Universitat de Girona

NETWORK PERFORMANCE PREDICTION USING GRAPH NEURAL NETWORKS: APPLICATION TO NETWORK SLICING

Miquel Farreras Casamort

Network performance prediction using Graph Neural Networks: application to network slicing

DOCTORAL THESIS

Miquel Farreras Casamort

2024

Network performance prediction using Graph Neural Networks: application to network slicing

DOCTORAL THESIS

Miquel Farreras Casamort

PhD program in Technology

Directed by:
Pere Vilà Talleda
Lluís Fàbrega Soler

This thesis is presented in fulfillment of the requirement for the conferral of the degree of Doctor of Philosophy by the Universitat de Girona

May 31st, 2024

Dr. Miguel Camelo
Principal Investigator/Research Lead
Network Intelligence research track
IDLab, University of Antwerp in collaboration with imec
The Beacon | Sint-Pietersvliet 7 | 2000 Antwerpen | Belgium

Subject: Certification of completion of a PhD stay

In my role as Senior Researcher at University of Antwerp - IMEC, I confirm that the PhD student Miquel Farreras Casamort from the University of Girona, with NIF document 41563096A, has been hosted in the facilities of the research group IDLab Antwerp (Antwerpen, Belgium) during his research stay from September 1st to December 1st of 2022 (3 months).

The proposed working plan was applied, where Miquel Farreras Casamort has collaborated with researchers in our research group on subject related to applying artificial intelligence in 5G networks. Part of the results of this research stay, a publication with the title “Improving network delay predictions using GNNs” has been submitted to the Journal of Network and Systems Management. In addition, novel research on the use of advanced Graph Neural Networks to solve the slicing problem in 5G Network Slicing has started. We expect that in the next weeks the results of this work will be submitted to a journal/conference.

The host’s details are the following:

Surname: Camelo Botero
Given name: Miguel Hernando
Department: IDLab Research Group
University/Research Institute: University of Antwerp - IMEC
Street: The Beacon, Sint-Pietersvliet, 7
Town and Postcode: 2000 Antwerpen
Country: Belgium
Telephone: +32468264434
E-mail: miguel.camelo@uantwerpen.be, miguel.camelo@imec.be

Miguel Camelo

Date: 01/December/2022

Related Author's Publications

Here there is a list of the international journals and proceedings of international conferences where this work has been published or is being reviewed:

[1] M. Farreras, P. Soto, M. Camelo, L. Fàbrega and P. Vilà. "Predicting network performance using GNNs: generalization to larger unseen networks". In *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, 2022, pp. 1-6*. `https://doi.org/10.1109/NOMS54207.2022.9789766`.

[2] M. Farreras, P. Soto, M. Camelo, L. Fàbrega and P. Vilà. "Improving Network Delay Predictions Using GNNs". In *Journal of Networks and Systems Management 31, 65, 2023*. Q1 journal. `https://doi.org/10.1007/s10922-023-09758-9`.

[3] M. Farreras, J. Paillissé, L. Fàbrega and P. Vilà. "Generation of a Network Slicing Dataset: The Foundations for Ai-Based B5G Resource Management". In *Data in Brief, Volume 55, 2024*. Q3 journal. `https://doi.org/10.1016/j.dib.2024.110738`.

[4] M. Farreras, J. Paillissé, L. Fàbrega and P. Vilà. "GNNetSlice: A GNN-based Performance Model to support Network Slicing in B5G Networks". In "Computer Networks". Q1 journal. Under review.

Other unrelated Author's Publications

Other relevant publications not related to the topic of this thesis have been produced concurrently with the thesis work:

[5] E. Calle, D. Martínez, R. Brugués-i-Pujolràs, M. Farreras, J. Saló-Grau, J. Pueyo-Ros, L. Corominas. "Optimal selection of monitoring sites in cities for SARS-CoV-2 surveillance in sewage networks". In *Environment International, Volume 157, 2021*. Q1 journal. `https://doi.org/10.1016/j.envint.2021.106768`.

[6] E. Calle, D. Martínez, G. Buttiglieri, L. Corominas, M. Farreras, J. Saló-Grau, P. Vilà, J. Pueyo-Ros, J. Comas. "Optimal design of water reuse networks in cities through decision support tool development and testing". In *npj Clean Water 6, 23, 2023*. Q1 journal. `https://doi.org/10.1038/s41545-023-00222-4`.

Acknowledgments

I would like to start by saying that many people have helped me achieve the completion of this thesis. First, I want to thank my supervisors, Lluís Fàbrega and Pere Vilà, for their support, patience, help, and guidance through all these years. I also want to express my gratitude to Miguel Camelo, Paola Soto and Jordi Paillissé; it has been a pleasure to collaborate on the articles with you all.

I would also want to thank the members and former members of the Broadband Communications and Distributed Systems (BCDS) group, particularly Eusebi, Josep Lluís, Teo, Antonio, David, Sergi, Aniol, Oriol, Gerard, Marc, Martí, Roser, Joan, and many others, for making this journey more enjoyable.

Additionally, I want to thank all the IDLab members (Miguel, Paola, Julián, David, Esra, Esmeralda, Nina) who hosted me at IMEC in Antwerp, Belgium. You all made me feel at home. I also want to thank Elisa, my flatmate during that stay, for the good friendship we made. I also thank Oscar, Robert, Roger and all my friends for giving me moral support for finishing this thesis.

A special mention goes to my girlfriend, Laia. She has given me all her love, support, and patience to accomplish this work.

Finally, I want to thank all my family for their unconditional support. Thank you father, mother, uncle, grandpa and specially grandma, who left us during this thesis. I dedicate this work to all of you.

This work has been partially funded by the Generalitat de Catalunya through a Consolidated Research Group project (grant reference SGR-01125), the Secretaria d'Universitats i Recerca del Departament d'Empresa i Coneixement de la Generalitat de Catalunya for the FI-SDUR fellowship funding 2020 FISDU 00590 assigned to Miquel Farreras.

Acronyms

3GPP 3rd Generation Partnership Project. 44, 63

5G Fifth Generation. 18, 20, 36–38, 62

AI Artificial Intelligence. xiii, xiv, 2, 4, 6, 11, 12, 18, 20, 40, 42, 60, 61, 77, 79, 80

B5G Beyond 5G. xiii–xv, 1–5, 7, 9–11, 15, 18, 39, 42, 50, 59, 60, 77–81

BNN-UPC Barcelona Neural Networking Center - Universitat Politècnica de Catalunya. xii, 21, 22, 38, 40

CNN Convolutional Neural Network. 12, 15, 61

DL Deep Learning. 18, 20, 21, 62

DT Digital Twin. 42, 60, 61

DTN Digital Twin Network. 2–4, 18, 45

eMBB Enhanced Mobile Broadband. xiii–xv, 1, 9, 38, 42–47, 49, 51, 54, 55, 57, 63, 70, 72, 73, 75, 78

GAIN Girona Antwerp Intelligence for Networks. x–xii, 22, 28–39, 79–81

GNN Graph Neural Network. x, xiii–xv, 2, 4, 5, 7, 11–15, 17–21, 33, 38–40, 50, 58–60, 62–67, 78–81

GPU Graphics Processing Unit. 31, 36, 37, 71

IoT Internet of Things. 10, 20, 62

IoV Internet of Vehicles. 1, 18

ITU International Telecommunication Union. 6

KPI Key Performance Indicator. viii, xiii–xv, 1, 2, 4–7, 13–18, 20, 34, 37–42, 44, 45, 59–63, 71, 72, 74, 79, 81

LSTM Long Short-Term Memory. 61

LTE Long-Term Evolution. 9

MAE Mean Average Error. 72, 76, 79

MAPE Mean Average Percentage Error. xi, 19, 29–32, 36, 37, 70

MEC Multi-access Edge Computing. 1, 37

mIoT Massive Internet of Things. xiii–xv, 1, 10, 38, 42, 44–47, 49, 51, 54, 55, 57, 63, 70, 72, 73, 78

ML Machine Learning. 2, 4, 6, 15, 18, 20, 21, 39, 40, 42, 60–63, 69, 71, 77, 79, 80

MSE Mean Square Error. 72, 76

NFV Network Function Virtualization. 1, 7, 10

NN Neural Network. 18, 28

QoE Quality of Experience. 1, 4

QoS Quality of Service. 1, 3–5, 10, 15, 18, 38, 42–44, 46, 50, 63, 64

RAN Radio Access Network. xi, 42, 43, 46, 48, 49, 56, 57, 62, 63, 79

RNN Recurrent Neural Network. 12, 15, 18–20, 61, 62

R^2 Coefficient of Determination. 72, 76, 79

SDN Software Defined Networking. 1, 7–11, 20, 62

SFC Service Function Chaining. 1

SLA Service Level Agreement. 2, 4, 5, 18, 37, 42, 43, 45, 60, 63, 64, 78

SMAPE Symmetric Mean Average Percentage Error. xi, 70–76, 79

SVM Support Vector Machine. 61

UE User Equipment. 46

URLLC Ultra-Reliable Low-Latency Communication. xiii–xv, 1, 4, 9, 38, 42–47, 49, 51, 54, 55, 57, 63, 70, 72, 73, 76, 78

WFQ Weighted Fair Queuing. xi, 42–45, 47, 55, 63, 67, 69

Contents

List of Figures

List of Tables

Abstract

This thesis addresses key challenges in the optimization of network slicing in Beyond 5G (B5G) networks, focusing on the use of Graph Neural Networks (GNNs) for performance prediction and resource allocation. It is structured into three main parts: improvement of an existing GNN model for Key Performance Indicator (KPI) prediction, dataset creation for network slicing, and the creation of a GNN model for predicting network slicing KPIs.

The ultimate goal of this work is to build a model for predicting network slicing KPIs. GNNs models are a novel and powerful technique for accurately learning from graph-structured data, making them suitable for predicting network KPIs. To learn GNNs programming, the first part of this work describes the participation in a ITU challenge. Autonomous network management is explored, being essential for the dynamic environments expected in B5G networks. The limitations of traditional modeling tools and network simulators are also explored, proposing GNNs as an effective alternative due to their high accuracy and low computational requirements. A significant contribution is the enhancement of the RouteNet baseline model, achieving an improvement in prediction accuracy for larger networks, in comparison to the networks seen during training.

As the goal is to build a GNNs model for predicting network slicing KPIs, and a lack of data containing network slicing scenarios is identified, the second part presents a the creation of a network slicing dataset designed to support Artificial Intelligence (AI)-based performance prediction in B5G networks. This dataset, generated through a packet-level simulator, includes diverse network scenarios with varying topologies, slice instances, and traffic flows, capturing the complexities of Enhanced Mobile Broadband (eMBB), Ultra-Reliable Low-Latency Communication (URLLC), and Massive Internet of Things (mIoT) slices. The dataset is a valuable resource for the research community, facilitating innovations in network slicing and resource management.

After creating the required data, the GNN model called GNNetSlice is developed in part three, introducing a novel model that leverages GNNs to predict the performance of network slices in the core and transport network. By adopting a data-driven approach, GNNetSlice balances prediction speed and accuracy. The model demonstrates high accuracy in predicting delay, jitter, and losses across various scenarios.

Overall, this thesis makes contributions to the field of network slicing, providing tools and datasets for efficient and accurate KPI prediction in B5G networks. The proposed models and datasets pave the way for more resilient and adaptive network management solutions, crucial for the next generation of mobile networks.

Resum

Aquesta tesi aborda els reptes clau en l'optimització del network slicing en xarxes Beyond 5G (B5G), centrant-se en l'ús de Graph Neural Networks (GNNs) per a la predicció del rendiment i l'assignació de recursos. La tesi s'estructura en tres parts principals: millora d'un model GNN existent per a la predicció de Key Performance Indicators (KPIs), creació de conjunts de dades sobre network slicing i un model de GNN per predir KPIs de network slicing.

L'objectiu final d'aquest treball és construir un model per predir els KPIs de network slicing. Els models GNNs són una tècnica nova i potent per aprendre amb precisió a partir de dades estructurades en grafs, la qual cosa els fa adequats per predir KPIs de xarxa. Per aprendre la programació de GNNs, la primera part d'aquest treball descriu la participació en un challenge organitzat per la ITU. S'explora la gestió autònoma de xarxes, que és essencial per als entorns dinàmics de les xarxes B5G. També s'exploren les limitacions de les eines de modelatge tradicionals i dels simuladors de xarxa, proposant les GNNs com una alternativa eficaç a causa de la seva alta precisió i els seus relativament baixos requeriments computacionals. Una contribució significativa és la millora del model base RouteNet, aconseguint una millora en la precisió de predicció per a xarxes més grans, respecte les vistes durant l'entrenament.

Ja que l'objectiu és construir un model basat en GNNs per predir els KPIs de network slicing, i s'identifica una manca de dades que continguin escenaris de network slicing, la segona part presenta la creació d'un conjunt de dades de network slicing dissenyat per donar suport a la predicció del rendiment basada en Artificial Intelligence (AI) a les xarxes B5G. Aquest conjunt de dades, generat a través d'un simulador a nivell de paquets, inclou diversos escenaris de xarxa amb diferents topologies, instàncies de slice i fluxos de trànsit, capturant les complexitats dels tipus de slice Enhanced Mobile Broadband (eMBB), Ultra-Reliable Low-Latency Communication (URLLC) i Massive Internet of Things (mIoT). El conjunt de dades és un recurs valuós per a la comunitat investigadora, ja que facilita les innovacions en la gestió de recursos i el network slicing.

Després de crear les dades necessàries, el model GNN anomenat GNNetSlice es desenvolupa a la tercera part, introduint un nou model que aprofita GNNs per predir el rendiment dels network slices al core i la xarxa de transport. En adoptar un enfocament basat en dades, GNNetSlice equilibra la velocitat i la precisió de predicció. El model demostra una gran precisió a l'hora de predir retards, fluctuacions de retard i pèrdues en diversos escenaris.

En general, aquesta tesi fa contribucions al camp del network slicing, proporcionant eines i conjunts de dades per a una predicció KPI eficient i precisa a les xarxes B5G. Els models i conjunts de dades proposats obren el camí per a solucions de gestió de xarxes més resistents i adaptatives, crucials per a la propera generació de xarxes mòbils.

Resumen

Esta tesis aborda los retos clave en la optimización del network slicing en redes Beyond 5G (B5G), centrándose en el uso de Graph Neural Networks (GNNs) para la predicción del rendimiento y la asignación de recursos. La tesis se estructura en tres partes principales: mejora de un modelo GNN existente para la predicción de Key Performance Indicators (KPIs), creación de conjuntos de datos sobre network slicing y creación de un modelo GNN para predecir KPIs de network slicing.

El objetivo final de este trabajo es construir un modelo para predecir los KPIs de network slicing. Los modelos GNNs son una técnica nueva y potente para aprender con precisión a partir de datos estructurados en grafos, lo que los hace adecuados para predecir KPIs de red. Para aprender la programación GNNs, la primera parte de este trabajo describe la participación en un challenge organizado por la ITU. Se explora la gestión autónoma de redes, que es esencial para los entornos dinámicos de las redes B5G. También se exploran las limitaciones de las herramientas de modelado tradicionales y de los simuladores de red, proponiendo las GNNs como una alternativa eficaz debido a su alta precisión y sus relativamente bajos requerimientos computacionales. Una contribución significativa es la mejora del modelo base RouteNet, consiguiendo una mejora en la precisión de predicción para redes de mayor tamaño al visto durante el entrenamiento.

Puesto que el objetivo es construir un modelo basado en GNNs para predecir los KPIs de network slicing, y se identifica una carencia de datos que contengan escenarios de network slicing, la segunda parte presenta la creación de un conjunto de datos de network slicing diseñado para apoyar la predicción del rendimiento basada en IA en las redes B5G. Este conjunto de datos, generado a través de un simulador a nivel de paquetes, incluye varios escenarios de red con diferentes topologías, instancias de slice y flujos de tráfico, capturando las complejidades de los tipos de slice Enhanced Mobile Broadband (eMBB), Ultra-Reliable Low-Latency Communication (URLLC) y Massive Internet of Things (mIoT). El conjunto de datos es un valioso recurso para la comunidad investigadora, ya que facilita las innovaciones en la gestión de recursos y el network slicing.

Después de crear los datos necesarios, el modelo GNN llamado GNNetSlice se desarrolla en la tercera parte, presentando un nuevo modelo que aprovecha GNNs para predecir el rendimiento de los network slices en el core y la red de transporte. Al adoptar un enfoque basado en datos, GNNetSlice equilibra la velocidad y precisión de predicción. El modelo demuestra una gran precisión a la hora de predecir retrasos, fluctuaciones de retraso y pérdidas en varios escenarios.

Por lo general, esta tesis hace contribuciones al campo del network slicing, proporcionando herramientas y conjuntos de datos para una predicción KPI eficiente y precisa en las redes B5G. Los modelos y conjuntos de datos propuestos abren el camino para soluciones de gestión de redes más resistentes y adaptativas, cruciales para la próxima generación de redes móviles.

1 Introduction

In this chapter, the stage for the exploration about automated network management is set. The motivation behind this research is clarified, identifying key challenges and trends within the field. Subsequently, the specific objectives are outlined, highlighting the goals aimed to achieve. Moreover, a discussion of the significant contributions this thesis aspires to make to the domain of network management is included. To assist readers in navigating this thesis, an outline of the thesis structure is also provided.

1.1 Motivation

Ensuring smooth operation of network infrastructures requires an accurate network management. This includes various functions, such as preventing network failures and restoring them quickly when they occur, maintaining proper equipment and software configurations, tracking network usage, optimizing network performance to handle increased traffic without service degradation, and numerous related tasks. Consequently, network management encompasses a wide range of mechanisms and shares a close connection with network control, involving automatic processes to configure the network.

In recent years, the landscape of network management has undergone a deep transformation with the advent of Beyond 5G (B5G) networks. These networks bring unprecedented advancements in bandwidth, reduced latency, and a host of capabilities that cater to both end-users and network operators. They are envisioned to support a large number of connected devices and facilitate applications and services demanding high-speed, low-latency connectivity, including Ultra-Reliable Low-Latency Communication (URLLC), Massive Internet of Things (mIoT), and Enhanced Mobile Broadband (eMBB) [4]. A critical component of enabling these services is the creation and management of network slices, which are logical virtual networks that overlay a shared physical network and computing resources (Fig. 1). Especifically, URLLC can also include edge functions, to reduce the latency of the applications, while mIoT or eMBB have less stringent latency restrictions.

However, amid the promise of these networks lies a formidable challenge—dynamic resource allocation for network slices [60]. Network slices must be orchestrated swiftly and effectively, adapting in (near)-real-time to ensure stringent Quality of Service (QoS) and Quality of Experience (QoE) requirements. These requirements encompass parameters such as latency thresholds, minimum bandwidth guarantees, and support for a multitude of connected devices, vital for applications like virtual reality and smart stadiums [23]. As networks evolve, technologies such as Software Defined Networking (SDN), Network Function Virtualization (NFV), Service Function Chaining (SFC), and Multi-access Edge Computing (MEC) become integral [72], requiring rapid analysis of Key Performance Indicators (KPIs) to manage the available resources in a near real-time way.

The challenge deepens as networks embrace emerging paradigms like the Internet of Vehicles (IoV) [95], demanding swift adaptation to evolving network conditions. This urgency

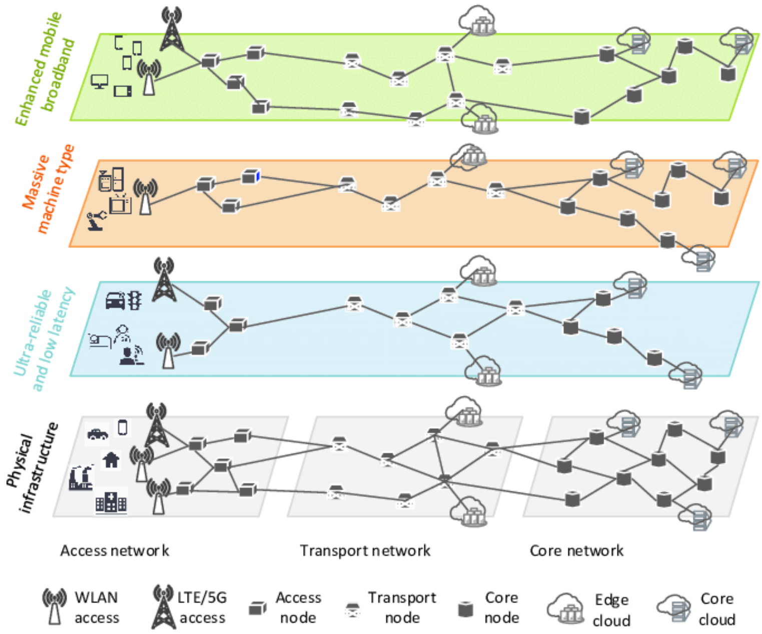

Figure 1: Network slicing usages in a real scenario [44].

underscores the need for highly automated and agile management systems that can effectively balance demand, benefits, and capacity across existing and emerging network slices.

Traditionally, network modeling relied on mathematical analysis but often oversimplified the dynamic nature of modern networks [96]. In contrast, network simulators provided accurate results but at the cost of processing time and resource utilization, making real-time analysis, monitoring, and management a complex task [65].

In response to these challenges, recent years have witnessed a surge in leveraging Artificial Intelligence (AI) and Machine Learning (ML) techniques for resource management in B5G networks. Data-driven approaches, including Deep Learning, Reinforcement Learning, and Graph Neural Network (GNN), have shown remarkable promise in optimizing resource allocation, fault detection, and traffic management [85]. These approaches use the large amount of data generated by B5G networks, adapting quickly to changing network conditions and optimizing resource utilization to meet Service Level Agreements (SLAs).

