, підтримка Anaconda, Python 3.8. 4.6. Вимоги до маркування та упаковки: жорстка упаковка з пластмаси, маркування на українській та англійській мовах. 4.7. Вимоги до транспортування і зберігання: транспортування в упаковці будь-яким способом, температура транспортування/зберігання +5-+20 С. 4.8. Спеціальні вимоги: відсутні.
5. Вимоги до програмної документації.	Програмною документацією до виробу «Модель структурної оптимізації локальної комп'ютерної мережі» вважати: 1) Справжнє Технічне завдання на розробку програмного виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи). 2) Програму і методику випробувань розробленого програмного виробу (представити у вигляді Додатку В до пояснювальної записки до кваліфікаційної роботи). 3) Опис програмного виробу (представити в розділах 3, 4 пояснювальної записки до кваліфікаційної роботи). 4) Текст програми (представити в Додатку Г до пояснювальної записки до кваліфікаційної роботи).

<table>
<tr><td>6. Техніко економічні показники</td><td colspan="2">Орієнтовна оцінка якості виконуваного прогнозування на основі середньої абсолютної відсоткової помилки (MAPE≈15%), порівняти точність прогнозованих показників з реальними.</td></tr>
<tr><td rowspan="9">7. Стадії і етапи розробки</td><td>Дата</td><td>Назва етапу</td></tr>
<tr><td>25.12.2022-01.03.2023</td><td>1. Огляд моделей і методів структурної оптимізації комп’ютерних мереж</td></tr>
<tr><td>01.03.2023 - 15.04.2023</td><td>2. Дослідження параметрів локальної комп’ютерної мережі</td></tr>
<tr><td>15.04.2023 - 01.05.2023</td><td>3. Пошук наборів даних із інформацією про локальні комп’ютерні мережі</td></tr>
<tr><td>01.05.2023 - 30.05.2023</td><td>4. Побудова моделі структурної оптимізації комп’ютерної мережі на основі нейронної мережі та її навчання</td></tr>
<tr><td>30.05.2023 - 30.07.2023</td><td>5. Оптимізація роботи локальної комп’ютерної мережі за допомогою нейронної мережі</td></tr>
<tr><td>30.07.2023 - 01.09.2023</td><td>6. Симуляція роботи локальної мережі до і після оптимізації</td></tr>
<tr><td>01.09.2023 - 01.10.2023</td><td>7. Порівняння та аналіз результатів</td></tr>
<tr><td>01.10.2023 - 23.11.2023</td><td>8. Оформлення пояснювальної записки</td></tr>
</table>

8. Порядок контролю і приймання	В даному розділі повинні бути вказані загальні вимоги до приймання розробленого програмного виробу наприклад: 1) Перевірка ходу розробки програмного виробу. Керівнику робіт виконувати 1 раз в 3 тижні. 2) Випробування програмного виробу відповідно до Програми і методики випробувань провести на базі комп'ютерного класу або приватного приміщення. 3) Захист розробленого програмного виробу провести на засіданні атестаційної комісії. 4) Пояснювальну записку надати на паперових носіях в одному примірнику, в електронному вигляді.

Виконавець:
студент групи КІ-61
Ющенко В.С.
_____ _____

Замовник:
кандидат технічних наук
Стрілець В.Є.
_____

ДОДАТОК В

Програма і методика випробувань
програмного виробу

«Модель оптимізації комп’ютерної мережі на основі нейромережі»

1. Об’єкт випробувань

1.1 Найменування випробовуваного програмного виробу: «Модель оптимізації комп’ютерної мережі на основі нейромережі»

1.2 Галузь застосування: оцінювання структури комп’ютерних мереж та прогнозування їх продуктивності і ефективності

1.3 Умовне позначення розробки (при необхідності): відсутнє

2. Мета випробувань

Перевірка правильності функціонування комп’ютерної моделі

3. Загальні положення

3.1 Підстави для проведення випробувань

Підставою для проведення випробувань є наказ про призначення атестаційної комісії.

3.2 Місце і тривалість випробувань

Приймальні (приймально-здавальні) випробування проводяться дистанційно в період роботи атестаційної комісії.

3.3 Обсяг випробувань

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

3.4 Організації, які беруть участь у випробуваннях

Приймальні випробування проводяться атестаційною комісією напередодні засідання за участю Замовника, Виконавця та інших осіб, присутніх на засіданні в дистанційному режимі.