The evolving techniques for predicting network KPIs create the possiblity of obtaining a Digital Twin Network (DTN), a virtual replica or representation of a physical network. It mimics the behavior, characteristics, and interactions of its real-world counterpart in a digital environment. Fig. 2 shows a closed control loop where multiple configurations are created and validated using the results of the DTNs. When a configuration is obtained and validated, it is applied to the physical network. Digital twins have gained substantial traction in various industries, including manufacturing, healthcare, urban planning, and infrastructure management. They serve as invaluable tools for monitoring, analyzing, and optimizing complex systems. In the case of networks, DTN can be applied to obtain multiple benefits [10]:

- Network design and optimization: simulation of various deployment scenarios, leading to more efficient and cost-effective network designs.
- Predictive maintenance: monitor the health and performance of network components, by analyzing real-time data. Digital twins can anticipate equipment failures or degradation, allowing operators to proactively address issues.
- Resource allocation and load balancing: traffic patterns, device density, and QoS requirements analyzed to dynamically allocate network resources.
- Security and threat detection: controlled environment for testing security measures and identifying vulnerabilities.
- Real-time performance monitoring: monitor network performance metrics, including latency, throughput, and packet loss.
- Energy efficiency: by analyzing power consumption patterns and optimizing resource utilization, operators can reduce the environmental footprint of their networks.

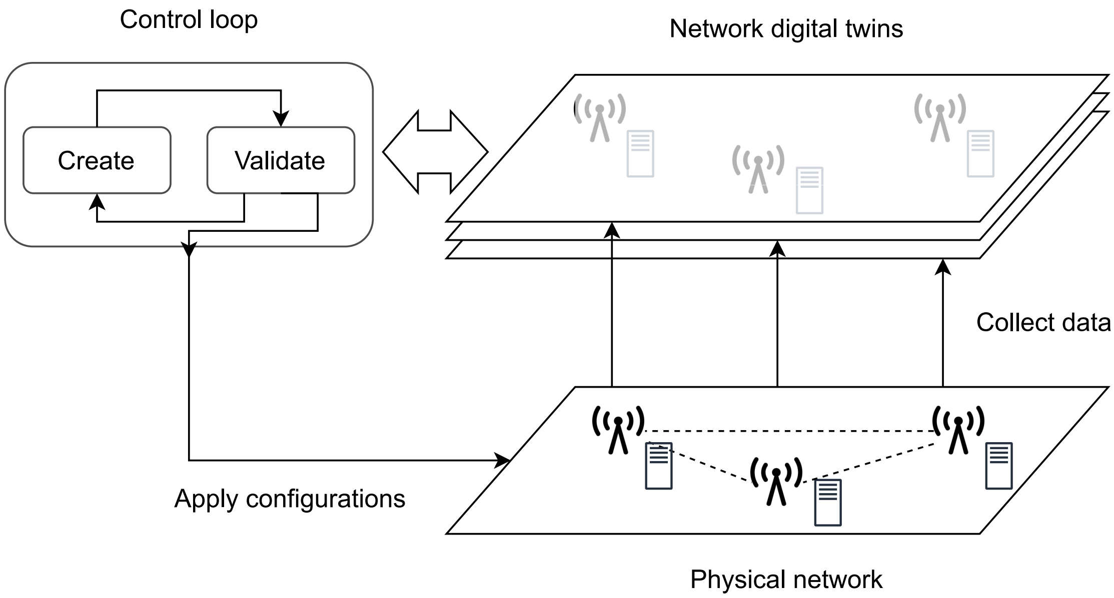

Figure 2: Digital Twin Networks of a physical network, used in a closed control loop.

While the integration of digital twins in B5G networks offers significant advantages, it also presents challenges such as data integration, model accuracy, and computational demands. As B5G networks evolve and expand, further research and development will be necessary to refine digital twin technologies for even more precise modeling and optimization.

Digital twins applied to B5G networks represent a transformative approach that empowers operators to build, manage, and optimize next-generation communication infrastructures more effectively. By providing real-time insights, predictive capabilities, and enhanced control, digital twins play a pivotal role in unlocking the full potential of B5G technology for various industries and applications.

The challenges posed by the dynamic and complex nature of B5G networks demonstrate the need for advanced techniques in managing network resources efficiently. The concept of DTN emerges as a powerful tool to tackle these challenges. By creating a virtual replica of the physical network, DTN provides a controlled environment to simulate various network scenarios, predict network behavior, and analyze critical KPIs. This capability plays a key role in addressing the intricate demands of network slicing infrastructures, such as dynamic resource allocation and real-time adaptation to QoS and QoE requirements. Integrating DTN into the network management process allows operators to explore multiple configurations and evaluate their impact on network performance, optimizing the deployment and orchestration of network slices.

In this context, the thesis focuses on predicting essential KPIs—using state-of-the-art AI and ML techniques, such as Graph Neural Networks (GNNs)—to enhance the precision and scalability of network slice management. These models can be used as DTNs, to provide valuable feedback that helps to validate and adjust the network configurations, ensuring that these proposed configurations align with real-world performance and SLA requirements. Through this approach, this thesis aims to improve the efficiency and reliability of network slicing infrastructures.

1.2 Objectives

The aim of this thesis is to develop fast and accurate prediction models for critical network KPIs. These KPIs play a key role in the management of network slicing infrastructures, impacting the quality and efficiency of services provided to users and applications.

Specifically, the KPIs considered relevant in this thesis include:

- **Latency:** The time delay between sending and receiving data, which is crucial for applications requiring low-latency connectivity such as URLLC.
- **Packet loss rate:** The percentage of data packets lost during transmission, which can degrade service quality and lead to poor user experiences.
- **Jitter:** The variation in packet arrival times, which affects real-time services like video conferencing and online gaming.

In modern network management, the prediction of network KPIs presents a challenge. Traditional methods, such as network calculus, often lack the precision required, while network simulators, though accurate, can be prohibitively slow. To address this issue, the thesis explores the application of cutting-edge artificial intelligence and machine learning techniques, seeking solutions to enhance the accuracy and speed of KPI predictions in network management.

By achieving this goal, this thesis seeks to empower network operators with tools for efficient network slicing management. This allows the network operators to meet diverse performance demands across various slices while ensuring optimal resource allocation, and accomplish the SLAs.

In addition to facilitating efficient network resource management, the thesis intends to contribute to overall network optimization. The predictive models developed here will aid in the allocation of resources, ultimately resulting in improved network performance.

Ultimately, this work aspires to advance the capabilities of network slicing technology, making it a more dynamic and responsive component within modern network infrastructures. Through these objectives, the thesis seeks to provide practical solutions that enhance the functionality and efficiency of network slicing infrastructures.

1.3 Contributions

The main contributions of this thesis are as follows:

1. **Enhanced GNN:** This research significantly contributes to the field by participating in a challenge aimed at improving a GNNs model. By refining and enhancing GNNs, by applying several techniques, such as normalization, feature engineering, and post-processing of predictions, among others, this research has paved the way for more accurate and effective predictions in network management. These contributions advance the state-of-the-art by improving the scalability and accuracy of GNN models for real-time network management, particularly in B5G scenarios.

2. **A novel dataset for network slicing research:** This thesis generates a novel dataset through simulations of network slicing infrastructures using the *OMNeT++* network simulator, which is widely recognized in the field of network research. The simulation environment follows established standards, from 3GPP for B5G and network slicing architectures. The dataset simulates various network slice types, traffic, and KPI requirements. This contribution is significant as it provides the research community with a benchmark dataset for KPI prediction, which is currently lacking in this field.

3. **Innovative GNN model:** This research introduces the application of a GNN model for predicting network KPIs such as delay, jitter and packet losses of network slicing networks. While GNN models have been explored in other domains, this work pioneers their application to the dynamic and complex problem of real-time KPI prediction in network slices. The novel aspect lies in adapting the GNN model to handle the unique requirements of network slicing, including different traffic patterns, stringent QoS requirements, and adaptation to network changes. This extends

the frontier of AI/ML techniques applied to network management, surpassing the capabilities of traditional approaches like network calculus or simulators.

These contributions collectively advance the state-of-the-art in network management by providing enhanced prediction tools, valuable simulation resources, and modeling techniques. They offer practical solutions to the challenges associated with network KPI predictions, ultimately benefiting the efficiency and reliability of network slicing infrastructures.

1.4 Outline of the Thesis

This document is organized into six chapters, each contributing to the comprehensive exploration of network management and KPI prediction in network slicing infrastructures.

Chapter 1 serves as the introduction, setting the stage for the entire document. It outlines the motivation behind this research, its specific objectives, and the contributions it aims to make to the field of network management. Additionally, this chapter provides a clear outline of the thesis structure.

Chapter 2 explores into the background of network slicing dynamic management. It encompasses three main sections. Firstly, it reviews network management functions, standards, and architectural classifications while discussing recent trends. Secondly, it summarizes the background in artificial intelligence and machine learning, specifically those techniques enabling dynamically manageable networks. Lastly, it introduces various approaches for network simulations and network calculus.

Chapter 3 shows the experience acquired with the participation in a International Telecommunication Union (ITU) challenge to improve a Graph Neural Network model to predict network on network KPIs. The problem, approaches used and results are described in this section.

Chapter 4 is dedicated to the generation of a network slicing dataset, where different traffic is used to create a dataset of simulations, using a network packet simulator. The proposed scenario, simulation configurations and results are shown in this section.

Chapter 5 presents the innovative approach achieved for prediction of multiple network KPIs with high precision and low resource requirements. The model is described and evaluated in this chapter.

Finally, Chapter 6 concludes this document by summarizing its main contributions to network slice management and KPI prediction in network slicing infrastructures. It also outlines potential areas for further research and development within the field.

2 Background

This chapter offers a contextual background for this thesis research, primarily centered around network management and control techniques, such as SDN, NFV, and notably, network slicing. It also introduces artificial intelligence techniques, machine learning, and Graph Neural Networks to simulate networks, all particularly within the domain of B5G network slicing and the construction of Digital Twin Networks. In addition, tools for simulating networks are discussed. This contextualization forms the basis for this research, which focuses on using GNNs to predict and optimize the prediction of KPIs of network slices within these evolving telecommunications landscapes.

2.1 Network management

Network management [73] is an essential processes to ensure the optimal functioning of a network. A primary approach to state-of-the-art techniques for B5G networks management is introduced in this section. Mainly two network technologies are related to this thesis, namely Software Defined Networking and Network Slicing.

2.1.1 Software Defined Networking

SDN is a modern approach to networking [84] that aims to make networks more flexible, programmable, and efficient by separating the control plane from the data plane. This separation allows network administrators to manage and control network traffic and services through software, providing greater agility and adaptability compared to traditional networking approaches. In conventional networking, such as the use of routers and switches, the control plane and data plane are tightly integrated within network devices. The control plane is responsible for determining how data packets should be forwarded through the network, while the data plane is responsible for actually forwarding the packets. On the other hand, SDN decouples the control plane and data plane [71]. The control plane is centralized in software, making high-level decisions about traffic management and routing, while the data plane in network devices simply follows the instructions provided by the control plane.

The key components of SDN are shown in Fig. 3:

- Northbound Interface and Application Plane: These interfaces expose the functionality of the SDN controller to applications and higher-level network services. Application developers can use these APIs to create custom network applications and services.
- SDN Controller (Control Plane): The central component of an SDN architecture is the SDN controller. It acts as the "brain" of the network, responsible for making global traffic management decisions based on network policies and conditions. Popular SDN controllers include OpenDaylight, ONOS, and Ryu [75].

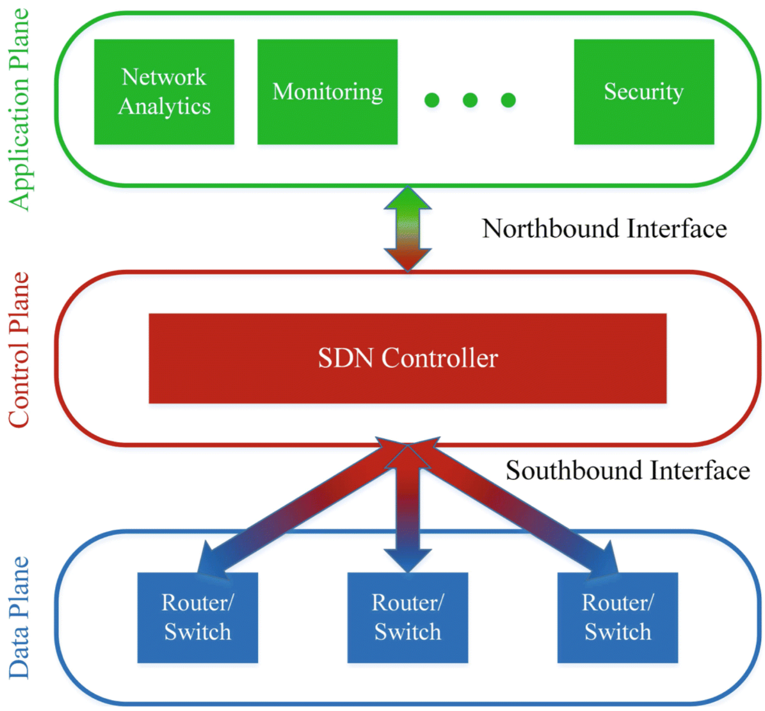

Figure 3: SDN architecture in a three-layer approach [71].

- Southbound Interface and Data Plane: These interfaces connect the SDN controller to the network devices in the data plane. Protocols like OpenFlow [60] are commonly used for communication between the controller and network switches and routers. Southbound APIs allow the controller to instruct network devices on how to handle traffic.

The main advantages of using SDN over conventional networking are [88]:

- Network Programmability: SDN allows for flexible network configuration and control through software, making it easier to adapt to changing requirements.
- Centralized Management: The centralized control plane provides a global view of the network, simplifying network management, monitoring, and troubleshooting.
- Dynamic Traffic Engineering: SDN can dynamically adjust traffic routing based on real-time conditions, optimizing network performance and resource utilization. This is the main funcionality where this thesis is focused.
- Security: SDN can enforce security policies and isolation by controlling traffic flows, making it easier to respond to threats and vulnerabilities.
- Vendor Independence: SDN reduces vendor lock-in, as the control plane is separate from network hardware, enabling the use of heterogeneous network devices.

Conversely, some challenges and considerations must be considered and properly addressed in SDN:

- Scalability: Large-scale SDN deployments require robust controllers and efficient communication mechanisms to handle the increased network complexity.
- Security: Centralized control introduces security risks, making the SDN controller a potential target for attacks.
- Interoperability: Ensuring compatibility and interoperability between different SDN components and vendors can be challenging.
- Transition: Migrating from traditional networking to SDN can be complex and may require changes to existing infrastructure.

SDN has gained traction in data centers, cloud environments, and enterprise networks, as it offers greater flexibility, automation, and control over network resources, allowing organizations to meet the demands of modern applications and services effectively. New protocols and inter-domain management techniques are being developed, increasing the opportunities to use SDN in core networks.

2.1.2 Network Slicing

Network slicing is a closely related concept to SDN and is a crucial component of next-generation telecommunications networks, particularly in the context of B5G [70]. It enables the creation of multiple virtualized network instances also called slices, within a single physical network infrastructure, each tailored to serve specific applications, services, or customer requirements, reserving all the required resources for the network service [2].

Network slicing is a network architecture paradigm that allows network operators to partition their physical infrastructure into multiple logically isolated and independent virtual networks, each with its own dedicated resources, characteristics, and service attributes. These virtual networks are referred to as "slices". These slices are created, managed, and customized according to the unique needs of different use cases (Fig. 1), as defined by 3GPP (TS-23.501 [2]):

- Enhanced Mobile Broadband (eMBB): a B5G service category primarily focused on delivering high-speed, high-capacity, and low-latency wireless connectivity to support applications that require massive data throughput. Significantly enhanced mobile broadband experience compared to previous generations of wireless networks (e.g., 4G Long-Term Evolution (LTE)). For example it can be used with 4K/8K video streaming, virtual reality experiences, augmented reality applications, and ultra-fast internet browsing.
- Ultra-Reliable Low-Latency Communication (URLLC): B5G service category designed to provide ultra-reliable and extremely low-latency communication for critical applications. It is mainly focused in applications where safety and real-time responsiveness are critical, such as autonomous vehicles, remote surgery, industrial automation, and public safety communications.

- massive Internet of Things (mIoT): also called Massive Machine Type Communications(mMTC), mIoT is a B5G service category tailored for scenarios with a massive number of connected devices. It focuses on providing efficient, scalable, and energy-efficient communication for a vast number of low-power, low-data-rate devices. It is suitable for a wide range of Internet of Things (IoT) applications, including smart cities, industrial automation, agriculture, environmental monitoring, and asset tracking.

The essential elements enabling network slicing include [3]:

- SDN: Network slicing relies heavily on the principles of SDN. In SDN, the control plane is decoupled from the data plane, allowing centralized control and management of network resources. Network operators use SDN controllers to orchestrate and configure network slices dynamically.
- Virtualization: Network functions and resources, including compute, storage, and network elements, are virtualized to create multiple instances that can be allocated to different slices based on their requirements. NFV plays a significant role in achieving this virtualization. NFV is a transformative technology that involves the virtualization of network functions that traditionally run on dedicated, proprietary hardware appliances, and instead, these functions are implemented in software and run on standard hardware infrastructure.
- Service Orchestration: Service orchestration systems are responsible for designing, provisioning, and managing network slices based on the specific needs of various applications and services. These systems ensure that each slice receives the necessary resources and QoS.

A wide range of benefits of using network slicing in SDN are [60]:

- Customization: Network operators can customize each network slice to meet the performance, latency, security, and bandwidth requirements of different services and applications, ensuring optimal service delivery.
- Resource Efficiency: Network slicing maximizes the efficient use of physical network infrastructure by allowing the allocation of resources on-demand. This reduces operational costs and capital expenditure.
- Isolation: Slices are logically isolated from each other, ensuring that one slice's performance or security issues do not impact others. This is critical for services with varying degrees of sensitivity and criticality.
- Scalability: Network slicing enables the network to scale dynamically as new services and applications are introduced, without compromising the performance of existing slices.

In summary, network slicing, closely integrated with SDN principles, is a crucial technology in modern telecommunications networks, offering the flexibility and customization required to meet the diverse and evolving needs of various services, applications, and industries in the era of B5G.

2.2 Artificial intelligence

In today's increasingly interconnected world, the efficient management and optimization of network resources have become critical. Ensuring that networks deliver reliable services while efficiently utilizing resources is a complex challenge. Deep Learning, a subfield of machine learning, has emerged as a transformative technology to address these challenges [4]. In particular, GNNs have proven to be a powerful tool for analyzing and optimizing network structures. This section provides a comprehensive introduction to these topics for better understanding.

2.2.1 Machine Learning and Deep Learning

Machine Learning is a subfield of AI that allows computers to learn and make decisions from data, without being explicitly programmed [83]. It's a paradigm that empowers machines to recognize patterns, make predictions, and improve their performance through experience. Machine learning involves the following key components:

- Data: machine learning relies on data as its foundation. This data can encompass various types, such as text, images, numerical values, depending on the task.
- Algorithms: mathematical models and statistical techniques used to analyze and process data. These algorithms learn from the data and extract the meaningful patterns.
- Training: in supervised training, machine learning models are exposed to labeled data, where the correct outcomes are provided alongside the input data. The model learns to make predictions by adjusting the internal parameters to minimize errors.
- Testing and evaluation: after training, the model's performance is checked using unseen data to ensure that it generalizes well to real-world situations. Evaluation metrics are used to measure the model precision.

The machine learning model's training can be supervised (where models learn from labeled data), unsupervised learning (where models identify patterns without labeled data), and reinforcement learning (where models make decisions to maximize rewards, in a dynamic way).

Deep Learning is a specialized subset of machine learning that focuses on using artificial neural networks [42]. Those are inspired by the structure of the human brain, to solve

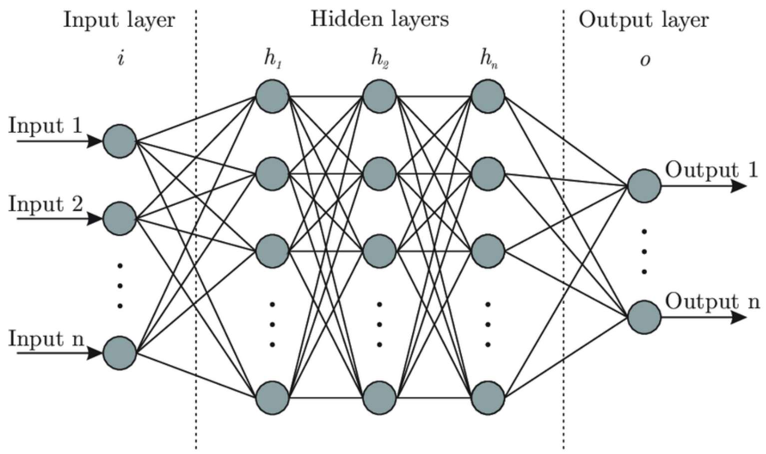

Figure 4: Artifficial Neural Network architecture [21].

complex tasks. Deep learning has the capability to handle and automatically learn hierarchical representations of data through deep neural networks. The enablers to build deep learning models are the neural networks. Deep learning models are built using artificial neural networks. These networks consist of interconnected layers of neurons, each layer performing computations on data and passing it to the next layer. The depth of these networks, with multiple hidden layers, distinguishes deep learning from traditional machine learning. The layers made of neurons are divided in three different types: input layers, hidden layers (which can have a variable number of layers) and output layers (Fig. 4).

Another key requirement for builidng deep learning models are big amounts of data [74]. Deep learning works better when the models are trained with large datasets, which allow it to discover complex patterns and relationships. The more data, the better deep learning models can perform.

In summary, machine learning is a broader concept that encompasses the field of deep learning. Machine learning includes a range of techniques, while deep learning focuses on neural networks with multiple layers to automatically learn complex representations of data. Deep learning models, especially Convolutional Neural Networks (CNNs) for images and Recurrent Neural Networks (RNNs) for sequences, have achieved groundbreaking results in various AI applications [97].

2.2.2 Graph Neural Networks (GNN)

Networks are inherently graph structures, where nodes represent entities (e.g., routers, antennas), and edges denote connections or interactions between these entities. Traditional neural networks are designed for grid-like data, such as images or sequences, and may not fully exploit the underlying graph structure of networks. This is where GNNs come into play. GNNs extend deep learning to graph-structured data, enabling the modeling of complex relationships and dependencies within networks.

GNNs have risen as a solution to tackle a broad spectrum of challenges associated with graph-structured data [50]. These challenges span various domains, from social networks and molecular structures to complex computer networks [39, 28]. GNNs leverage the inherent interactions between nodes and edges in a graph to make predictions or carry out tasks such as regression and classification [81]. At the core of GNNs lies the concept of message-passing, a mechanism where information flows among neighboring nodes, updating the features of the target node.

The predictions a GNN can provide can be at different levels of detail within graph data:

- Graph-Level tasks: categorize entire graphs, useful for social network analysis and text classification.
- Node-Level tasks: label individual nodes, e.g., predicting user relations in social networks based on interactions, categorize.
- Link-Level tasks: forecast connections between nodes or predict properties of the edges, aiding link prediction for potential friendships in social networks.

In a GNN, a graph is represented as an assembly of nodes and edges, each containing specific features. The GNN architecture is composed of multiple layers. At each layer, information flows throughout the graph, gathering relationships from the neighboring nodes and edges. This characteristic is also known as the expressive power [80]. The message-passing process is composed of three stages: message transformation, aggregation, and update:

1. Message transformation functions govern how nodes and edges generate messages.
2. Aggregation functions combine these messages.
3. Update functions merge the aggregated messages into the features of the target node, enhancing its understanding of the broader network context.

Ultimately, a readout function [69] is used to perform the essential classification or regression tasks inherent to GNNs. The message-passing process expands iteratively across the input graph, as illustrated in Fig. 5. In this figure, the example of predicting values for the node A is shown. Each node, denoted by distinct colors, contributes its features to vectors. These vectors, that can be seen as an array of values, are used as the input values for the neural network architecture. The primary objective is to predict values for the target node, labeled as "A". The process initiates with message transformation, followed by aggregation, and culminates in the update of the target node's feature values. In cases where specific node attributes are missing, a pooling technique is used to consolidate information from these nodes [68].

Due to all the benefits exposed in this chapter for creating network models, this thesis main focus of attention is to apply GNNs network KPI prediction. In the next section, other options for network modeling are discussed.

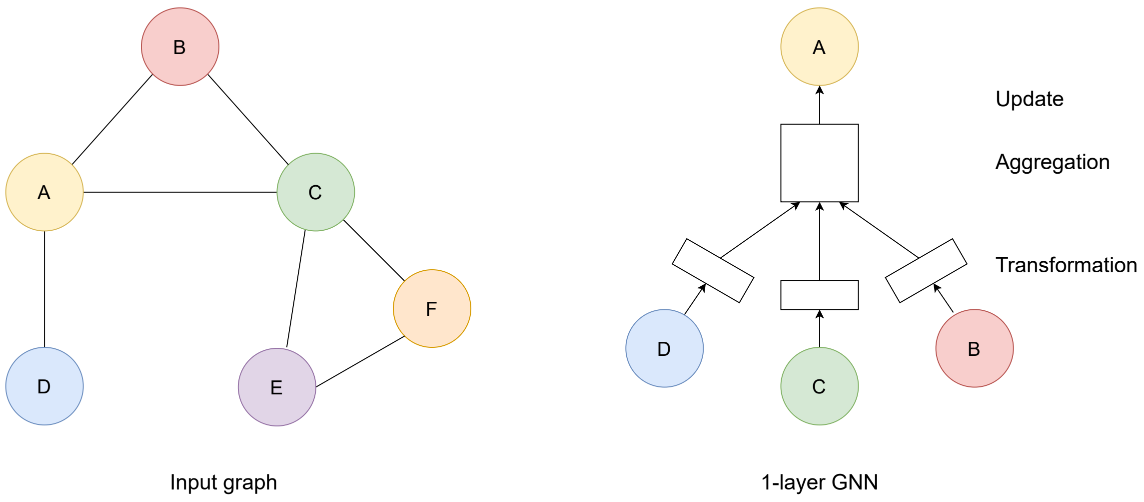

Figure 5: GNN basic structure with message transformation, aggregation and update [50].

2.3 Network modeling

In the dynamic landscape of modern networking, the ability to model, analyze, and optimize network performance is of crucial importance. The available tools, apart from deep learning techniques, are network simulators and network calculus. Both are briefly introduced in this section.

2.3.1 Network simulators

Network simulators are software platforms designed to replicate real-world network environments, allowing to experiment with various network scenarios in a controlled and virtual context. These simulators enable the simulation of diverse network topologies, traffic patterns, and routings, all without the need for costly and complex physical setups.

One well-known example of a network simulator is OMNeT++ [92]. It provides a versatile framework for modeling and simulating networks, being a valuable resource for exploring network behaviors and generating performance evaluations. OMNeT++ can be used to check the impact of network changes and test the robustness of network designs under different conditions.

2.3.2 Queuing theory

Queuing theory [24, 22], conversely, is a mathematical theory that offers a systematic approach to analyze the performance of networks. It provides a set of tools to determine bounds on various network metrics, such as delay, jitter, throughput, and packet loss.

Queuing theory employs mathematical concepts to model the flow of traffic through a network, taking into account factors like service curves, traffic arrival patterns, and buffer capacities. By mathematically calculating network behavior the estimated KPIs of the analyzed network are obtained.

Apart from GNNs, these techniques can and have been used in predicting network performance. In the literature, network models are used for this objective and are typically created using analytical modeling, which includes queuing theory [24], Markov chains [22], and other similar techniques. Such models can produce fast but poor predictions as a result of relying on unrealistic and static assumptions about real-world networks. Packet-level network simulators are often utilized to model network behavior under specific cases and get more realistic results. Nevertheless, their longer computations and execution times when dealing with larger networks [92] make them unsuitable for the required fast predictions. Despite this, network simulators produce highly accurate data which can be employed to train different ML models. This is the reason this thesis uses a network simulator for generating new network slicing data, while it uses a GNN for predicting the performance.

While various machine learning techniques, including CNNs, RNNs, and traditional machine learning algorithms, have been applied to network optimization problems, GNNs offer unique advantages for modeling networked data. The graph structure of communication networks, where nodes represent devices (e.g., routers, antennas) and edges represent the connections between them, aligns naturally with the design of GNNs.

Unlike CNNs, which work well with grid-like data, or RNNs, which are designed for sequential data, GNNs can efficiently handle arbitrary, non-Euclidean structures such as networks. Traditional machine learning models, such as random forests or support vector machines, often require manual feature extraction and struggle with capturing the complex, interdependent relationships in network topologies. In contrast, GNNs leverage message-passing mechanisms to aggregate and update node features, allowing for more precise predictions of KPIs in dynamic environments like B5G networks [35].

Moreover, GNNs have demonstrated superior performance in tasks involving the prediction of node-level, edge-level, and graph-level attributes, making them an ideal choice for network KPI prediction [33]. This is crucial for network slicing, where the behavior of individual network slices need to be modeled accurately to meet stringent QoS requirements.

In summary, this thesis advances the current state-of-the-art in several ways:

- **Enhanced GNN architecture:** Building on existing GNN models, this thesis improves the representation of dependencies in network slicing environments. This allows for more accurate KPI predictions in real-time, addressing the limitations of previous static and computationally expensive models.

- **Novel application to network slicing:** While GNNs have been applied in other domains, their use in dynamic network slicing KPI prediction is still limited. This thesis uses GNNs to model the interactions between network slices and their impact on QoS requirements, providing a more dynamic and scalable solution.

- **Benchmark dataset:** A key contribution of this work is the creation of a new dataset for network slicing simulations, generated using the OMNeT++ simulator. This dataset serves as a benchmark for future research in KPI prediction and network slice optimization, filling a gap in current research resources.

Each section of the work presented in this thesis includes a more detailed analysis of the state-of-the-art for each advance performed.

3 Improving a Graph Neural Network model for KPI prediction

In this chapter, it was necessary to study how the GNN models worked before creating a new one. At the same time, an ITU challenge focused on predicting network KPIs was published. This challenge served as an ideal learning method, as it provided a baseline model that needed to be improved for better generalization, and a step by step procedure was applied to learn and see the improvements.

This part of the thesis describes the step-by-step approach to outperform RouteNet [78], a well-known and state-of-the-art GNN model designed to predict network performance metrics. This baseline was proposed during the 2021 Graph Neural Networking Challenge: Creating a Scalable Network Digital Twin, which was part of the ITU AI/ML in 5G Challenge [49]. The contributions of the thesis in this chapter are summarized as follows[1]:

- A GNN model is proposed. It predicts the network delay with high accuracy and is scalable compared to the baseline model RouteNet by exploiting feature engineering techniques and fundamental networking concepts.
- A detailed description of the approach is provided, including the model design and an exhaustive analysis of the network properties related to the dataset used to train the model. This aspect is fundamental to support the decisions on the different models designed and can be used as a methodology to create GNN to solve related problems.
- An extensive set of experimental evaluations is performed, where this thesis approach is compared with recent state-of-the-art GNN approaches and analytical models, and discuss how different changes to the baseline model affect the model's performance. Real and synthetic datasets are used for a comprehensive prediction accuracy evaluation.

From this work, two publications were achieved:

[1] M. Farreras, P. Soto, M. Camelo, L. Fàbrega and P. Vilà. "Predicting network performance using GNNs: generalization to larger unseen networks". In *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, 2022, pp. 1-6*. `https://doi.org/10.1109/NOMS54207.2022.9789766`.

[2] M. Farreras, P. Soto, M. Camelo, L. Fàbrega and P. Vilà. "Improving Network Delay Predictions Using GNNs". In *Journal of Networks and Systems Management 31, 65, 2023*. Q1 journal. `https://doi.org/10.1007/s10922-023-09758-9`.

[1] This chapter of the thesis describes the work done before and during the stay in the IDLab (group from IMEC and the University of Antwerp). The collaboration with the IDLab started in mid 2021, thanks to the participation in the challenge. A stay of three months, from September 2022 to December 2022 has been performed in the IDLab research group in Antwerp, Belgium.

3.1 Problem context

AI/ML, particularly GNNs, are being proposed as the next generation of predictive models. GNNs have gained significant attention in the networking community due to their ability to learn complex relationships among nodes and links [11]. This makes them more suitable for this domain compared to classical ML techniques or Deep Learning (DL) [85].

Recently, GNNs have been suggested as DTNs [8]. As introduced previously, DTN represents a virtual real-time depiction of a physical network in the digital world [101]. Digital twins have sparked a revolution in various industries, providing an enhanced view of physical entities to support diagnosis and what-if analyses. The requirements for near-real-time network analysis are closely linked to the concept of DTNs, which is still in the early stages of adoption for Fifth Generation (5G) and B5G networks [6, 51].

In general, ML is becoming a crucial enabler for developing new DTNs for these networks, as the demands of new services grow increasingly stringent. For instance, in smart cities, DTNs are utilized to design networks that can ensure QoS without wasting energy, as discussed in [7]. Moreover, the emerging IoV services also rely on DTNs for network planning, which can incorporate Neural Networks (NNs) and even incorporate techniques like Federated Learning [52].

An interesting use case in B5G networks is a dynamic, real-time, routing optimization tool that accomplishes the QoS requirements. These networks continuously evolve and require an adaptive behavior, where routing can be decisive in achieving the SLA requirements. In this case, different routing configurations can be tested, selecting the best result according to the target KPIs. This tool should have the following functionalities:

- Monitor the current resource allocations and performance over the entire network.
- Detect or be notified by an external actor, of new network events that might affect the KPIs.
- Test multiple routing configurations and choose the best tested configuration for the real network.
- Minimize the used resources while satisfying the SLAs.
- Apply new network routing configurations.

3.2 Baseline

RouteNet was used as a baseline model for this work. RouteNet is a GNN model whose principal function is to predict per-source-destination network KPIs given a specific network configuration. An RNN GNN exchanges information of the graph and saves the states to the memory cell of each recurrently until convergence. In particular, as shown in Fig. 6, RouteNet learns the network model from the data collected over different network topologies, such as their traffic and routing configurations, which can later be used to predict their performance. The resulting model captures the complex relationships between the properties of links and the source-destination paths in topologies.

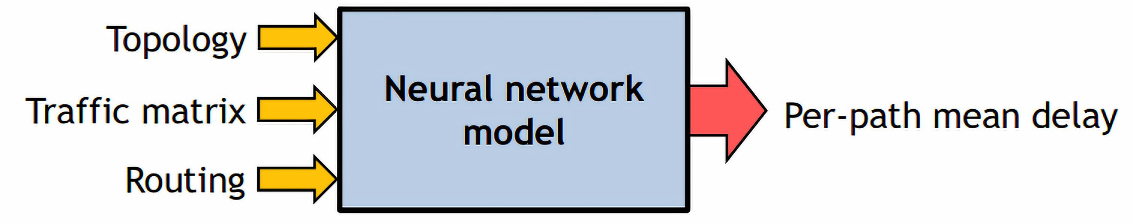

Figure 6: RouteNet model inputs and output [78].

The representation of the graph that will feed the GNN is the first issue to be addressed in its design. Edge prediction is a limitation of current state-of-the-art GNNs, despite GNNs being able naturally to support them [41]. Then, the input graph has to be adapted to read and predict link features. The input graph of RouteNet converts links into nodes, creating a hyper-graph for each sample, where two node types exist: path and link. The main working principle of RouteNet is the differentiation of the data between the path level (e.g., end-to-end delay, end-to-end packet loss) and the link level (e.g., link delay, link utilization). This information is encoded in learnable vectors. Based on this assumption, the message-passing procedure used by RouteNet [78] follows the principles:

1. The state of a path depends on the state of all the links on the path.
2. The state of a link depends on the state of all the paths that traverse the link.

The learnable vectors are used to execute a message-passing procedure (with a determined number of rounds) to collect messages from all the links and paths. This message-passing procedure, combined with the usage of RNNs, makes the GNN architecture capable of inferring path or link-level metrics. In RouteNet, each RNN's hidden state represents a function with the information of the path or link. These RNNs have modifiable hyperparameters to adapt to the specific use case.

In this work, Mean Average Percentage Error (MAPE) is used as the prediction error. The MAPE is the average value of the relative errors, in percentage, as shown in Eq. (1). The relative error is calculated by the difference between the predicted value $\hat{y}_l$ and the real value y_i, divided by y_i and obtaining the result in absolute value. The lower the MAPE value, the better the predictions.

$$MAPE = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{\hat{y}_l - y_i}{y_i} \right| \tag{1}$$

3.3 Related work

A predictive model and a decision-making algorithm are the main blocks needed to achieve the automated network management tasks described in Section 3.1. In this work, the focus is on improving a prediction model called RouteNet, which predicts the network performance according to a given topology, routing, and offered traffic.

AI/ML solutions such as shallow Feed Forward Neural Networks, Support Vector Machine (SVM), Random Forest (RF), Reinforcement Learning (RL), Multi-Layer Perceptrons (MLP), or Convolutional Neural Networks (CNN) have been used to resolve similar 5G challenges. Some state-of-the-art solutions apply these methods for predicting network delay, throughput, or other KPIs. In [13], 3D-CNNs are applied to the problem of provisioning resources for network slices in real-time, predicting network capacity requirements depending on the time of the week.

In [12], a zero-touch control for network slicing is proposed, predicting each slice's network capacity needs. The authors in [53] implement a Logistic Regression (LR), an SVM, and a Decision Tree (DT) to predict the delays experienced by the users. Following a DL approach, [37] predicts the traffic that a network will support at a specific time, similarly to [13, 12], but using a RNN. [5] also uses RNNs to predict delays in 5G networks for IoT and Tactile Internet. All these ML solutions work well with static scenarios but are less adapted to create graph models in comparison to GNNs. The main advantage of GNNs over other ML methods is their expressive power [80], allowing to model different graphs (e.g., different topology, number of nodes) with higher accuracy. Some additional motivations to apply GNNs to the network performance prediction are:

- GNNs have been successfully applied to combinatorial optimization problems [14], and can achieve relational reasoning [11].
- A graph representation can better capture the relationship of the nodes in a network, i.e., there is a one-to-one mapping between the network topology and the graph representation.

In that sense, RouteNet [78] is a GNN model that can directly learn from graph-like data, including network topologies, routing configurations, and offered traffic, to predict network KPIs. In recent research, other GNN solutions have been applied to solve autonomous network management [36], network slicing monitoring [76], network slicing control [90], or SDN end-to-end delay prediction [38]. However, GNNs, as ML algorithms, are sensitive to the mismatch between training data and testing data, i.e., training and testing data do not come from the same distribution. Therefore, one of the main challenges in using GNNs for predicting network performance is to improve the out-of-distribution generalization, understood as the difference in accuracy between training and testing data [48].

Several works already attempt to improve the generalization capabilities of GNNs. The authors of [29, 56] presented PARANA, a GNN model that extracts path and link-level features related to Queuing Theory (QT). The model trains two message-passing procedures. The first procedure focused on learning features from larger networks, while the second focused on learning features from smaller networks. The output is the average of both procedures. SOFGNN team [25] averaged two models and used data augmentation and feature engineering to solve the out-of-distribution problem. To obtain a higher amount of samples for the training dataset, data augmentation is applied, creating new samples similar to the ones found in the training sets. Regarding feature engineering, they created a new feature called *link load* (%), defined as the sum of the traffic of all flows traversing the link and divided by link capacity. The average result of two trained models was used, one using as features the square value of *link load*, and the other model including both the square and cube values of *link load*. Finally, the hyperparameters were fine-tuned. Barcelona Neural Networking Center - Universitat Politècnica de Catalunya (BNN-UPC) also proposed a solution to improve RouteNet's out-of-distribution generalization [35]. The proposed model implements an extra message-passing procedure and introduced a scaling factor combined with the data augmentation to estimate the delay values of higher-capacity links. Unfortunately, this model required a long time to train, in part due to applying only feature selection and not applying feature engineering to improve the model generalization.

Table 1 provides a summary of the mentioned recent methods for using ML, DL, and GNNs for network modeling. Many works have tried to address the network performance prediction problem. Among the most used solutions, GNNs are the ML models that are best suited to the graph representation seen in networks. Nevertheless, GNNs suffer an out-of-distribution generalization problem and some works have tried to resolve it.

Table 1: Comparison of Girona Antwerp Intelligence for Networks (GAIN) and other contributions focusing on GNN.

Contribution	Deep learning model	Architecture	Comparison against	Source code availability	Hyperparameter tune	Fast train	Feature engineering	Multiple models
[29, 56]	GNN	Custom with MLPs, GCGR and GAT	RouteNet	Public	Yes	No	Yes	Yes
[25]	GNN	Modified RouteNet	RouteNet	Public	Yes	Yes	Yes	Yes
[17]	GNN	RNN + GRU	OMNeT++ simulation	Public	No	No	No	No
[13]	CNN	3D-CNN + MLPs	Real scenario	Private	N/A	N/A	Yes	No
[12]	CNN	4D-CNN + MLPs	Real scenario	Private	N/A	N/A	Yes	No
[37]	RNN	SLSTM	Real data	On request	N/A	N/A	Yes	No
[5]	RNN	NARX	AnyLogic Simulator	Private	N/A	Yes	Yes	No
[53]	Classification models	SVM, DT, LR	Real data	Private	N/A	Yes	Yes	No
[36]	GNN	GCN + MLPs	City simulations	Public	N/A	N/A	Yes	No
[90]	GNN	GraphSAGE	OMNeT++ simulation	Private	No	No	Yes	No
[76]	GNN	GAT + MLPs + genetic algorithm	Genetic Algorithm + greedy solution	Private	Yes	N/A	Yes	Yes
[38]	GNN	STGCN	OMNeT++ + simulation	Private	Yes	No	Yes	No
GAIN	**GNN**	**Modified RouteNet**	**RouteNet**	**Public**	**Yes**	**Yes**	**Yes**	**No**

3.4 Dataset

The dataset for training and testing the GNN model was provided by BNN-UPC [16], and was generated with OMNeT++, a packet-level network simulator. The model receives the following inputs per each sample in the dataset:

1. Topology: graph including node and link-level properties (e.g., nodes, links, queue sizes, link capacity).

2. Flow performance: flow level and aggregate source-destination measurements (e.g., dropped packets, average delay).

3. Flow traffic: flow level and aggregate source-destination time and size distributions used to generate traffic (e.g., average bandwidth, packets generated, average packet size).

4. Routing: paths connecting source-destination pairs.

5. Link performance: metrics of output ports (e.g., utilization and losses, average port occupancy).

The dataset also includes a preset split into training, validation, and testing as follows:

- Training: small networks between 25 and 50 nodes.

- Validation: larger networks from 50 to 300 nodes to analyze the ability of the models to generalize.

- Test: equivalent characteristics to the validation dataset.

Moreover, the testing dataset is subdivided into three subsets called settings, having different features each:

1. Setting 1 (S_1). Longer paths. Focused on the artificially generated longer paths regarding the training dataset. However, the link capacities have the same value ranges as in the training dataset. Some source-destination pairs do not transmit traffic.
2. Setting 2 (S_2). Increased link capacity. Focus on the larger link capacity with respect to the training dataset. All the source-destination pairs transmit traffic using shortest-path routing. Moreover, there is higher aggregated traffic and higher capacity links than in the training dataset.
3. Setting 3 (S_3). Both S_1 and S_2 properties are mixed. All source-destination pairs transmit traffic using longer paths with higher capacity links than the training dataset.