4. Вимоги до програми або програмного виробу

Модель повинна:

1) Представляти з себе комп'ютерну реалізацію прогнозування значень затримок та тремтіння сигналу в комп'ютерних мереж з використанням графових нейронних мереж.
2) Дозволяти користувачу використовувати інші дані відповідного формату для виконання прогнозів.
3) Забезпечувати можливість спостерігати за навчанням моделі; обирати для прогнозування різні моделі; відображати середню абсолютну відсоткову помилку під час прогнозування

4.2 Вимоги до надійності:

1) Програма повинна видавати повідомлення про помилки
2) Програма дозволяє призупиняти процес навчання.

4.3 Вимоги до умов експлуатації:

1) Умови експлуатації відповідають умовам експлуатації будь-якого персонального комп'ютера, який сумісний з програмою
2) Для використання програми потрібно ознайомитись із особливостями формату даних, і як змінювати гіперпараметри мережі

4.4 Вимоги до складу і параметрів технічних засобів: комп'ютер або ноутбук із 8 ГБ оперативної пам'яті та процесором не нижче Intel Core i3 7-го покоління.

4.5. Вимоги до інформаційної та програмної сумісності: ОС Linux або Windows 8/10, підтримка Anaconda, Python 3.8.

4.6. Вимоги до маркування та упаковки: відсутні

4.7. Вимоги до транспортування і зберігання: відсутні

4.8. Спеціальні вимоги: відсутні.

5. Вимоги до програмної документації

Програмною документацією до виробу «Модель структурної оптимізації локальної комп'ютерної мережі» вважати:

1) Справжнє Технічне завдання на розробку програмного виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи).

2) Програму і методику випробувань розробленого програмного виробу (представити у вигляді Додатку В до пояснювальної записки до кваліфікаційної роботи).

3) Опис програмного виробу (представити в розділах 3 пояснювальної записки до кваліфікаційної роботи).

4) Текст програми (представити в Додатку Г до пояснювальної записки до кваліфікаційної роботи).

6. Засоби і порядок випробувань

6.1 Засоби випробувань

Для проведення випробувань необхідний проект для розробки програмної моделі за допомогою мові програмування Python з використання бібліотеки ignnition та вхідним датасетом з параметрами трафіку комп'ютерної мережі.

Випробовування проводиться на технічних засобах, таких як персональний комп'ютер в повній комплектації, чи ноутбук, який має не менше ніж 8Гб оперативної пам'яті, та рекомендований мінімальний процесор Intel Core I3-7 го покоління. Підтримовані операційні системи "Windows" та "Linux".

6.2 Порядок проведення випробувань

1) Перевірка програмної докуметації

1.1 Перевірка складу програмної документації. Перевірку здійснюваи за критерієм наявності, представленої в ТЗ документації.

1.2 Критерієм успішності тесту вважати відповідність наявної документації згідно зі списком в ТЗ та відповідність якості наявної документації згідно з вимогами ЕСПД.

1.3. Перевірка якості програмної документації. Перевірку здійснювати за критерієм відповідності вимогам ЕСПД.

1.4. Модель працює відповідно до умов експлуатації ОС «Windows» та «Linux» із встановленим пакетом python 3.7

Тест 1. Перевірка сумісності моделі з програмним забезпеченням

Критерієм успішності вважати успішний запуск навчання моделі та закінченням першої епохи

Порядок проведення випробувань:

Переконатись в наявності бібліотеки ignition за допомогою команди

pip install -y ignnition

Скопіювати в програмну директорію файли опису тренувань та моделі

```yaml
# OPTIMIZATION OPTIONS
loss: MeanSquaredError
optimizer:
  type: Adam
  learning_rate: 0.001
metrics: [MeanAbsoluteError]

# TRAINING OPTIONS
batch_size: 16
epochs: 100
epoch_size: 3000
shuffle_training_set: True
shuffle_validation_set: False
val_samples: 1000
val_frequency: 1
```

Рисунок В.1 – файл опису тренувань

В корінь директорії додатку розпакувати датасет, і за необхідності за допомогою наступної команди, підготувати його до використання.

python migrate.py -d <PATH TO DATASET> -o <PATH TO OUTPUT> -n <NUM_SAMPLES_PER_FILE> -s <PERCENTAGE_TRAINING>

Шлях до підготовленого датасету необхідно вказати в файлі train_options.yaml

Запустити скрипт програми за допомогою команди python main.py і дочекатись успішного виконання першої епохи, для того щоб закінчити випробування