Dataset class balancing is relevant to train, validate and test the model with all the cases that it might have to predict [20]. In addition, the number of samples predicted later has also to be balanced, to check the accuracy stability in different model use cases. In the following subsections, the training, validation, and test datasets are analyzed, in order to check the feature distributions and the dataset balancing.

3.4.1 Training dataset analysis

The training dataset consists of approximately 173 million path samples. The graph size represents the number of nodes for each network. As evidenced in Fig. 7, each graph size is directly related to the number of paths (i.e., the number of samples to train the model). The paths on bigger graphs are proportionally longer. The paths are mainly of lengths 3 and 4, as illustrated in Fig. 8.

The traffic rate in bits per second are balanced through the entire dataset for each graph size and path length, as displayed in Fig. 9 and Fig. 10. Packets per second are correlated with traffic in bits per second, as they depend directly on constant packet size. Other path features values, including packets dropped and the predictor variable, delay, are more dispersed in terms of value distributions. The selected link features of utilization, maximum queue occupancy, and offered traffic intensity (O_t), a variable created in Section 3.5.4, are also correctly balanced in the dataset, as shown in Fig. 11.

3.4.2 Validation dataset analysis

The validation dataset contains approximately 52 million path samples and follows the structure of the training dataset. The main difference is the increased number of graph nodes, ranging from 50 to 300 nodes, to test the generalization of the trained model. As in the training dataset and represented in Fig. 12, the paths on bigger graphs are proportionally longer. The paths mainly have a length between 3 and 10, as shown in Fig. 13.

Also similarly to the training dataset, the values of traffic in bits per second are also balanced through all the dataset for each graph size and path length as detailed in Fig. 14 and Fig. 15. In the case of longer paths, a larger deviation in the values can be observed. In terms of the analyzed link features, utilization, maximum queue occupancy, and O_t, the validation dataset is also correctly balanced, as shown in Fig. 16.

3.4.3 Testing dataset analysis

The testing dataset consists of approximately 26 million path samples and follows the structure of the train and validation datasets. As in validation, it includes the increased number of graph nodes, from 50 to 300 nodes. The main difference, in this case, is the partition in the three different subsets, S_1, S_2, and S_3. For S_2 and S_3, the paths on the bigger graphs are proportionally longer, as shown in Fig. 19 and Fig. 20. On the other hand, the quantity of paths in S_1 is not regular as depicted in Fig. 18.

The paths mainly have a length between 9 and 11 in S_1, between 3 and 6 in S_2 and between 6 and 10 in S_3, as shown in Fig. 17. However, S_1 has significantly fewer path samples than the other settings.

None of the datasets were modified, as the testing and validation datasets were already balanced. The subsets S_2 and S_3 of the testing dataset were also found to be balanced. On the other hand, the S_1 test dataset had very few samples and with higher deviation. These features were related to the link properties, including link utilization, maximum queue occupancy, and O_t, as shown in Fig. 21. These features were out of the training and validation distributions, mainly in the graph samples with a higher number of nodes. The cause of this deviation was that not all the source-destination paths transmitted traffic, only the particularly long ones. These characteristics lead to slightly higher errors in the S_1 predictions, as described in Section 3.6, compared to S_2 and S_3. However, these errors were specific to this small subset designed for particular testing purposes (i.e., longer path testing with lower traffic).

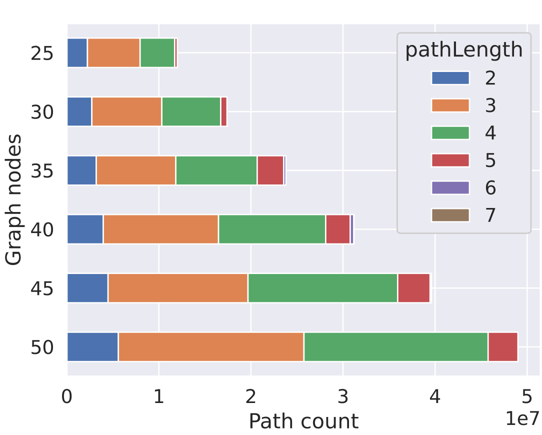

Figure 7: Number of samples per graph size, and proportion of path lengths per graph size in training dataset.

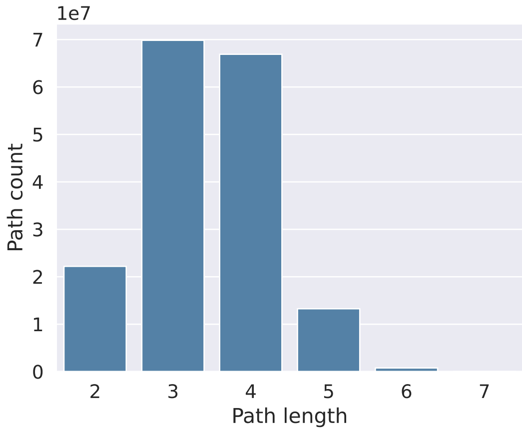

Figure 8: Number of samples of per-path length in training dataset.

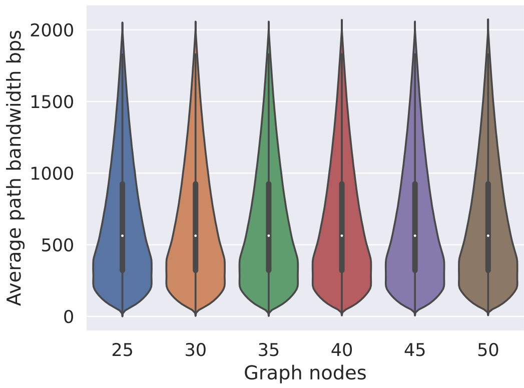

Figure 9: Average path bandwidth distribution per graph size in training dataset.

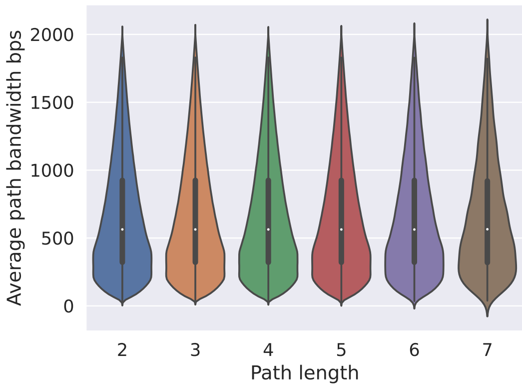

Figure 10: Average path bandwidth per path-length in training dataset.

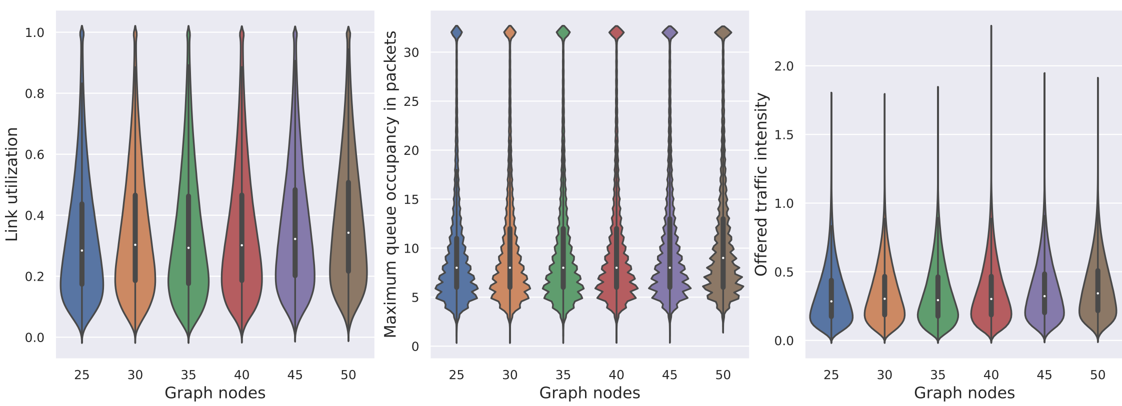

Figure 11: Link utilization, maximum queue occupancy and O_t values distributions compared to graph size in training dataset.

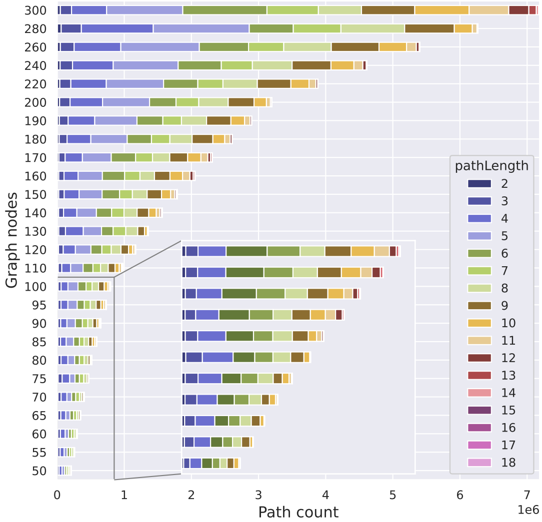

Figure 12: Number of samples per graph size, and proportion of path lengths per graph size in validation dataset.

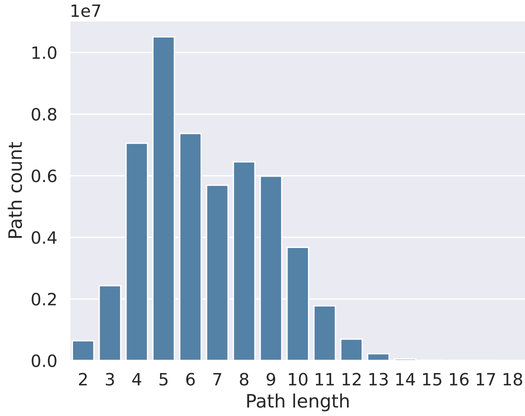

Figure 13: Number of samples per-path length in validation dataset.

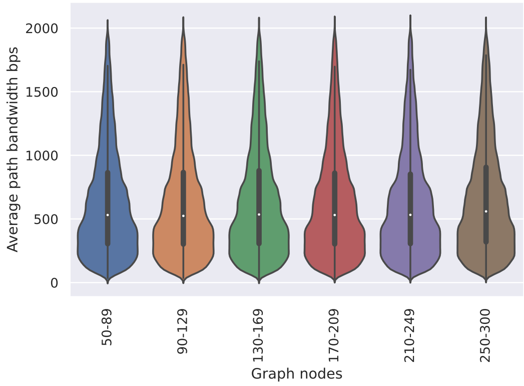

Figure 14: Average path bandwidth distribution per graph size in validation dataset.

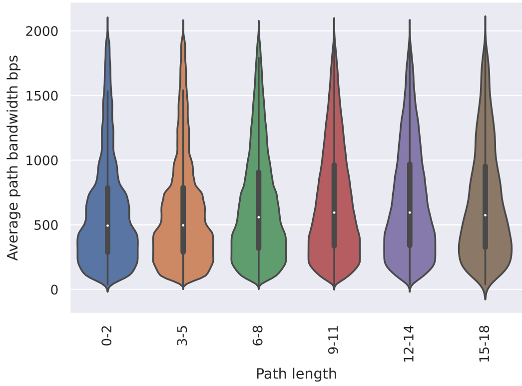

Figure 15: Average path bandwidth distribution per path-length in validation dataset.

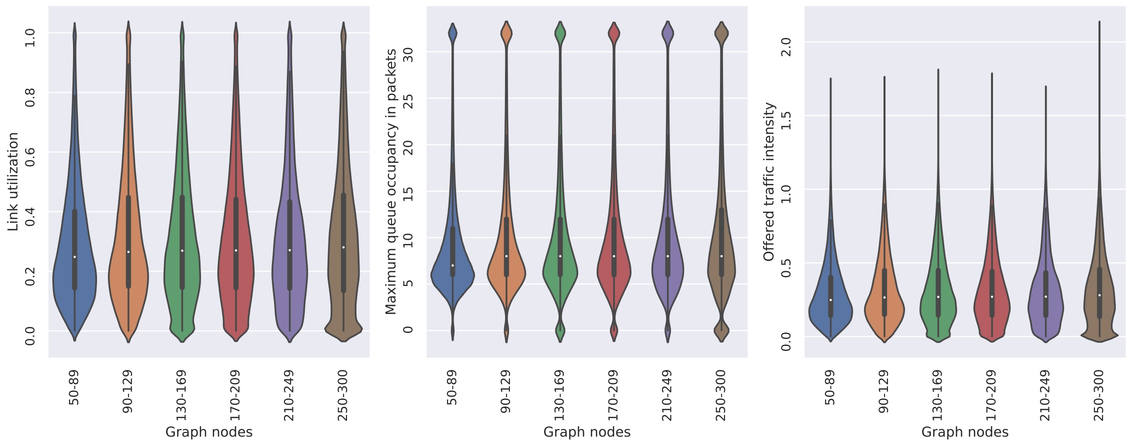

Figure 16: Utilization, maximum queue occupancy and O_t values distributions compared to graph size in validation dataset.

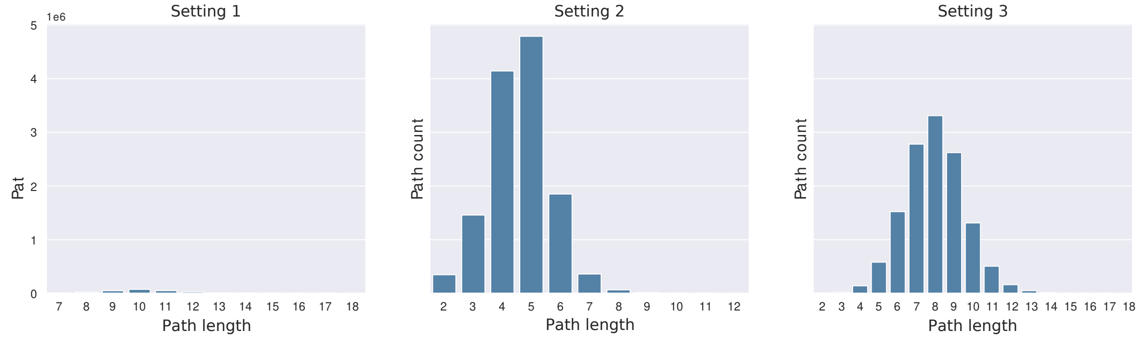

Figure 17: Per-path length, number of samples in test dataset.

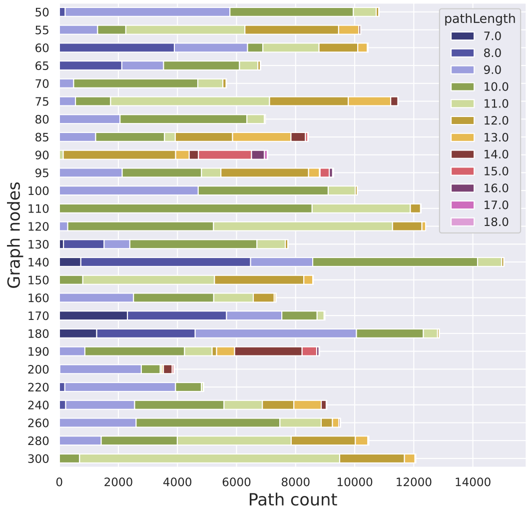

Figure 18: Number of samples per graph size, and proportion of path lengths per graph size in test dataset, S_1.

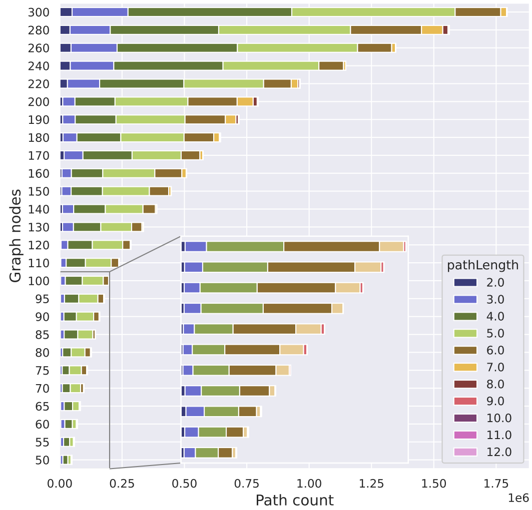

Figure 19: Number of samples per graph size, and proportion of path lengths per graph size in test dataset, S_2.

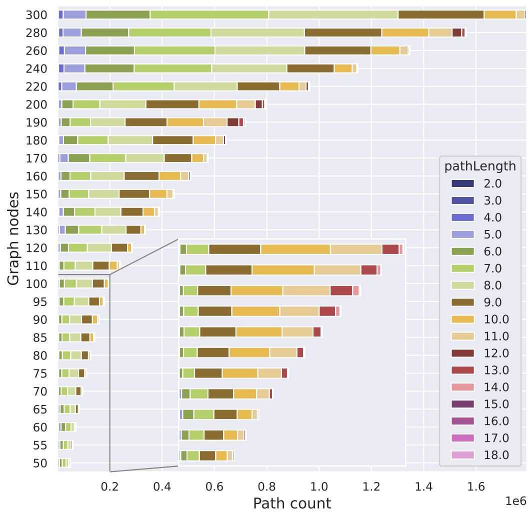

Figure 20: Number of samples per graph size, and proportion of path lengths per graph size in test dataset, S_3.

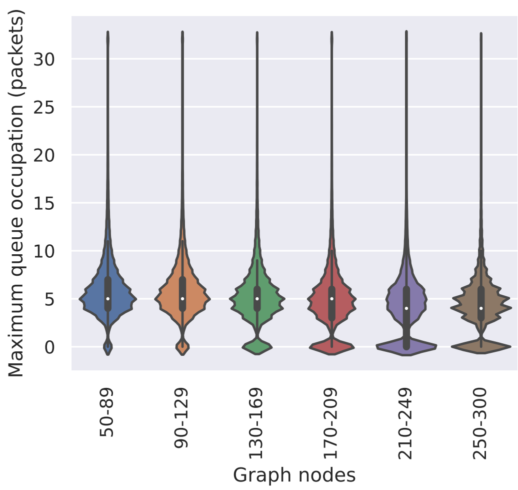

Figure 21: Maximum queue occupancy distribution per each graph size in S_1.

3.5 Improving RouteNet for generalization

In this section, the improvements to the RouteNet baseline implemented in TensorFlow [17] are described step by step, including the improvement achieved. In this work follows step-by-step testing of individual modifications and measuring the enhancement of each applied change, one after the other.

3.5.1 Inferring per-path delay from predicted queue occupancy (GAIN-1 solution)

Since the validation and test datasets contain topologies with higher link capacities, the delay values in larger topologies are lower than in smaller topologies. This implies that the model is trained with data following a particular data distribution but has to predict values that follow another distribution, which is problematic for NNs [47].

The initial idea to solve this problem consisted in finding an indirect metric that keeps a similar distribution among all the datasets and can be used, followed by a post-processing step, to predict the metric of path delay with low error. Based on this idea, the first approach was to use the occupancy of a link (Q_o) as an indirect metric, representing the average utilization of a queue, as it encapsulates local relationships between offered traffic, queue size, and link capacity. The first step, titled GAIN-1, was to test the improvement of the original RouteNet implementation, replacing the initial prediction of delay with the prediction of the indirect metric Q_o and inferring the path delay from that prediction.

After predicting Q_o, each flow's delay is estimated by adding the queue delays belonging to a path in a post-processing step. A path with three nodes is assumed, as depicted in Fig. 22. When the packets are sent from the *src* node, they are queued in Q_1 accumulating a delay. Depending on the queuing policy (assumed to be the same in each queue), the queue occupancy, and the link capacity, the packets will suffer more or less delay. If a link has more capacity, the waiting time of a packet in the queue will be lower. Once the packets are sent to the second node using $Link_1$, they are queued in Q_2, adding another delay. Finally, the packets are sent through $Link_2$ to the *dst* node. Therefore, the delay of a source-destination flow can be obtained as the sum of the delays of every link of the flow, starting from $K = 1$ and ending with the total number of links of the flow N_f , as shown in Eq. (2).

Each queue delay of this flow was calculated using Eq. (3), where Q_o is the occupancy of the queue, Q_s is the queue size in number of packets, A_p is the average packet size in number of bits, and C_l is the capacity of the outgoing link of the queue. The link delay and flow delay formulations are presented below.

$$Delay_{flow} = \sum_{k=1}^{N_f} Delay_{link} \tag{2}$$

$$Delay_{link} = Q_o \times Q_s \times A_p / C_l \tag{3}$$

The post-processing code was optimized using the appropriate TensorFlow data structures to increase the processing speed and reduce code complexity. The resulting MAPE of the GAIN-1 solution was 44.73 %.

3.5.2 Normalization of predictor features (GAIN-2 solution)

After analyzing the training and validation datasets, it was observed that the data distributions of the scalar features used in the model (detailed in Table 3) were different for training, compared to the validation and test subsets, as shown in Table 2. Specifically, the traffic of a flow (T_f) and the capacity of a link (C_l) had different value ranges for each dataset and between the training and validation/test datasets, respectively.

Therefore, in GAIN-2, a min-max normalization was applied as a pre-processing in the *transformation* function, using the training dataset min-max values. The max-min normalization was calculated using the formula in Eq. (4). Normalized value N_v equals the original value V subtracting the minimum value of the feature for all the training dataset V_{min}, and dividing the result by the difference between the maximum V_{max} and V_{min} of the training dataset. The TensorFlow functions were used to find the maximum and minimum values on the training dataset's tensor, which efficiently reduce the tensor to one dimension to return the desired value.

Table 2: GAIN used features maximum and minimum values.

	Training		**Validation**		**Test**	
	Min	**Max**	**Min**	**Max**	**Min**	**Max**
Traffic	30.787	2048.23	30.543	2064.93	0	2061.17
Packets	0.0329	2.03633	0.03107	2.03985	0	2.0543
Capacity	10000	100000	10000	2500000	10000	2500000

$$N_v = \frac{V - V_{min}}{V_{max} - V_{min}} \tag{4}$$

The result of the GAIN-2 solution was a 28.739 % MAPE.

3.5.3 Feature selection (GAIN-3 solution)

The features included in the baseline were T_f and C_l, as described in Table 3. After a correlation test between the features available and the expected delay results in the datasets, packets (P_f) revealed a high correlation to path delay. Furthermore, this feature had value ranges very similar in the three datasets, enhancing generalization. Consequently, it was added to the GAIN-3 path state, also including the min-max normalization preprocessing. The MAPE of the GAIN-3 solution was 18.471 %.

Table 3: Features used in the GAIN solutions.

Feature	Definition	Used in
Path features		
Traffic (T_f)	Traffic in a path (bits/time unit)	All GAIN and baseline
Packets (P_f)	Packets generated in a path (packets/time unit)	GAIN 3,4,5
Link features		
Capacity (C_l)	Link bandwidth (bits/time unit)	GAIN 1,2,3 and baseline
Offered Traffic Intensity (O_t)	Sum of T_f in a link divided by C_l	GAIN 4,5

3.5.4 Offered Traffic Intensity (GAIN-4 solution)

When approaching the total link capacity, increasing the total T_f in a link directly impacts the queue delay by increasing it too. Fig. 23 shows an example where a source *src* sends traffic to *dst*. If the sum of T_f of each flow in the link is close to or exceeds C_l, the Q_1 delay will increase.

The dependence between T_f demand in a link and the queuing delay [62], leads to the creation of a new feature named *offered traffic intensity* (O_t), as shown in Eq. (5). It was defined as the sum of T_f of all flows f passing through the network link N_l divided by the bandwidth C_l of that link, resulting in a scalar feature assigned to the link state.

$$O_t = \frac{\sum_{f \in N_\ell} T_f}{C_\ell} \tag{5}$$

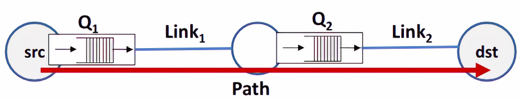

Figure 22: Queue occupancy example to infer delay.

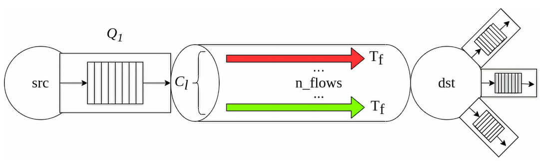

Figure 23: Offered traffic intensity example of flows in a link.

Finally, only the feature O_t was used in GAIN-4. For that reason, C_l did not require normalization, as it was removed from the model, and the original values were used to calculate O_t. The resulting MAPE of the GAIN-4 solution was 2.612 %.

3.5.5 Hyperparameters optimization (GAIN-5 solution)

The hyperparameters, i.e., number of neurons for the path-state and link-state, readout units, message-passing iterations, and epochs, were optimized for the model, testing different combined values using a grid search, defining and combining values for each parameter, as detailed in Table 4. For each combination of hyperparameters, a new training was performed followed by a check of the MAPE result.

One of the effects of optimizing the hyperparameters is the reduction of the training epochs, as the model stabilizes the validation results much earlier than the original 100 epochs. For reference, the best model achieved the best result after seven epochs. An *early stopping* epoch limiter is a technique that stops the training after a number of epochs where the model does not improve its results. The number of epochs monitored to do the early stopping is called *patience*. It was used in the GAIN-4 and GAIN-5 solutions, with a patience of 5 epochs, to speed up the hyperparameter tests.

After hyperparameter tuning, the GAIN-5 model was trained during seven epochs. It was found that reducing the dimensions of the link state and the path state and increasing the readout units along the iterations of the message passing procedure significantly improved the results from 2.612 % to 1.838 %, the best result of all the GAIN solutions, also improving for the different subsets.

Another interesting effect of keeping hyperparameters and epochs at a low value is the reduced Graphics Processing Unit (GPU) power consumption, temperature, compute, and memory utilization as exposed in Section 3.6.

Table 4: Values tested for optimization of the hyper-parameters.

Hyperparameter	**Values tested**	**Original RouteNet values**	**Optimized GAIN 5 values**
Link state dimension	4, 8, 16, 32, 64	16	4
Path state dimension	4, 8, 16, 32, 64	32	16
Readout units	4, 8, 16, 32, 64	8	64
Message passing iterations	6, 8, 10, 12, 14, 16	8	10
Epochs	100, variable	100	7

3.6 Results

This section reports the results obtained following the steps described in the previous section. The GAIN-2 solution was the one presented to the challenge, achieving a 28.739% MAPE. GAIN-3, 4, and 5 are the results of additional improvements after the competition. In addition, a dataset with real traces of traffic is included for further evaluation.

3.6.1 Additional models for evaluation

For further analysis and comparison, two models are added to the predictions phase. Apart from the RouteNet baseline and the GAIN-5, two other delay prediction models are used: RouteNet-Erlang [32] and Queuinx [77].

RouteNet-Erlang also uses RouteNet as a baseline, applying improvements to increase the prediction accuracy in larger networks (≈10x), longer paths, and larger link capacities, using queue status to improve the prediction accuracy. The model is trained with the same dataset used with the baseline and GAIN-5.

Queuinx is a model that uses queuing theory, being focused on finite queues, using a fixed point algorithm. This model is used to validate the RouteNet [78] results in comparison to queuing theory computations. Both models are used to predict the test dataset and are included in the analysis of the results.

3.6.2 Models evaluation with test dataset

The gradual improvement in each step is shown in Table 5, where the RouteNet baseline is shown to have a bad generalization for larger graphs (as an example, S_1, S_2, S_3 with 300 nodes) resulting in a MAPE of 187.28% for the full testing dataset. This is mainly caused by the out-of-distribution values when predicting in the validation and testing dataset, especially in S_2 and S_3. The MAPE was calculated separately for each column and row in Table 5. The GAIN solutions gradually reduced MAPE, applying improvements step by step as described in Section 3.5, using the three settings with the graph sizes 50 and 300 as a reference. From the RouteNet Baseline and GAIN-1 solution, which was the first change over the baseline, until the GAIN-5 solution, achieving an improvement by a factor of 24 (from 44.73 % to 1.838%) and by a factor of 101 (from 187.28% to 1.828%) lower MAPE in all settings, respectively.

The improvement between the baseline and GAIN-5 can be compared in Fig. 24, where the subsets S_1, S_2 and S_3 are represented. The absolute error is represented in a logarithmic scale for comparison reasons. For GAIN-5, the error is 32x smaller than in the baseline. GAIN-5 is compared to the baseline in terms of graph size and path length, two features of a graph that scale up when the sample is bigger, as shown in Section 3.4. It can be observed that in setting S_2 and S_3, the tendency of the prediction error is to become lower when the graph increases the number of nodes, improving the generalization compared to the baseline.

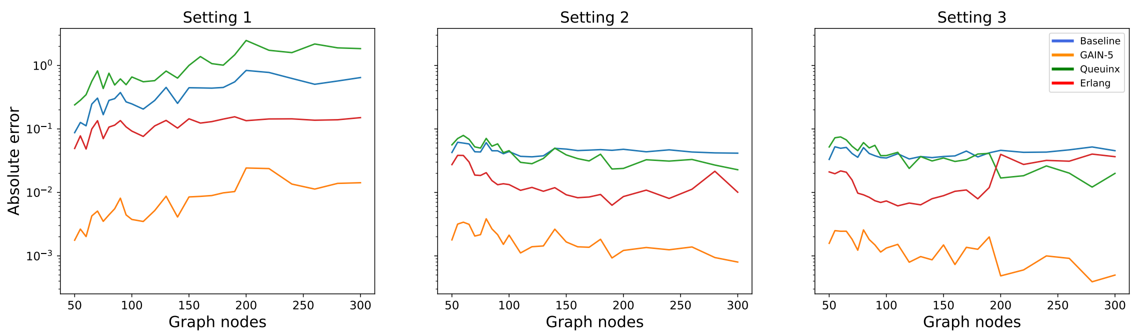

Figure 24: Baseline, Queuinx, RouteNet-Erlang and GAIN-5 logarithmic absolute errors compared to graph size.

Table 5: Test dataset MAPE (%) results and performance for baseline and each GAIN solution.

	Full testing dataset	**S_1**	**S_1 50 nodes**	**S_1 300 nodes**	**S_2**	**S_2 50 nodes**	**S_2 300 nodes**	**S_3**	**S_3 50 nodes**	**S_3 300 nodes**	**Train time**	**Avg. train GPU**
Baseline	187.28	79.145	68.481	92.979	253.075	68.481	345.135	247.217	44.669	368.019	12h 15m	∼19%
GAIN 1	44.73	13.074	11.705	13.108	54.318	16.214	42.374	67.44	70.994	90.739	9h 45m	∼21%
GAIN 2	28.739	11.719	11.026	11.864	35.067	17.092	39.773	31.754	17.581	31.353	9h 48m	∼19%
GAIN 3	18.471	9.436	6.893	12.468	26.897	22.106	30.067	18.143	12.569	21.862	9h 40m	∼15%
GAIN 4	2.612	2.652	1.539	3.687	2.492	1.386	2.567	2.584	1.607	2.363	3h 25m	∼4%
GAIN 5	**1.838**	**1.407**	**1.111**	**1.808**	**1.929**	**1.573**	**1.535**	**1.756**	**1.388**	**1.462**	**2h 20m**	**∼15%**

As introduced in the test analysis in Section 3.4.3, the S_1 subset had a lower number of samples including features with high deviation compared to S_2 and S_3. These characteristics lead to increased errors in the S_1 predictions, both in baseline and GAIN-5 implementations, but still much lower in GAIN-5. The small datasets make it hard to rigorously evaluate GNNs, as they are data-hungry models [94], requiring significant amounts of data to check the correctness of the prediction accuracy. This is the main problem in analyzing the results of S_1, where the high deviation paired with a low amount of samples resulted in more variability in the accuracy of the predictions.

On the other hand, the predictions on S_2 with higher capacity links and on S_3, with long paths mixed with higher capacity links, resulted in a low overall error for all the graph sizes, becoming lower as the graph was bigger. As shown on the error scales, the absolute error variability range on the GAIN-5 solution on all settings is much smaller than in the baseline results.

Also in Fig. 24 the prediction results for the Queuinx and RouteNet-Erlang models are shown. Queuinx reveals similar results compared to the baseline. It can be concluded that both models exhibit a higher error than GAIN-5, with RouteNet-Erlang being better than the challenge baseline in every setting. Comparing the errors linked to the path lengths in Fig. 25, it can be observed that with longer paths, the error deviation is larger. This is due to the low amount of samples and their particularity in the S_1 of the dataset. In Fig. 26 this problem is shown to have a small impact on the solution precision. On the left, all settings are mixed to obtain the average absolute error per each graph size. GAIN-5 is shown to have a lower error on all settings mixed in comparison to the other tested models, lowering the error as the graph size increased. On the right, all settings compared to the path length are checked, where a higher error with the out-of-distribution samples is observed, as a small number of samples were above 15 hops.

In a further error analysis to confirm the generalization of the model, heat maps for the three settings were generated. In Fig. 27, S_1 was found to scale slightly worse than the other settings as detected in the previous analysis, especially with the graphs of 200 and 220 nodes. Inversely, Fig. 28 and Fig. 29 show the generalization of the model, keeping the errors low and even lowering the absolute errors as the networks become bigger.

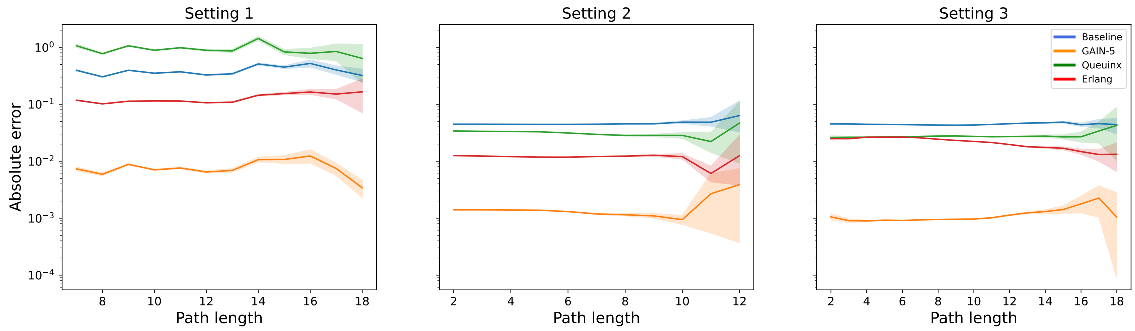

Figure 25: Baseline, Queuinx, RouteNet-Erlang and GAIN-5 logarithmic absolute error compared to path length.

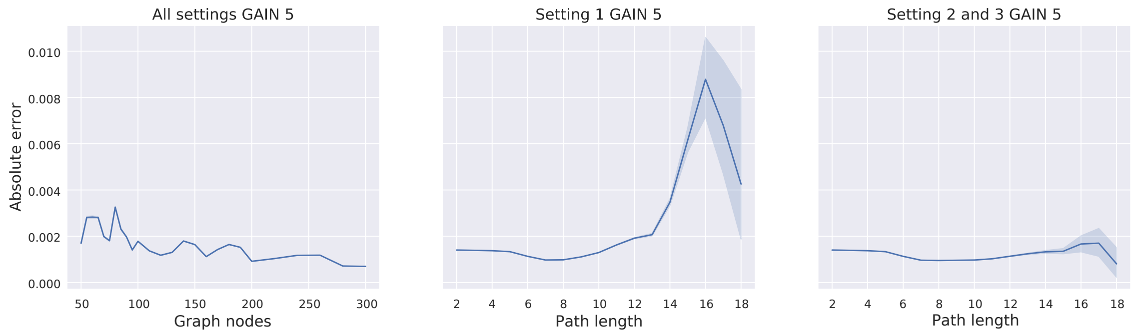

Figure 26: Baseline, Queuinx, RouteNet-Erlang and GAIN-5 all settings, logarithmic absolute errors compared to path length.

3.6.3 Models evaluation with real traffic traces dataset

To further validate the accuracy and generalization of the models, a real traffic dataset was used. This real traces dataset [18] was generated using real-world traffic matrices and realistic packet inter-arrival times. The dataset contains network topologies from actual production networks, which were collected through traffic monitoring tools deployed on operational internet networks. These networks do not explicitly involve network slicing, but they represent a broad range of real-world traffic patterns and topologies that are highly applicable to general network management and KPI prediction.

The topologies in this dataset include the ABILENE, GEANT, and GBN networks, which are large-scale internet networks, used in both academic and research communities for network performance studies [31]. These networks have varying characteristics such as traffic loads, link capacities, and topologies, providing a diverse testing ground for model evaluation. ABILENE, for instance, is a high-speed internet network, GEANT is a pan-European research and education network, and GBN is a global backbone network used for testing, each with 11, 22 and 17 nodes respectively.

These networks do not directly support network slicing, but the dataset is used to evaluate how well GAIN-5 can generalize to different network environments. The traffic matrices in these datasets are derived from real-world internet usage patterns, which include a mix of high- and low-priority traffic, analogous to what could be found in network slices

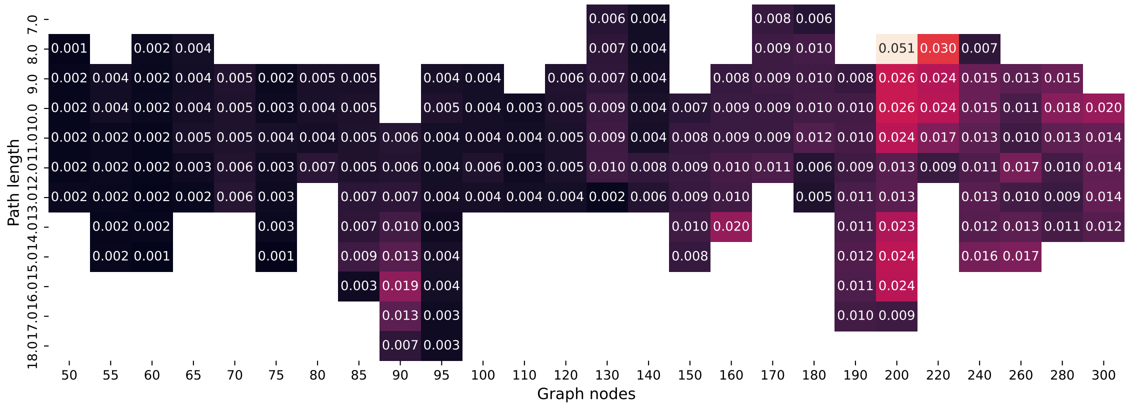

Figure 27: GAIN-5 absolute error on S_1 compared to path length.

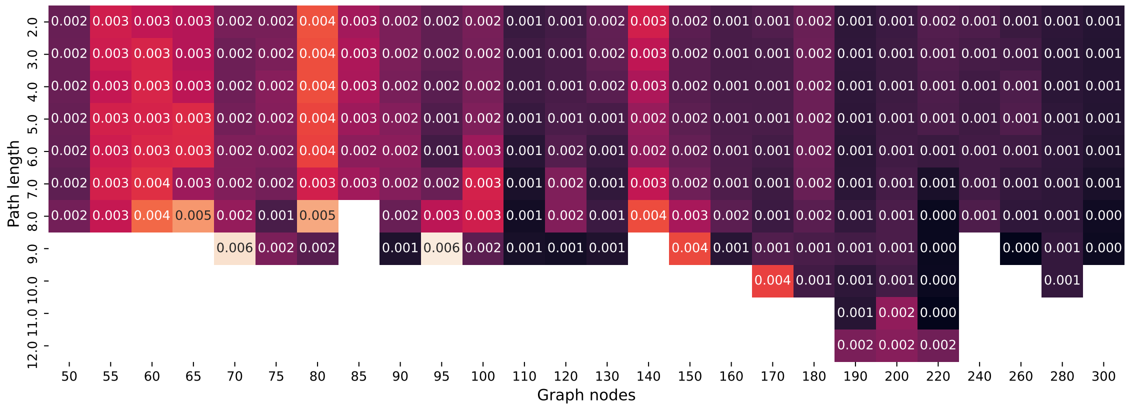

Figure 28: GAIN-5 absolute error on S_2 compared to path length.

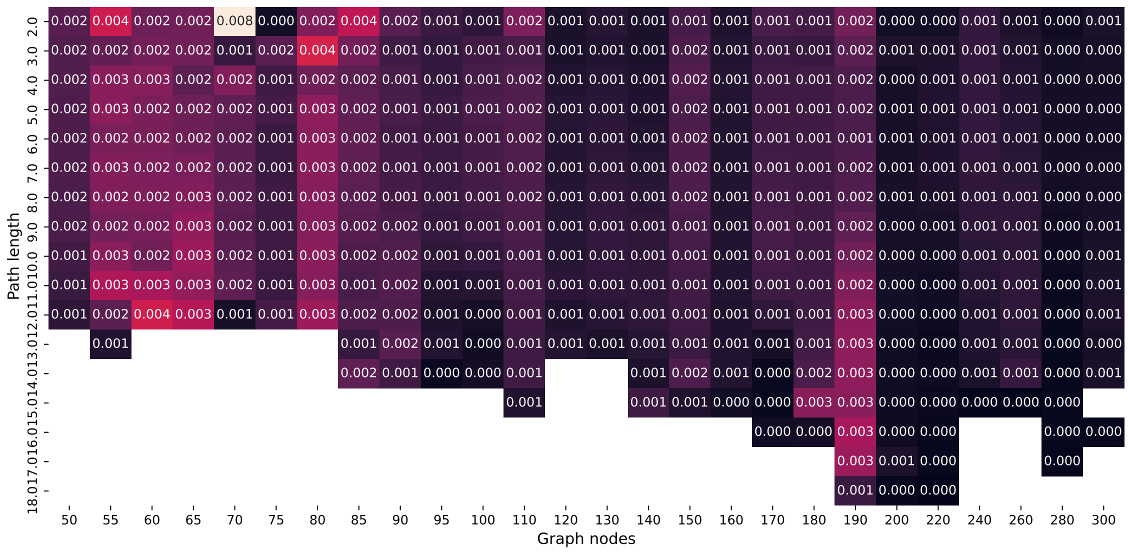

Figure 29: GAIN-5 absolute error on S_3 compared to path length.

with varying QoS requirements. As such, while the datasets do not explicitly include network slicing, they represent traffic scenarios relevant to 5G and B5G networks that require accurate KPI prediction.

In this context, the GAIN-5 model is evaluated for its ability to predict KPIs such as delay and throughput under real-world traffic conditions. This evaluation provides insights into the model's performance in non-simulated, realistic environments.

In Fig. 30 the MAPE results for each graph and model are shown. For ABILENE and GEANT networks, the results are similar, with RouteNet-Erlang achieving the best results, followed closely by GAIN-5. Meanwhile, testing the models with the GBN network, which contains smaller links and lower traffic, the results are clearly better using GAIN-5, which demonstrates a better generalization maintaining a lower overall error on the three scenarios.

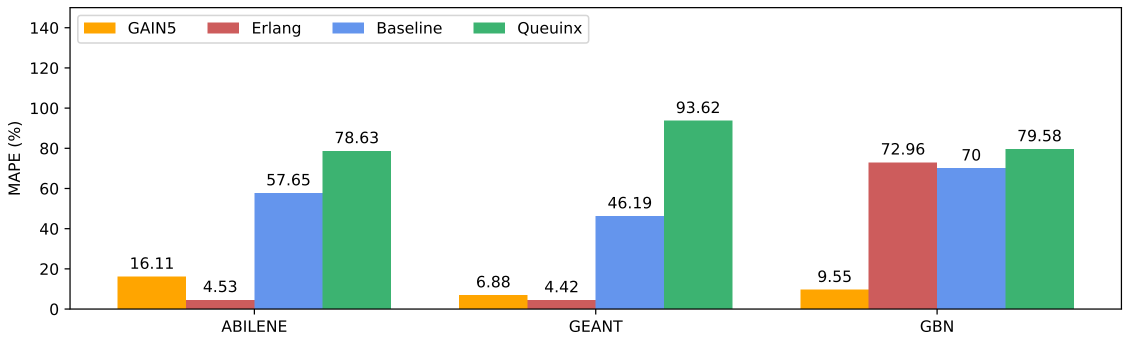

Figure 30: Baseline, GAIN-5, Queuinx and RouteNet-Erlang MAPE results with a real traces dataset.