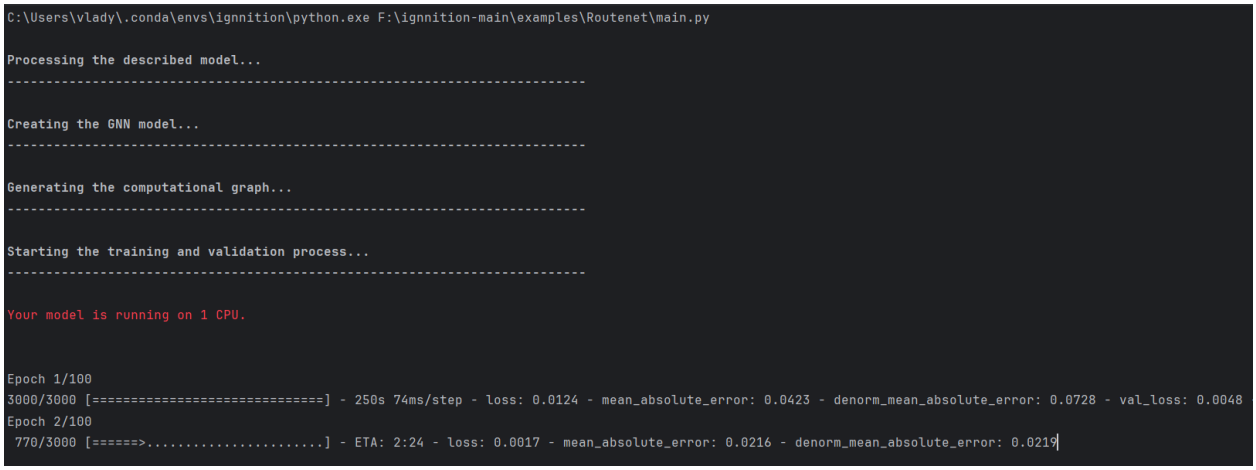

Рисунок В.2 – Успішний вивід програми

Тест 2: Можливість проведення передбачень

Критерій успішності – отримання результатів передбачення.

Після закінчення першої епохи, ми отримали модель яку можна використовувати для виконання передбачень.

Для того щоб виконати передбачення необхідно скопіювати в корінь директорій файл predict.py із Додатку Г, а в файл train_options.yaml додати відповідні шляхи до навченої моделі, і до даних над якими буде проведено передбачення.

Запуск прогнозування виконується за допомогою команди python predict.py

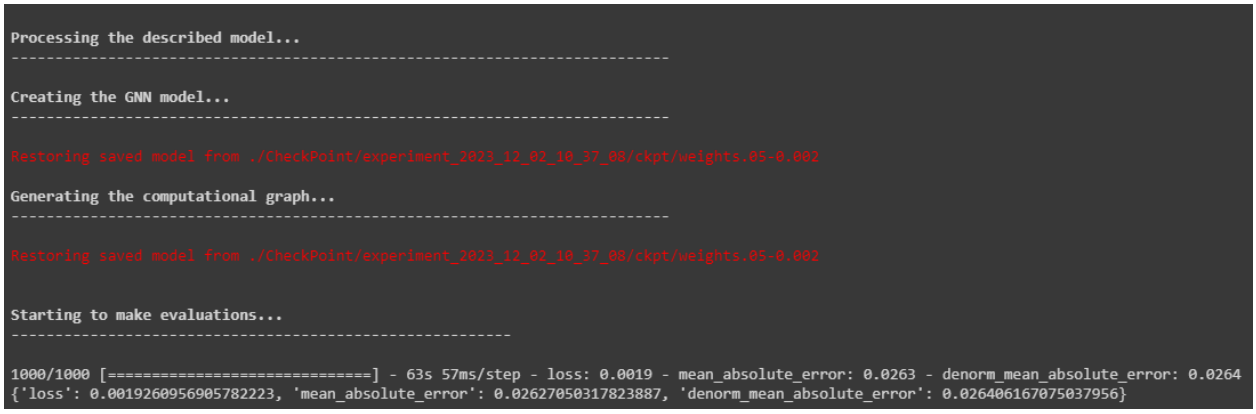

Рисунок В.3 – успішне завантаження моделі та виконання передбачення.

Висновки: випробування пройшло успішно, за наявності необхідної бібліотеки, модель запустилась, провела навчання, та передбачення.