3.6.4 Resource utilization

Concerning the GPU usage, it can be observed that in the GAIN-4 solution is $\sim$4% while in the baseline it is $\sim$19%, with the added benefit of a much faster training. One of the causes of this reduction is the better convergence of the model, as illustrated in Fig. 31. In GAIN-5 the GPU consumption is higher than in GAIN-4, because of the higher hyperparameters, but GPU consumption is still kept under 15%. The hardware of the testbed used for the training was composed of two GPUs Nvidia GeForce GTX 1070 with 8 GB GDDR5 memory, paired with an AMD Ryzen 5 5600X CPU, 32 GB of DDR4 RAM, and a 500 GB M.2 SSD. The time to train and test the GAIN-5 model was about 3 hours 30 minutes, 2 hours 20 minutes for training, which is a significant reduction compared to the original 12 hours training of the baseline.

3.6.4.1 Training time of GAIN-5 in real 5G networks

The training time of the GAIN-5 model is an important factor when considering its deployment in real 5G networks, particularly in environments with network slicing where

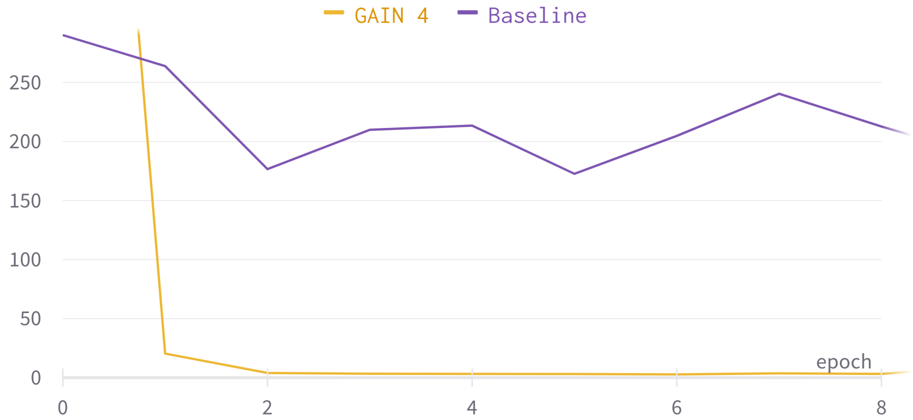

Figure 31: Validation MAPE of the RouteNet baseline vs the GAIN-4 solution.

dynamic KPI prediction is crucial for maintaining SLAs. In this work, the GAIN-5 model achieved a significant reduction in training time compared to the baseline RouteNet model. While the baseline model required 12 hours of training, GAIN-5 reduced this to approximately 3 hours 30 minutes on a system equipped with two Nvidia GTX 1070 GPUs.

In the context of 5G networks, the location and timing of training are key considerations. For instance, if retraining is required frequently due to changing network conditions or new configurations, the training could take place at the network edge or in centralized cloud environments depending on the available computational resources. The model's relatively low GPU requirements make it feasible for training in edge computing environments, where smaller-scale GPUs can be deployed. This is particularly relevant for edge data centers that support MEC, allowing the model to be retrained closer to the network's edge where latency-sensitive applications are managed.

For more centralized 5G network deployments, GAIN-5 could be trained in a core network or cloud-based infrastructure where more powerful computational resources are available. The lower training time reduces the operational impact on the network and allows for frequent updates to the model, ensuring that it remains adaptive to dynamic network conditions without introducing delays or requiring significant network resources for retraining. This flexibility in training location makes GAIN-5 a scalable and practical solution for real-time network slicing management in 5G networks.

Finally, to compare GAIN-5 with other solutions, Table 6 provides a summary of the best different approaches and MAPE results. The uniqueness of the solution presented in this work (GAIN-5) [40] is the preservation of the architecture and data of the original RouteNet model, as the message passing procedure was not modified and data augmentation was not used, resulting in an implementation with lower overhead than in the other solutions. Additionally, this work approach did not use the average of multiple models to maintain the simplicity of the baseline. The use of the new feature *offered traffic intensity*, designed from the knowledge of traditional queuing theory, significantly improved the accuracy of the predictions. GAIN-5 demonstrates that it is possible to obtain a low MAPE utilizing a simpler approach, just improving the original RouteNet model, and avoiding the usage of multiple tuned models. This simplicity allows fast training and easier management of the model, enhancing its usage in new environments by just retraining and deploying it. No other solution in the BNN-UPC challenge has fast training and good precision without using multiple models.The fine-tuning of GNN hyperparameters set smaller in the GAIN solution, allowed lower training and prediction times, and less memory and power usage, thus making the GAIN solution more suitable in case of frequent retraining requirements.

Table 6: Comparison of the best solutions for the BNN-UPC challenge.

Solution properties	**GAIN 5**	**PARANA**	**SOFGNN**
RouteNet baseline	X		X
Hyperparameter tune	X	X	X
Fast train/execution	X		X
Feature engineering	X	X	X
Reimplemented baseline		X	
Message passing redesign		X	
Multiple models		X	X
Data augmentation			X
MAPE (%)	**1.838**	**1.267**	**1.389**

3.6.5 Predicting different types of KPIs in network slices

While the main focus of GAIN-5 in this chapter has been on predicting network delay, the model could be also adapted to be versatile in predicting a variety of KPIs relevant to 5G and network slicing environments. These KPIs include delay, packet loss, and jitter, all of which are critical for maintaining the QoS requirements of different network slices.

In a network slicing context, GAIN-5 can be used to handle diverse traffic types. For example:

- eMBB: For slices that require high throughput, GAIN-5 can be trained to predict delay, jitter and loss, ensuring that the slice meets the necessary requirements.
- URLLC: In these slices, delay and packet loss are critical KPIs. GAIN-5 can be fine-tuned to prioritize these metrics, ensuring real-time applications meet stringent latency requirements.
- mIoT: For IoT applications, packet loss is more important. GAIN-5 can be adapted to predict and optimize this KPIs in IoT-focused slices.

GAIN-5's ability to generalize across different KPIs makes it a basic solution for managing specific network slices with varying configurations and performance demands. A more advanced model for multiple KPI prediction is proposed in Section 5, where specific details about network slices are also provided.

3.7 Conclusions

The network models based on GNNs are being suggested as a good approach for building autonomous networks, as they can both properly forecast network KPIs and easily be modified and updated to account for the current networks rising complexity and dynamism. Out-of-the-box GNNs are unable to generalize and scale to larger topologies, which causes their accuracy to decline. In this work, GAIN is introduced, a GNN-based model that accurately forecasts the average per-path network delay. The good generalization of the model, tested with heterogeneous datasets, enables it for predicting per-path delay in many scenarios, thereby proving it useful for managing B5G services.

In order to achieve good generalization, GAIN is built on RouteNet's GNN and makes a number of additions. These additions include the inference of per-path delay from the predicted link queue occupancy, feature normalization, feature selection, feature engineering (offered traffic intensity), and hyperparameter optimization. Additionally, compared to other solutions currently available, GAIN has less implementation complexity, lower resource requirements, and faster training. As shown in this chapter, the main differences of the GAIN model are the fast training, high accuracy, the usage of a simple and single model, and the public availability of the code and the datasets.

A closer analysis, filtering, and preparation of the datasets is key for achieving improved results in a ML approach. In this case, the training and validation datasets were well generated for the training task. The testing dataset had particularities discovered during the test phase, where a closer analysis helped to know the cause of the slightly higher errors in a specific case. In addition, this challenge allowed us to learn data analytics and helped us learning GNNs programming.

In the upcoming sections, this work will leverage the knowledge gained from working with GNNs to explore various GNN architectures, frameworks, and methods for enhancing performance. The following steps, outlined in Section 4, involve the creation of new data samples, which is a pivotal precursor to the development of a novel GNN model detailed in Section 5. This new model is specifically designed to predict KPIs related to network slicing in B5G networks.

4 Generating network slicing simulations

After learning how GNNs work, a network slicing dataset is required before it was possible to create a new GNN model for predicting KPIs, as large and varied amounts of data are required to train a GNN model. As there were no available datasets including network slicing information, in this chapter the creation of a new and complete dataset for network slicing is described. These datasets are crucial for future research on applying AI/ML algorithms to network slicing and also for the next Chapter 5 of the thesis, where a GNN model is built to predict network slicing performance. The key contributions of this chapter are summarized as follows:

- Generating Network Slicing samples: the first step is to generate simulations that represent network slicing scenarios. At first, this task was done for one specific graph, and then it was expanded to cover various situations.
- Adapting the simulator: a simulator provided by BNN-UPC [32][2] with no support for network slices is extended to simulate these new network slice results, obtaining the resulting samples. This simulator is key to obtain the desired samples to later train the GNN model. BNNetSimulator is a packet-level network simulator that is built on the OMNeT++ framework. It allows users to easily generate network datasets for research and analysis. With BNNetSimulator, it is possible to define a network topology with various node and link attributes, configure source-destination routing, and establish a traffic matrix. The simulator then outputs performance metrics, including average delay, jitter, packet loss per path, link utilization, and statistics for QoS queues.ive understanding of this work, a description of the generated samples is provided. This evaluates the wide-range of values and use-cases in the context of B5G networks, thereby demonstrating the utility of this dataset.

In summary, this chapter serves as a pivotal bridge between data preparation and model development, to advance the state of the art in network slicing and laying the foundation for more models based in this dataset, enabling new technologies for the use-cases in the dynamic landscape of B5G networks.

The generated dataset is available for the research community in open access in Zenodo (`https://zenodo.org/records/10610616`). When uncompressed, the complete dataset weighs 6.55 GB and contains 7,900 different samples with varied traffic, graphs and capacities, as described in the following subsections.

In addition, from this work, a publication was obtained:

[1] M. Farreras, J. Paillissé, L. Fàbrega and P. Vilà. "Generation of a Network Slicing Dataset: The Foundations for Ai-Based B5G Resource Management". In *Data in Brief, Volume 55, 2024*. Q3 journal. `https://doi.org/10.1016/j.dib.2024.110738`.

[2]https://github.com/BNN-UPC/BNNetSimulator

4.1 Problem

Currently there is a lack of datasets dedicated to network slicing suitable for training machine learning models. This research aims to address this gap by generating new datasets using a network simulator. Specifically, the goal is to facilitate the training of models to predict network KPIs. An initial simplified scenario is developed to check the correctness of the simulations, followed by extensive simulations of various topologies to improve model generalization and evaluation.

There are two main solutions to generate a network slicing dataset:

- Physical testbed scenarios: simulating and collecting data from a physical network can provide accurate performance measurements in real conditions, allowing for real-world validation. However, scalability is limited, and it can be resource-intensive and costly to generate multiple network scenarios.
- Network packet simulators (synthetic simulations): these provide a cost-effective solution with precise control over network parameters and reproducibility. Conversely, they may not fully capture real-world network behaviors, and larger simulations can be time and resource consuming. Despite this, they offer high accuracy [100, 79].

Although network simulators provide high accuracy, it is important to clarify what "high accuracy" means in this context. Simulators can precisely model network parameters, routing configurations, and traffic patterns based on predefined mathematical models. They work well when producing deterministic, repeatable results under controlled conditions. However, they may not fully capture the complexities and variabilities of real-world networks, such as changing traffic demands or unpredictable hardware failures.

For instance, real-world networks might experience bursts in traffic that are difficult to simulate accurately. This limitation can lead to a difference between the behavior observed in simulated environments and that in actual networks. Therefore, while simulators are highly accurate in controlled settings, their accuracy might reduce when trying to replicate the sometimes unpredictable real-world network scenarios.

A safer approach to ensuring realistic performance evaluations is a hybrid methodology. In such an approach, a model could be trained using the rich data generated by simulators but validated and tested on physical testbeds, where real-world conditions can be incorporated into the validation process. By combining the scalability and control of simulators with the real-world insights from testbed environments, hybrid approaches can achieve a balance between practicality and realism. This ensures that models not only perform well under idealized conditions but also under complex scenarios found in actual deployments.

This network slicing dataset, generated through a realistic packet-level simulator, BNNet-Simulator, encompasses diverse scenarios, including different slice instances, traffic flows and QoS requirements for eMBB, URLLC, and mIoT applications. It facilitates rigorous evaluation of AI/ML models and algorithms for network performance prediction and flow allocation, especially in SLA-constrained scenarios. To sum up the main contributions to the simulator, support for different Weighted Fair Queuing (WFQ) queue weights on each node was added, and also the support for weights of 0 value (if a queue was not used), both of them being a limiting factor to generate multiple dynamic scenarios.

Network slicing in the simulator is implemented across the core and transport network segments. The transport and core networks include a packet labeling mechanism to differentiate between network slices. This mechanism allows packets to be mapped to specific queues, with the traffic within each queue corresponding to a different slice.

The Radio Access Network (RAN) falls outside of the scope of this dataset. The omission of the RAN is due to the complexity of simulating the wireless interface, which involves additional factors such as radio propagation, mobility, interference, and specific scheduling mechanisms at the physical layer, in addition of new simulation or hardware components. While the RAN is an essential part of end-to-end network slicing, the focus of this work is on the core and transport networks, as the majority of the state-of-the-art is focused on RAN network slicing. Additionally, incorporating the RAN would significantly increase the simulation time and complexity. For future work, integrating the RAN to provide a fully end-to-end slice simulation could be considered.

Researchers in B5G networks, AI, and network slicing can leverage this dataset for in-depth analysis, model training, and performance evaluation, especially regarding resource allocation algorithms. Network architects and engineers can use the dataset to consider diverse scenarios, aiding in the design and optimization of slicing strategies for different applications.

Other researchers can reuse this data to advance their work, validating and comparing existing models or developing new ones for predicting network metrics such as delay, jitter, and loss. The dataset also offers opportunities for experimentation with different network configurations, traffic scenarios, and slicing strategies.

This dataset serves as a benchmark for comparing different algorithms for network KPI prediction, serving as a starting point to develop and test new and improved traffic models.

4.2 Scenario description

The following initial scenario is considered (Fig. 32): the Core Controller receives slice requests and decides if they can be allocated in the network. The slice admission module incorporates a model used as a Slice Performance Digital Twin (DT), which, given the network topology, traffic requirements, and slice configurations, outputs the performance KPIs of each flow. Subsequently, the SLA validation module acts as an admission control algorithm, ensuring that all flows comply with predefined SLAs.

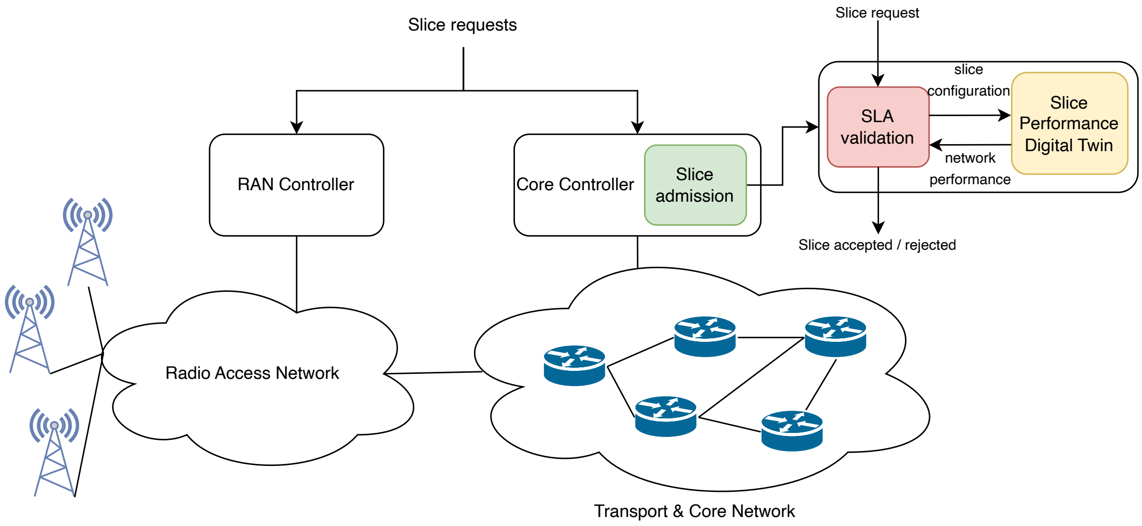

Figure 32: Functional block diagram of the slice admission controller.

Weighted Fair Queuing (WFQ) was the scheduling discipline implemented to provide isolation between the slices. Each queue in the WFQ system is associated with a specific slice, and the traffic for each slice is allocated a portion of the total link capacity based on the queue's weight. The queue weight is dynamically adjusted according to the traffic requirements of each slice. This ensures that each slice receives the necessary resources, while also allowing for scenarios where some slices are under or over-provisioned in terms of bandwidth.

Routers map these network slicing labeled packets to queues using a WFQ scheduling policy. This choice of queuing discipline is made to effectively isolate each slice from others [30]. Each queue serves a different slice, spanning from the originating RAN node(s) to the transport network's exit point. This isolation guarantees the allocated bandwidth for each slice, and if a slice experiences a reduction in traffic, the rest of the queues can benefit from the extra capacity. The weights of the WFQ queue are dynamically adjusted to meet the traffic requirements of each slice. This allows scenarios of over and underprovisioning, where some slices are allocated more or less bandwidth than they require, respectively.

The isolation among flows within a slice is maintained by reserving specific resources for each slice. The WFQ policy provides isolation at the queue level: each slice has its own queue in the WFQ system, ensuring that excess traffic in one queue will not impact other queues. This is crucial for maintaining the SLA for different services, such as URLLC (ultra-reliable low-latency communication) and eMBB (enhanced mobile broadband), which have very different QoS requirements. Therefore, even if one user within a slice generates more traffic, the system will prioritize the traffic based on the WFQ configuration, protecting the QoS for other users connected to the same slice.

However, in real-world deployments, implementing strict isolation using WFQ can be more challenging. The WFQ system is effective in simulations where network resources and traffic patterns are precisely controlled, but in a real-world setting, achieving the same level of isolation requires careful network planning and dynamic resource allocation strategies. The challenge lies in maintaining the queue weights across changing traffic patterns, link conditions, and service demands.

Concerning the flows, multiple flows of the same type are assigned to each slice instance, and several instances of the same slice type can coexist. Additionally, a slice instance can include multiple paths, each path with represented by an origin and a destination, allowing to create connections between the required nodes in the network inside a specific slice instance. In Fig. 33, an example illustrates the concept. The URLLC slice instance 1 traverses the top three nodes of the figure with 5 flows, while mIoT instance 1 (2 flows) and URLLC instance 2 (6 flows) are routed through the lower node. The WFQ queues facilitate the isolation of these instances, as the weight for each slice instance is determined during the admission step.

Lastly, as described in Chapter 2, three distinct slice types are considered, specifically those defined by 3rd Generation Partnership Project (3GPP) [4]: eMBB, URLLC, and mIoT. More concretely, in this secion Table 7 is presented, giving a summary of the intended network slice usages, and expected KPIs for each slice type.

For each slice, a set of QoS metrics is defined, based on various scenario variables, such as network size, number of clients, and application requirements. Delay, jitter and losses constitute the primary KPIs used to measure network performance for each network slice. These KPIs are defined for each slice type, although jitter values per slice are yet to be precisely defined in the existing literature. However, jitter is expected to be minimal, especially for the most critical usage, URLLC.

Table 7: Slice types considered and flow characteristics.

Type of flow	Maximum delay	Maximum packet loss	Intended use	Applications	Bit rate	Packet size [67]
eMBB	10 ms	10 %	Broadcasting, media delivery, gaming, general usage	Video streaming, Virtual Reality, High-speed Internet	18 Mbps	Average: 6000 bits Minimum: 12000 bits Maximum: 18000 bits
URLLC	0.5 ms	0.001 %	High reliability, ultra-low latency, high availability	Autonomous vehicles, Remote surgery, industrial automation	3.5 Mbps	Average: 800 bits Minimum: 256 bits Maximum: 1600 bits
mIoT	Up to 10s	1%	Long battery, low cost devices, extreme coverage	Smart cities, Agriculture, Smart metering	0.2 Mbps	Average: 160 bits Minimum: 320 bits Maximum: 480 bits

While the configuration of network slicing simulations in terms of capacity is clear, it is also essential to consider real-world 5G standalone (SA) [1] network configurations and how they are driven by demand, rather than purely by capacity. In 5G SA networks, telco operators configure network slices based on specific service requirements and user demand. For example, the number of eMBB slices in a network is generally determined by the demand for high-throughput services, rather than just available capacity.

In a real-world setting, the allocation of network slices, particularly for eMBB services, is highly dynamic and depends on real-time traffic patterns, user density, and geographic dis-

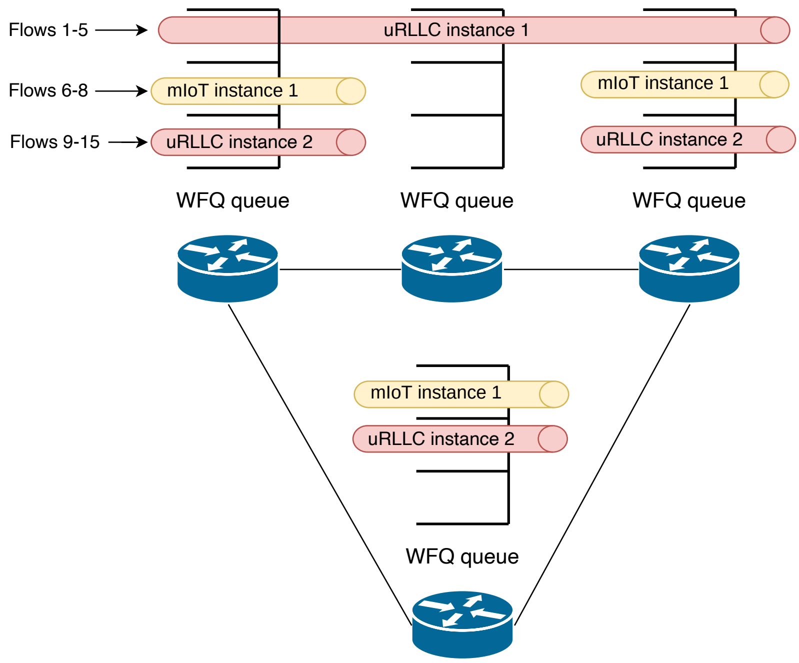

Figure 33: Mapping of flows to slices and slices to WFQ queues, distributed across different paths.

tribution. Telco operators typically monitor these factors using advanced network management systems and orchestration tools to configure slices accordingly. This dataset could be used as a resource to train a DTN, for obtaining KPIs from the current, past or future configurations of the network, and then apply configurations to the production network depending on the requirements.

4.3 Methodology

To obtain the samples, first it is need to generate simulation configurations for each sample. The dataset comprises pairs of (scenario, performance). The scenario includes all configuration parameters of the network scenario, such as the number and type of slices, traffic flows in each slice, packet arrival processes, WFQ queue weights, topology, and routing configurations. The performance includes the KPIs needed to validate if the SLAs of a slice are met, such as per-flow delay, jitter and packet loss.

For the first tests, a topology is simulated (Fig. 34) with a wide range of slicing configurations and traffic intensities. The traffic originates at different routers connected just after the RAN, and exits the network at the same router at the other end of it. The slices have the following characteristics:

- There are three different network slices types (eMBB, mIoT, URLLC) with a varying number of instances of the same slice.
- The proportion of slice types among all the simulations is variable. Each slice created can exist in one or multiple origin nodes.

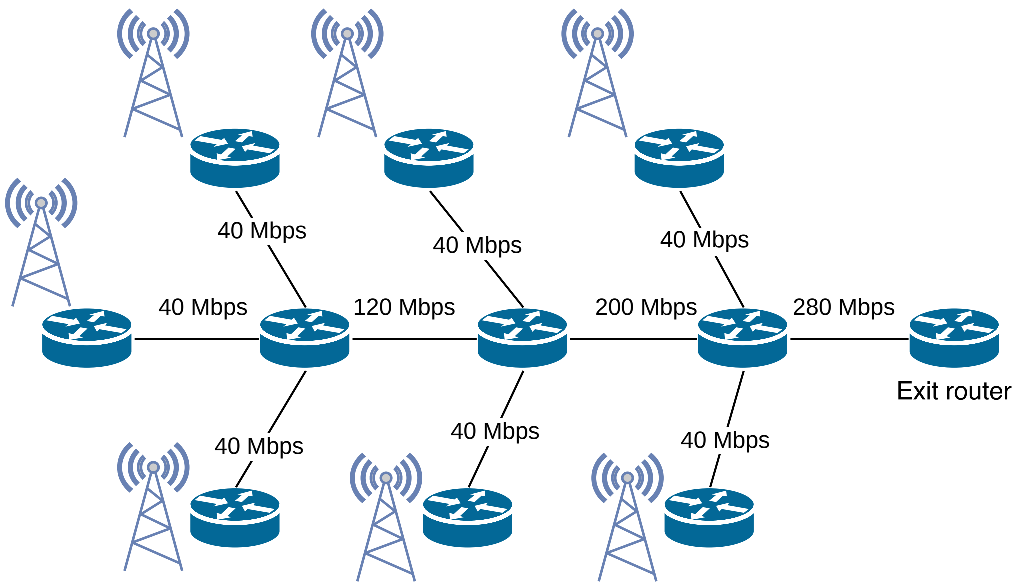

Figure 34: Initial simulated core and transport network topology. Routers with antennas are entry points from the RAN.

- Each slice has at least one flow assigned.
- At least one eMBB flow is included for each simulated origin node, as it is considered as the standard network usage. The rest of the flows for each access origin node and slice are randomly assigned. These flows instantiate any of the three types available.
- The creation of many flows is needed in order to fill in the link capacity. It is assumed that each flow requires an amount of bit rate ($V_{reserved}$), and these bit rates are added until the link capacity is reached.
- Each slice instance contains different flows with the traffic characteristics depicted in Table 7.
- The link capacities are fixed to an standard of 40 Mbps per origin node, calculating proportionally from [23] an standard capacity of 1 eMBB flow, 7 URLLC flows and 2 mIoT flows. These proportions are only used for defining the capacity of the links, they are not used later for the flow creation.

After testing with the initial scenario, multiple network topologies, traffic matrices, routing configurations and slice allocations are considered. Traffic flows originates from different routers connected just after the RAN part and exits the network to a router at the other end (Fig. 34 shows a simplified network example). The users are connected to the RAN using their User Equipment (UE). Each user is represented as a node connected to its origin router, represented with an antenna. That flow is then routed to the exit router shown in the right. A flow can represent either a single UE, or a device that uses different network slices, i.e. it sends data with different connections that have different QoS requirements. Therefore, a slice consists of a set of traffic flows from different sources to destinations with a common set of performance requirements. The allocation of network resources to slices is done in the following way: The output links of nodes use the WFQ queue scheduling policy. Traffic is classified in several virtual WFQ queues, and each queue is assigned a weight that determines the minimum portion of the link capacity that the traffic going through that queue will receive. The WFQ scheduling policy provides isolation among queues in the sense that excess traffic in one queue does not affect the traffic in other queues, i.e., each queue has a guaranteed minimum portion of the link's capacity. Each WFQ queue is assigned to an individual, specific slice, and the flows of this slice share the same queue. The size of each WFQ queue is static for all queues in each simulation, 32 packets. Each flow has a reserved bit rate, $V_{reserved}$, and their sum over all flows of a slice determines the required portion of the link capacity and therefore the queue weight; the sum of the reservations never exceeds the link's capacity. Given a flow with an average bit rate, its $V_{reserved}$ lies between two values, the minimum bit rate V_{min} (below the average) and the maximum bit rate V_{max} (above the average); a $V_{reserved}$ close to V_{min} represents a situation of resource underprovisioning and close to V_{max} of overprovisioning. The weight of the slice in each WFQ queue is calculated using the $V_{reserved}$, assigning the weight as the percentage of the total link capacity that the $V_{reserved}$ uses. The network scenarios simulated in this dataset aim to reflect real-world conditions by incorporating diverse topologies, realistic traffic patterns, and various network slicing configurations. The use of the Internet Topology Zoo [57] ensures that the topologies are based on actual network structures. Moreover, the traffic patterns and slice configurations are designed to mimic real-world applications, such as eMBB for media delivery, URLLC for high-reliability applications, and mIoT for IoT devices. Slices exhibit the following characteristics: Three different network slices types (eMBB, mIoT, URLLC) with a variable number of instances. The proportion of slice types is variable in each simulation, with each slice possibly existing in one or multiple origin nodes. Each slice has at least one assigned flow. At least one eMBB flow is included for each simulated origin node, serving as the standard network usage. When defining the rules for running the simulations in this thesis, it was necessary to balance between creating realistic scenarios and simplifying assumptions to make the simulations feasible. While the simulations use a smaller number of eMBB slices, the configuration reflects typical real-world usage where eMBB demands substantial bandwidth, requiring fewer slices but with higher capacity allocations. The remaining flows for each access origin node and slice are randomly assigned. Flows are created to fill link capacities, and reservations are made based on each flow's required average bit rate $V_{reserved}$.

Finally, each simulation has a different number of slice instances according to a tunable parameter that allows assigning more or less bandwidth than the required to the flows within a slice. This way, different scenarios can be simulated, that range from underprovisioning to overprovisioning. More specifically, the $\delta \in [0,1]$ parameter is defined, to control the amount of under and overprovisioning as:

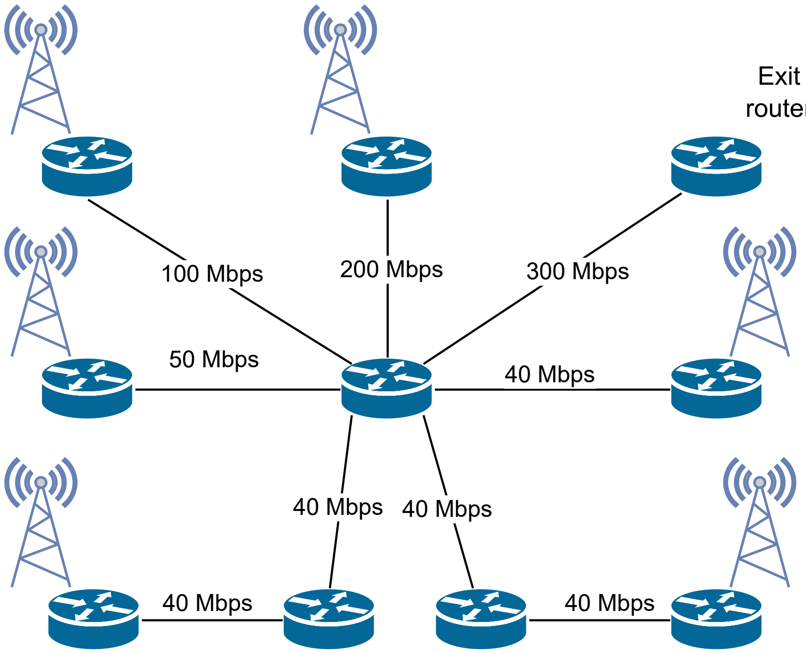

Figure 35: An example of the Ai3 network topology: routers with antennas (the RAN part) would be the origin nodes of the transport network while the exit router would be the destination node.

$$V_{reserved} = V_{min} + \delta(V_{max} - V_{min})$$

Where $V_{reserved}$ represents the actual reserved average bit rate in the simulation, δ the under/overprovisioning parameter, with a random value between $[0,1]$ for the flows in the same slice, and V_{min} and V_{max} are the minimum and maximum flow averge bit rates, respectively. This allows for a range of reserved bit rates from an under provisioning of V_{min} to an overprovisioning of V_{max}. The maximum and minimum bit rates are calculated using the average bit rate -10% for underprovisioning and +20% for overprovisioning, respectively.

To make the simulations possible, each source of a flow is technically an origin node added to the network, connected to the antenna where is located. The destinations of the generated flows are common for each type of flow, located at the right port of the exit router in the example graph (Fig. 35).

Successive to the evaluation of the initial samples with the simplified approach, further graphs are used to simulate more complex environments. To achieve this, the Topology Zoo dataset is used, obtanining random real-world graphs and adding randomized origin nodes, destinations, and capacities for the different links.

The ground truth is generated using a packet-level simulator, a modified version of OM-

NeT++ [33]. The 129 network topologies are obtained from the Internet Topology Zoo, simulating a wide range of slicing configurations and traffic intensities for each topology. Traffic originates from different routers connected just after the RAN and exits the network at other nodes selected as output routers on the other end. A simplified example can be observed in Fig. 35, where multiple antennas connected to origin nodes generate input traffic to the network and an exit router is placed at the right side. The different simulation scenarios are generated using the following steps:

1. Retrieve a network topology from the Internet Topology Zoo.
2. Randomly allocate link capacities from a predefined range of 25 to 500 Mbps.
3. Designate at least two nodes as origin nodes. A maximum of N/3 nodes should serve as origin nodes, where N represents the total number of nodes in the graph.
4. Randomly define the three different destination nodes (eMBB, mIoT, URLLC). The remaining nodes should be categorized as transport nodes.
5. Randomly set the number of slice instances between 1 and M, where M is the remaining number of unassigned type of nodes.
6. Specify the slice type and traffic for each slice.
7. Assign one eMBB flow for each origin node, while determining the others based on network capacity and reservations. Routing will be accomplished using the NetworkX Python library [45], obtaining first the shortest path and using the other possible routes when the path capacity is complete.
8. Perform under and overprovisioning tests using a custom access control script in Python to prevent exceeding the link capacity. The over and underprovisioning are calculated based on the sum of reserved bit rates ($V_{reserved}$) of the slices that traverse each link.
9. Eliminate any empty slices.
10. Execute the simulation using the network simulator.

After the generation of the required files (routing, traffic and topology) for simulating the scenarios, the simulator based on OMNeT++ is used for generating the samples of the dataset.

4.4 Results

4.4.1 Structure of the dataset

The dataset consists of a set of 7900 simulation results. There are three folders with 7900 files each (one with network topologies, one with routing configurations and another with slices). In addition, 79 .tar.gz compressed files are included. Each compressed file has 6 files with 100 results inside. Each line of each file represents the results of a unique simulation.

The dataset consists of a set of 7,900 simulation results, where each simulation corresponds to a unique combination of network topology, routing configuration, and network slice allocation. This number of samples represents a good starting point for training a GNN model. In the context of GNNs, which are data-hungry models, having a sufficiently large dataset is crucial to ensure that the model generalizes well and accurately predicts network performance under various conditions. The number of samples in the dataset is carefully chosen to balance the need for diverse scenarios with sustainable simulation times.

While 7,900 samples may seem limited when compared to other machine learning applications like image classification, in GNN training, the network's topology, routing, and traffic configurations add significant complexity to each sample, allowing the model to learn effectively even with a smaller number of samples. This number is consistent with similar research in the domain of network performance prediction using GNNs, where datasets often range from thousands to tens of thousands of samples [33].

In addition to the number of samples, the structure and diversity of the dataset play an essential role in the model's ability to generalize to unseen networks. With 129 different topologies derived from the Internet Topology Zoo, a broad spectrum of real-world conditions is covered, allowing the trained GNN to adapt to different networks and traffic conditions similarly to real B5G deployments.

Each simulation is composed by: one file with network topology data (graph_n.txt), one file with routing configuration data (routing_n.txt) a file with slice data (slices_n.json), and the corresponding line inside the compressed .tar.gz file containing n in the range. The folders and files structures are represented in Fig. 36. The x in the .tar.gz names is between [0,7800]

Firstly, network topologies for each simulation are stored in GraphML format [87], with nodes, queues and edges. These topologies are randomly selected from the Internet Topology Zoo, with added attributes. The node attributes are:

- id: Unique identifier in the simulation.
- type: node role, such as "transport", "origin_node", "destination_embb", "destination_mmtc" or , "destination_urllc".Origin nodes generate traffic, transport nodes forward traffic, and destination nodes receive traffic of a single type of slice.
- schedulingPolicy: "FIFO" (First-In First-Out) or "WFQ" (Weighted Fair Queuing)
- schedulingWeights: weights for each QoS queue (used with "WFQ").
- levelsQoS: number of QoS classes.

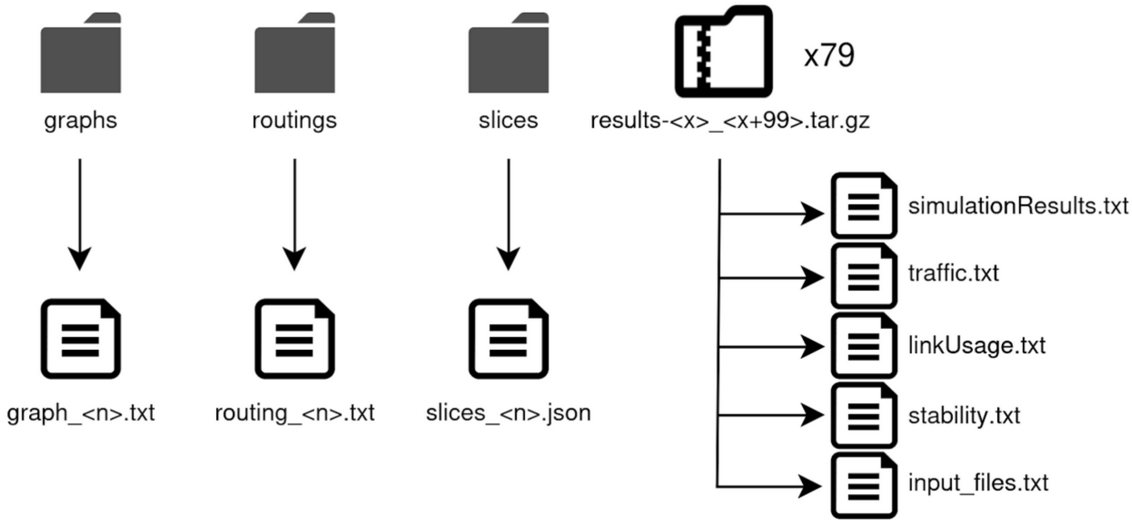

Figure 36: Folders and files structure of the dataset.

- queueSizes: capacities (in number of packets) for each output queue.
- tosToQoSqueue: service types assigned to each queue.

The attribute tosToQoSqueue maps flows to queues during simulations. Routing configurations are saved as .txt files, containing routing matrices for each node. Network slices are saved as JSON files, listing slices with multiple flows. Attributes of slices include:

- Delta (δ): value that determines whether the reserved average bit rate of a flow is close to the minimum flow bit rate or close to the maximum flow bit rate.
- Type: usage type of the slice. The three slice types defined by 3GPP [2] are considered: eMBB, mIoT, and URLLC.
- Number: numeric identifier for the slice.

Each flow has a unique origin and a destination node. Attributes of flows are:

- Origin: node where the traffic flow originates. It represents a device connected to an origin node.
- Destination: node where the traffic flow is received.
- Origin_node_antenna: node where the origin_node is connected. It represents the antenna node.
- Path: list of hops of a flow.
- Bandwidth: bits per second of the traffic generated.
- Traffic_string: traffic properties, including:
  - Type of traffic (Poisson in this case)

- Bandwidth generated by the flow, average number of packets per time unit, and exponential maximum factor (used for the Poisson distribution).
- Packet size distribution: size distribution, average packet size and minimum/maximum packet sizes for binomial distribution.

Finally, the results of each simulation are saved in a .tar.gz file, each containing groups of 100 simulations to reduce the total size and number of compressed files. The files inside each .tar.gz are briefly described below:

- simulationResults.txt: it contains flow and link information such as delays, jitter and losses.
- traffic.txt: defines the traffic for each simulation.
- linkUsage.txt: it includes the occupancy of each queue of each output port.
- stability.txt: it contains the iteration time, curly stability, status at the end, memory used and elapsed time.
- input_files.txt:indicates which network topologies, routing configurations and slices correspond to each line of the previously described files.

To facilitate reading the dataset contents for each simulation, the datanetAPI.py Python file is provided in the repository. It is based on the DataNetAPI project [15], but extended to also process the network slicing data collected in this work. To use it, it can be called from a Python code as shown in Fig. 37. Then, creating an iterator for the dataset and obtaining the data structures.

```python
from datanetAPI import DatanetAPI
tool = DatanetAPI("/path/to/dataset/extracted", shuffle=False)
it = iter(tool)
for s in it:
    G = nx.DiGraph(s.get_topology_object())
    R = s.get_routing_matrix()
    T = s.get_traffic_matrix()
    D = s.get_performance_matrix()
    P = s.get_port_stats()
    S = s.get_slices()
```

Figure 37: Basic code to iterate through each sample of the dataset.

The shuffle option can mix the different samples (e.g. for providing an input to a training Machine Learning algorithm). Each sample contains the following features:

- global_packets: total number of packets transmitted in the network per time unit (packets/time unit).

- global_losses: total number of packets lost in the network per time unit (packets/time unit).
- global_delay: Average per-packet delay over all the packets transmitted in the network (time units).
- maxAvgLambda: overall traffic intensity of the network scenario. Maximum average traffic rate (bits/time unit) that a path can generate in the simulation scenario. Note that this traffic rate may be split into several flows, sending traffic over the same src-dst path.
- performance_matrix: Matrix with aggregate source-destination and also flow-level performance measurements (delay, jitter and loss) measured on each source-destination pair.
- traffic_matrix: Matrix with the time and size distributions used to generate traffic for each source-destination pair.
- routing_matrix: Matrix with the paths to connect every source-destination pair.
- topology_object: NetworkX graph object including topology-related information at the node and link-level.
- links_performance: list of dictionaries with the performance metrics associated with each link.
- port_stats: list of dictionaries with the performance metrics associated with each output port.

In terms of individual samples, the node count ranges from 16 to 764 nodes. The simulations stop when the stationary state is detected, where the state of the simulation reaches the equilibrium level at a certain point. That is, when the difference between the maximum and the minimum of the delays, divided by the mean delay, is smaller than 0.001 during two evaluation windows. Simulation time spans between 1.4 hours and 30.4 hours, while the memory utilization varies from 217 MB to 30 GB.

The hardware used to run the simulations was a cluster with 40 computing nodes, each with two Intel Xeon E5-2630L v2 2.40 GHz CPUs, 128 GB of RAM and 1 TB disks. The simulations were allocated in the cluster (sharing resources with other processes), with a limitation of 30 GB of RAM.

4.4.2 Data analysis

Some metrics of the simulations are shown in Table 8. It can be observed that, as expected from the characteristics of the types of slice, the average number of eMBB slices is lower due to the higher bit rate of each slice, while there are many more mIoT slices due to their lower bit rate.

Table 8: Metrics of the generated simulations.

Metric	Value
Total samples	7900
Number of different topologies	130
Average samples per topology	61
Average total number of slices per sample	219
Average total number of eMBB slices per sample	9.2
Average total number of URLLC slices per sample	45.6
Average total number of mIoT slices per sample	164.9

In the following subsections, the distribution of the values for the different set of features at the flow, queue, and link level has been analyzed.