ДОДАТОК Г

Лістинг коду

Файл опису моделі model_description.yaml

```yaml
entities:
- name: link
  state_dimension: 32
  initial_state:
    - type: build_state
      input: [$capacity]

- name: path
  state_dimension: 32
  initial_state:
    - type: build_state
      input: [$traffic]


# Definition of the message passing phase
message_passing:
  num_iterations: 8
  stages:
    # STAGE 1:
    - stage_message_passings:
      # links to paths
      - destination_entity: path
        source_entities:
          - name: link
            message:
              - type: direct_assignment
        aggregation:
          - type: ordered
        update:
          type: neural_network
          nn_name: recurrent1

    # STAGE 2:
    - stage_message_passings:
      # paths to links
      - destination_entity: link
        source_entities:
        - name: path
          message:
          - type: direct_assignment
        aggregation:
          - type: sum
        update:
          type: neural_network
          nn_name: recurrent1

# Definition of the readout
readout:
- type: neural_network
  input: [path]
  nn_name: readout_model
  output_label: [$delay]

# Definition of the Neural Networks
neural_networks:
```

```
- nn_name: readout_model
  nn_architecture:
  - type_layer: Dense
    units: 256
    kernel_regularizer: 0.1
    activation: selu
  - type_layer: Dense
    units: 256
  - type_layer: Dense
    units: 1
    kernel_regularizer: 0.01
    activation: None

- nn_name: recurrent1
  nn_architecture:
    - type_layer: GRU
```

Файл з параметрами тренування train_options.yaml

```
# PATHS
train_dataset: ../data/train
validation_dataset: ../data/validation
predict_dataset: ../data/test
additional_functions_file: ./main.py
output_path: ./
# Set the path to load a trained model
#load_model_path:
./CheckPoint/experiment_2023_11_21_09_46_58/ckpt/weights.08-0.229

# OPTIMIZATION OPTIONS
loss: MeanSquaredError
optimizer:
  type: Adam
  learning_rate: 0.001
metrics: [MeanAbsolutePercentageError]

# TRAINING OPTIONS
batch_size: 1
epochs: 100
epoch_size: 1000
shuffle_training_set: True
shuffle_validation_set: False
val_samples: 5
val_frequency: 1
```

Основний файл запуску тренування main.py

```
import numpy as np
import ignnition

def normalization(feature, feature_name):
    if feature_name == 'delay':
        feature = np.log(feature)
    return feature

def denormalization(feature, feature_name):
    if feature_name == 'delay':
        feature = np.exp(feature)
    return feature
```

```
def main():
    model = ignnition.create_model(model_dir= './code/')
    model.computational_graph()
    model.train_and_validate()
    predictions = model.predict()
    #Do stuff here
    print(predictions)

if __name__ == "__main__":
        main ()
```

Файл predict.py

```
import os
import numpy as np
import pickle

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

import tensorflow as tf
import ignnition

def main():
    model = ignnition.create_model(model_dir='./')
    model.computational_graph()
    all_metrics = model.evaluate()

    convert_to_np = []
    for elem in all_metrics:
        convert_to_np.append(elem.numpy())

    with open('Results.pkl', 'wb') as f:
        pickle.dump(convert_to_np, f)


if __name__ == "__main__":
    main()
```

Файл підготовки датасету migrate.py

```
import numpy as np
import networkx as nx
from datanetAPI import DatanetAPI
import argparse
from networkx.readwrite import json_graph
import json
import tarfile
import os

def generator(data_dir):
    tool = DatanetAPI(data_dir)
    it = iter(tool)
    for sample in it:
        G_copy = sample.get_topology_object().copy()
        T = sample.get_traffic_matrix()
        R = sample.get_routing_matrix()
        D = sample.get_performance_matrix()

        HG = network_to_hypergraph(network_graph=G_copy,
                                   routing_matrix=R,
                                   traffic_matrix=T,
                                   performance_matrix=D)
        yield HG
```

```
def network_to_hypergraph(network_graph, routing_matrix, traffic_matrix,
performance_matrix):
    G = (nx.DiGraph(network_graph)).copy()
    R = np.copy(routing_matrix)
    T = np.copy(traffic_matrix)
    P = np.copy(performance_matrix)

    D_G = nx.DiGraph()
    for src in range(G.number_of_nodes()):
        for dst in range(G.number_of_nodes()):
            if src != dst:
                D_G.add_node('p_{}_{}'.format(src, dst),
                             entity='path',
                             traffic=T[src, dst]['Flows'][0]['AvgBw'],
                             delay=P[src, dst]['AggInfo']['AvgDelay'])

                if G.has_edge(src, dst):
                    D_G.add_node('l_{}_{}'.format(src, dst),
                                 entity='link',
                                 capacity=int(G.edges[src,
dst]['bandwidth']))

                for h_1, h_2 in [R[src, dst][i:i + 2] for i in range(0,
len(R[src, dst]) - 1)]:
                    D_G.add_edge('p_{}_{}'.format(src, dst),
'l_{}_{}'.format(h_1, h_2))
                    D_G.add_edge('l_{}_{}'.format(h_1, h_2),
'p_{}_{}'.format(src, dst))

    D_G.remove_nodes_from([node for node, out_degree in D_G.out_degree() if
out_degree == 0])

    return D_G


def migrate_dataset(input_path, output_path, max_per_file, split):
    gen = generator(input_path)
    data = []
    file_ctr_train = 0
    file_ctr_eval = 0
    topology = input_path.split('/')[1]

    tmp_dir = output_path+"tmp/"
    os.system("rmdir /s /q %s" % (tmp_dir))
    os.makedirs(tmp_dir)

    counter = 0
    while True:
        try:
            G = next(gen)
            parser_graph = json_graph.node_link_data(G)
            data.append(parser_graph)
            if counter == max_per_file:
                a = np.random.rand()
                path = output_path
                if a < split:
                    path += 'eval/'
                    with open(tmp_dir+'data.json', 'w') as json_file:
                        json.dump(data, json_file)

                    tar = tarfile.open(path + topology + "sample_" +
str(file_ctr_eval) + ".tar.gz", "w:gz")
                    tar.add(tmp_dir+'data.json', arcname="data.json")
```

```
                    tar.close()
                    os.remove(tmp_dir+'data.json')
                    file_ctr_eval += 1
                else:
                    path += 'train/'
                    with open(tmp_dir+'data.json', 'w') as json_file:
                        json.dump(data, json_file)

                    tar = tarfile.open(path + topology + "sample_" +
str(file_ctr_train) + ".tar.gz", "w:gz")
                    tar.add(tmp_dir+'data.json', arcname="data.json")
                    tar.close()
                    os.remove(tmp_dir+'data.json')

                    file_ctr_train += 1

                data = []
                counter = 0
            else:
                counter +=1

        # when finished, save all the remaining ones in the last file
        except:
            a = np.random.rand()
            path = output_path
            if a < split:
                path += 'eval/'
                with open(tmp_dir+'data.json', 'w') as json_file:
                    json.dump(data, json_file)

                tar = tarfile.open(path + topology + "sample_" +
str(file_ctr_eval) + ".tar.gz", "w:gz")
                tar.add(tmp_dir+'data.json', arcname="data.json")
                tar.close()
                os.remove(tmp_dir+'data.json')
            else:
                path += 'train/'
                with open(tmp_dir+'data.json', 'w') as json_file:
                    json.dump(data, json_file)

                tar = tarfile.open(path + topology + "sample_" +
str(file_ctr_train) + ".tar.gz", "w:gz")
                tar.add(tmp_dir+'data.json', arcname="data.json")
                tar.close()
                os.remove(tmp_dir+'data.json')
            os.system("rmdir /s /q  %s" % (tmp_dir))
            break

if __name__ == "__main__":
    # python migrate.py -d ../../../nsfnetbw/ -o ./datansfnet/ -n 100 -s 0.8
    # Parse logs and get best model
    parser = argparse.ArgumentParser(description='Parse file and create
plots')

    parser.add_argument('-d', help='Origin data directory', type=str,
required=True, nargs='+')
    parser.add_argument('-o', help='Output data directory', type=str,
required=True, nargs='+')
    parser.add_argument('-n', help='Number of samples per file', type=int,
required=True, nargs='+')
    parser.add_argument('-s', help='Percentage split of files used for
TRAINING. 1-percentage will be added to EVALUATION set.', type=float,
required=True, nargs='+')
```

```
    args = parser.parse_args()

    origin_dir = args.d[0]
    output_dir = args.o[0]
    split = 1-float(args.s[0])
    num_samples_file = args.n[0]

    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    if os.path.exists(output_dir+'/eval'):
        os.system("rmdir /s /q  %s" % (output_dir))

    if os.path.exists(output_dir+'/train'):
        os.system("rmdir /s /q  %s" % (output_dir))

    os.makedirs(output_dir+'/eval')
    os.makedirs(output_dir+'/train')

    migrate_dataset(origin_dir, output_dir, num_samples_file, split)
```

Файл необхідного API datanetAPI.py можна завантажити за посиланням https://github.com/BNN-UPC/ignnition/blob/main/examples/Routenet/datanetAPI.py