4.4.2.1 Flows

Fig. 38 shows the Probability Density Function (PDF) of values of the numeric features of the slices. For each type of slice, it can be observed that the delay and logarithm of the delay (*AvgLnDelay*) is higher for the mIoT slices, while it is lower for the URLLC and eMBB slices. The jitter is low in all cases, being slightly higher for the mIoT slices. The jitter is calculated as the standard deviation of the variance of the delay. The delta (δ) values are mostly uniform for each type of slice, while the packet loss is lower in the URLLC slices. It can be seen that the parameters present a wide range of values in order to cover as many scenarios as possible. For example, the delta parameter presents a quasi-uniform distribution between 0 and 1.

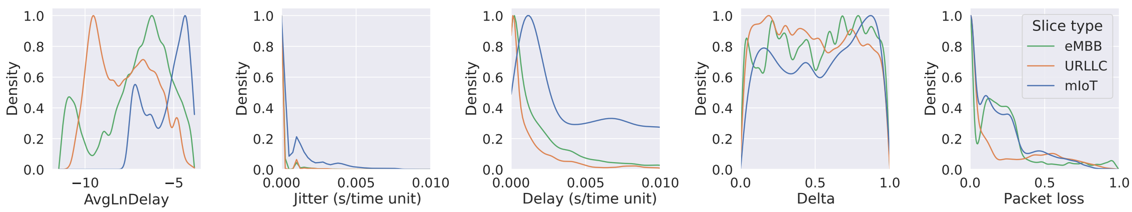

Figure 38: Probability Density Function (PDF) of several flow features for each slice type.

It can be seen that the parameters present a wide range of values in order to cover as many scenarios as possible. For example, the δ parameter presents a quasi-uniform distribution between 0 and 1.

4.4.2.2 Links

Fig. 39 and Fig. 40 show the link data distributions, showing high utilization, as it was intended in the scenarios, low losses in most cases, various bandwidths and an *offered traffic intensity* (O_t) with most values around 0 and 1, although other values also exist. O_t is a feature created from other existing features and it is detailed in Eq. (5). It is defined as the sum of traffic (T_f) of all flows f passing through the network link N_l divided by the bandwidth C_l of that link, resulting in a scalar feature assigned to the link state.

$$O_t = \frac{\sum_{f \in N_\ell} T_f}{C_\ell} \tag{6}$$

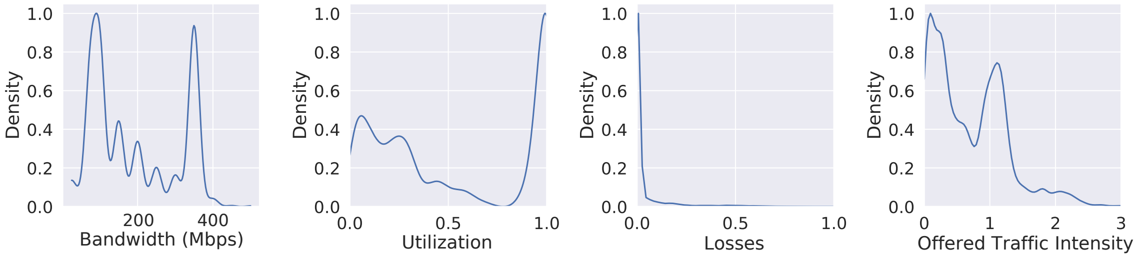

Figure 39: Probability Density Function (PDF) of several link features for each slice type.

The links that use FIFO queues, which connect each antenna to the origin node, are excluded from the analysis, as they are overdimensioned for any type of traffic and therefore are not significant for the isolation of the WFQ queues.

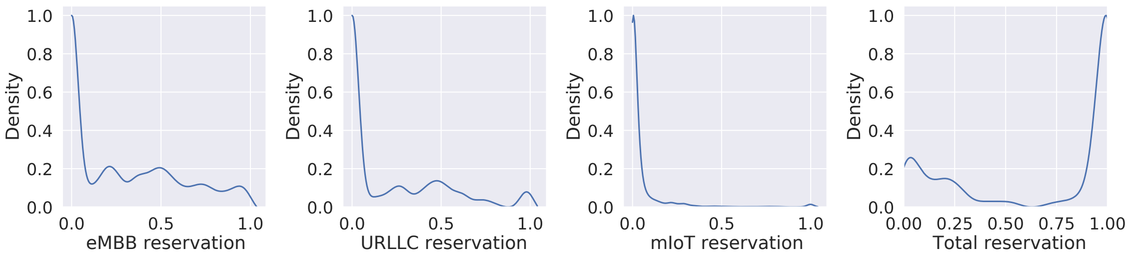

Figure 40: Probability density function of the reservations of the links.

4.4.2.3 Queues

Finally, Fig. 41 shows the Probability Density Function (PDF) of several queue features for each slice type. It can be seen that most weights of the WFQs queues are under 50%. The utilization of the queues is higher for the eMBB slices, while it is very low for the mIoT case. This is due to the fact that eMBB slices are the ones with the most bandwidth assigned. The losses are lower in the URLLC queues analyzed as also seen in the links, while the delays are also lower in the URLLC. The maximum queue occupancies, represented in number of packets, are shown to be a little bit higher in the URLLC case on average, but the total maximum queue occupancy is reached more commonly by the mIoT slice types. It can be seen that the queues are full in some situations. This is expected, in order to have some packet loss.

The dataset includes 7,900 simulation results, each corresponding to a unique combination of network topology, routing configuration, and network slice allocation. This diversity is critical as it reflects different network configurations and complexities encountered in

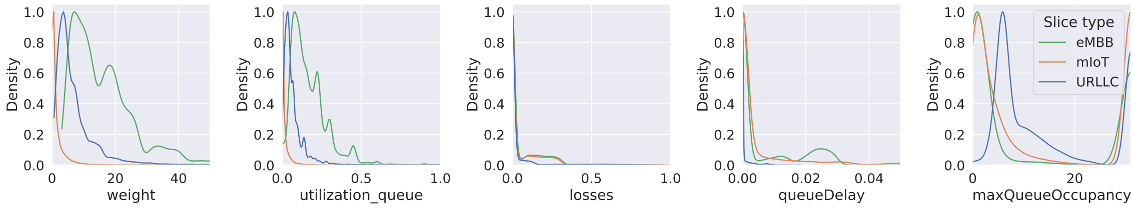

Figure 41: Probability Density Function (PDF) of several queue features for each slice type.

real-world scenarios. 129 different network topologies have been used, meaning that, on average, 61 different scenarios have been built on each network topology. This repetition allows for variations in traffic patterns, routing decisions, and network slice allocations to be explored. By varying these parameters, it can be observed how each topology behaves under different conditions, encompassing a wide range of operating conditions. This can help the models trained using this dataset to capture the behavior of the network under multiple conditions.

4.5 Limitations

The primary limitations of this study were related with time and memory demands of the simulations. Despite parallelization efforts, individual simulations required approximately two hours, so the cumulative time for the volume of generated simulations was substantial. In addition, the parallelization was limited by the fact that every simulation required an allocation of 30 GB of RAM memory to run successfully. Larger simulations (with more traffic or more nodes) were discarded due to resource and simulation time limitations.

In order to validate the representativeness of the proposed dataset, statistical analyses were carried out to ensure that the performance metrics are not biased towards any specific topology or scenario. This includes checking the distribution of performance metrics across different topologies and scenarios to ensure a balanced representation. Despite the efforts to create a realistic dataset, some simplifications and assumptions were required. These are presented, paired with a possible mitigation strategy:

- Exclusion of RAN: The simulations focus on the transport network, excluding the RAN infrastructure. While this limits the ability to simulate end-to-end scenarios, the exclusion does not harm the dataset's utility for evaluating core and transport network slicing. The goal of the dataset is to provide insights into the performance of the core and transport segments, which are critical for resource allocation and KPI prediction. Future work could incorporate the RAN to expand the scope.

- Traffic patterns: Although Poisson traffic is widely used in network simulations, it may not fully capture the burstiness of real-world traffic. This simplification could lead to discrepancies in real-world application, but the dataset's large size and diverse scenarios help mitigate this by covering a wide range of traffic intensities and patterns. Furthermore, additional traffic models could be introduced in future iterations to address this limitation.

- Fixed routing configurations: The use of static routing does not account for dynamic adjustments that can occur in real-world networks. This could affect the realism of the simulations under changing network conditions. However, the dataset includes multiple routing configurations across different scenarios to approximate such dynamics. Future extensions could explore dynamic routing simulations to enhance realism.
- Fixed UE location: In the current setup, users are static during each simulation. While this limits mobility testing, the dataset can still provide valuable insights into network behavior under static conditions, which are essential for foundational resource allocation studies. Moreover, the dataset includes multiple slices and flows, which increases variability and reduces the impact of this simplification.

In addition, network topologies have link capacities of up to 500 Mbps and a maximum number of nodes of 764; going beyond would have required much more computational resources and simulation time, and these values achieved a good balance between simulating realistic scenarios and an affordable simulation time. Three types of slices have been considered, eMBB, URLLC and mIoT, and although there are others, these are the most common. The flow's traffic model for the all three types of slices has been the same and consisted of exponential inter arrival packet times and binomial packet size, but with different values for each slice type to model their different characteristics. Finally the network scenarios consider the transport network and the RAN part is not included. Future work could involve dynamic routing protocols, more traffic models and RAN components.

While a direct comparison to real-world measurements is not performed in this thesis, the extensive variety of scenarios simulated can serve as validation for models. Among this variety of scenarios, some will exhibit greater or lesser degrees of similarity to real-world over- and under-supply situations, allowing artificial intelligence models to capture the range of variability that real networks may present.

In summary, while these limitations introduce certain constraints, the dataset remains highly useful for core and transport network slicing studies. The mitigation strategies ensure that the obtained dataset provides valuable data with network performance, even under these constraints.

4.6 Conclusions

The simulations generated have demonstrated a satisfactory level of quality, rendering them suitable for training a neural network model effectively. Furthermore, the ease of usage of the dataset including extensive documentation and an adapted DataNet API, provides a seamless access to the data for future research, enabling easier analysis and later training of a neural network.

In the forthcoming chapter, the dataset derived from these simulations assumes an elemental role as the foundation for training a GNN model. This utilization emphasizes the practicality of the dataset and exemplifies the potential for advancing the research of network slicing management and optimization.

Thus, the significance of this dataset extends far beyond its immediate application, laying the groundwork for future breakthroughs in network slicing research and paving the way for a more efficient and resilient communication infrastructure.

5 GNNetSlice: A Graph Neural Network model for Network Slicing KPIs prediction

This section of the thesis describes the model created to predict network slice KPIs of a given network. Using the previous acquired knowledge about GNNs as described in Chapter 3, and the creation of the required network slicing dataset in Chapter 4, in this section a model called GNNetSlice is proposed to predict performance metrics (delay, jitter, loss) of the transport and core networks (Fig. 42). This way, operators can know beforehand if the network can satisfy a set of network slicing demands, or if a new slice can be added to the network.

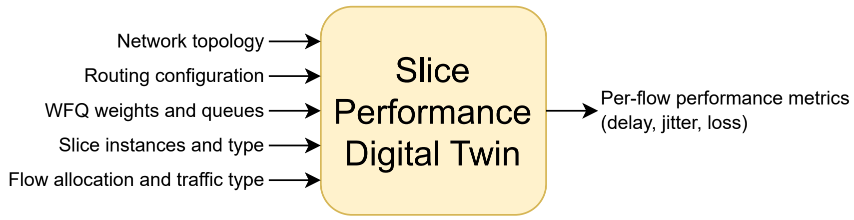

Figure 42: Black-box representation of the model used as part of a DT to predict network slice performance.

From this work, a publication has been prepared and submitted:

[4] M. Farreras, J. Paillissé, L. Fàbrega and P. Vilà. "GNNetSlice: A GNN-based Performance Model to support Network Slicing in B5G Networks". In "Computer Networks". Q1 journal. Under review.

5.1 Problem description

As seen in the previous chapters, efficient management of network resources is crucial for building next-generation networks. Resource management remains a topic of ongoing discussion and development in B5G networks, especially regarding network slicing [60, 86] and the different usage types available.

However, effective resource management for diverse network slices is computationally expensive due to: (i) the large number of connected devices, (ii) the massive amounts of data generated by these devices, and (iii) the heterogeneity of the networks (Radio Access, core, edge).

To overcome these obstacles, resource management in B5G networks requires sophisticated prediction algorithms in combination with decision-making processes. In order to guarantee network KPIs, e.g. latency or losses, network operators need accurate and fast tools to manage the allocation of resources to each network slice. A critical part are network models that predict the performance of all the flows, with the aim of ensuring that the network can meet the SLAs for all flows and slices. This way, either the network operator or an automatic resource allocator can easily determine if a new flow or slice can be allocated, making it possible to respond faster to dynamic flow requests.

As introduced in Chapter 3, tools like Queuing Theory and Discrete Event Simulators estimate network performance. Queuing Theory oversimplifies modern networks complexity [96], while simulators offer accurate packet-level results but require much more resources as network size and complexity grow. Therefore, near real-time analysis, monitoring, and management of current networks remain challenging [65].

Numerous studies have explored applying AI and ML to B5G resource management. As shown in Section 5.2, data-driven approaches like Deep Learning, Reinforcement Learning, and GNNs show promising optimizations in resource allocation, fault detection, and traffic management [19]. These approaches require extensive B5G data to adapt to changing conditions, offering a tradeoff between Queuing Theory and network simulators.

GNNetSlice predicts performance metrics (delay, jitter, losses) for network slices in core and transport networks. Fig. 42 depicts the input data of the model, and the output predictions. This tool allows operators to assess network configurations in near real-time (e.g. add new slice, add/remove flows, etc) before deploying them in production, ensuring the satisfaction of network slicing demands. This usage is in line with the concept of a DT [101], a virtual replica of a physical network, that mimics one or more characteristics of the physical network.

GNNetSlice is built using a Graph Neural Network (GNN), offering high accuracy in predicting network slicing KPIs. Using a GNN also provides a swift prediction of the behavior of the network for each scenario, which is a pivotal element to use the model as part of a DT [93].

Furthermore, the approach to create the dataset is described, analyze the network properties, build the model, train it, and obtain predictions. GNNetSlice is evaluated and compared with recent state-of-the-art GNN approaches, as well as the results of a packet-level simulator.

The contributions of this research can be summarized as: (i) creating a state-of-the-art dataset with simulations of network slicing reservations, (ii) providing a GNN model design, (iii) proposing an accurate model for network slicing performance metrics estimation, and (iv) presenting an extensive set of experimental evaluations.

5.2 Related work

5.2.1 Traditional techniques for network performance prediction

In the previous chapters, an state-of-the-art for predicting network KPIs is already provided, being extended in this section, with a focus in predicting network slicing KPIs.

The current state-of-the-art contains a large variety of techniques for predicting network performance, including analytical modeling methods such as Queuing Theory [24], Markov chains [22, 54], and other related methodologies. As discussed in earlier chapters, although these models often provide quick predictions, their accuracy is degraded due to their dependence on unrealistic and static assumptions about packet arrival processes and real-world network conditions [32].

To achieve more realistic results, packet-level network simulators are commonly utilized, offering fine granularity of the experiments. However, these simulators are computationally intensive and time-consuming, particularly for larger networks, making them unsuitable for scenarios that require near real-time predictions [92]. Nevertheless, the high accuracy of data generated by these simulators is useful for training various ML models, as demonstrated in [98].

5.2.2 ML-based Network Performance Prediction

When it comes to network slicing, the demand for rapid network analysis algorithms in evolving scenarios has led to exploration in AI and ML techniques for accurate predictions with low response time. Significant efforts include a zero-touch control for network slicing proposed in [12], predicting the network capacity needs of each slice, using the previous traffic demands of mobile traffic data to shape the future requirements. Results of two forecasting blocks, one short term, and the other long term, are combined to aggregate the demand of the total shared capacity and allocate it in the short-term.

Other ML methods, such as Logistic Regression (LR), Support Vector Machine (SVM) and Decision Trees, have been implemented in [53] to predict packet's Round-Trip Time (RTT), where data of RTT measurements for training the models are obtained from mobile operators in Italy. Then, the aforementioned methods are used to predict the packet's RTT, concluding that Decision Trees (DT) had a much better precision. Still, this was an initial approach to demonstrate the usability of the dataset.

Additionally, DT approaches are proposed to predict network traffic handling in [13, 12], using RNNs. RNNs, more concretely Long Short-Term Memory (LSTM), increase the long term memory function, saving a selected history of past predictions that may be relevant for the current prediction, as used in [37]. Static and in-vehicle 5G data is used to train and predict using three different methods: CNN, Gated Recurrent Unit (GRU), and LSTM, the last obtaining their best results.

RNNs are also employed in [5] to predict delays in 5G networks for the IoT and Tactile Internet. The model is trained in a simulated IoT system, collecting and later predicting the IoT traffic network delays.

5.2.3 Application of Graph Neural Networks to computer networks

In the context of computer networks, GNNs learn directly from graph-structured data. This data includes network topologies, routings, and offered traffic, in order to predict network KPIs. Research has been conducted on applying GNNs in various scenarios of computer networks. For example, autonomous network management is exemplified in [36], where GNNs are used to enable reinforcement learning agents to adapt policies of autonomous mobility-on-demand. Another application in network slicing monitoring [76] demonstrates the effectiveness of GNNs, combined with genetic algorithms, for locating network monitors. GNNs have also been utilized to predict network delay in standard transit networks, represented as fixed-size graphs [31]. Finally, SDN end-to-end delay prediction in [38] employs advanced techniques, such as Spatial-Temporal Graph Convolutional Networks (STGCN), which integrates the RNN technique to GNNs, enhancing the delay prediction.

The main differences of GNNetSlice, in comparison to recent methods utilizing ML, DL, and GNNs, are summarized in Table 9. GNNetSlice offers several key advantages, including fast training, high accuracy, a simple and unified model, a focus on predicting network slicing environment performance, and the public availability of both code and datasets.

Table 9: Comparison GNNetSlice and other contributions focusing on GNN and Network Slicing.

Contribution	Deep learning model	Architecture	Ground truth	Source code availability	Hyperparameter tune	Fast train	Feature engineering	Multiple models	Focused on Network Slicing
[12]	CNN	4D-CNN + MLPs	Real scenario	Private	N/A	N/A	Yes	No	No
[37]	RNN	SLSTM	Real data	On request	N/A	N/A	Yes	No	No
[5]	RNN	NARX	AnyLogic Simulator	Private	N/A	Yes	Yes	No	No
[53]	Classification models	SVM, DT, LR	Real data	Private	N/A	Yes	Yes	No	No
[36]	GNN	GCN + MLPs	Real data	Public	N/A	N/A	Yes	No	No
[76]	GNN	GAT + MLPs + genetic algorithm	Genetic Algorithm + greedy solution	Private	Yes	N/A	Yes	Yes	Yes
[38]	GNN	STGCN	OMNeT++ simulation	Private	Yes	No	Yes	No	No
[9]	GNN	MPNN + GRU	OPNET simulation	Private	N/A	Yes	Yes	No	Yes
[27]	GNN	GCN/MTDRL	Open dataset, Tor [59] + OpenDayLight scenario	Private	N/A	N/A	Yes	No	Yes
[33]	GNN	MPNN + GRU	OMNeT++ simulation	Public	Yes	No	Yes	No	No
GNNetSlice	GNN	GRCN	OMNeT++ simulation	Public	Yes	Yes	Yes	No	Yes

5.3 Scenario

As in the previous chapter, the network consists of a core network and a transport network, with the RAN falling outside of the scope. The following functions are considered in the admission control block (Fig. 32): the Core Controller receives slice requests and decides if they can be allocated in the network. The slice admission module incorporates GNNetSlice, that acts as a Slice Performance Model, which, given the network topology, traffic

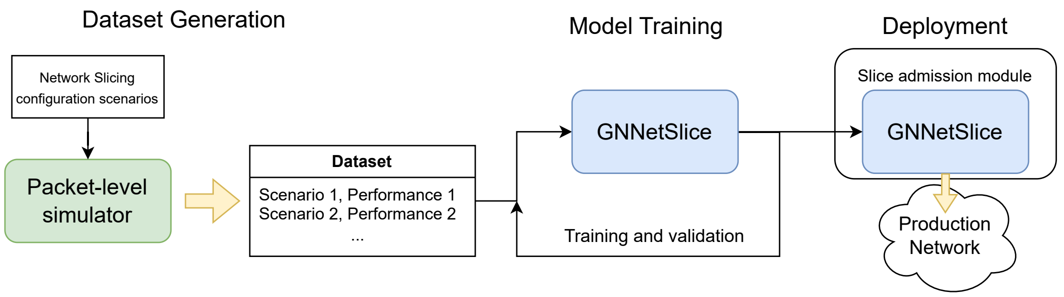

Figure 43: Machine Learning workflow, from dataset generation to model deployment, for constructing the GNNetSlice.

requirements, and slice configurations, outputs the performance KPIs of each flow. Subsequently, the SLA validation module acts as an admission control algorithm, ensuring that all flows comply with predefined SLAs. The Core Controller syncs the configurations with the RAN Controller, to align the configurations for the changing network requirements.

As stated in Section 2.1.2, three distinct slice types are considered, specifically those defined by the 3GPP [4]: eMBB, URLLC, and mIoT. Each slice type is designed for different use cases (Table 7). For each slice, a set of QoS metrics is used, such as network size, number of clients, and application requirements. Delay, jitter and losses constitute the primary KPIs used to measure network performance for each network slice. These KPIs are defined for each slice type, although jitter values per slice are yet to be precisely defined in the existing literature. However, jitter is expected to be minimal, especially for the most critical usage, URLLC.

5.4 Methodology

In order to build the GNNetSlice GNN-based model for estimating slice performance, a data-driven approach is adopted, typical in ML workflows (Fig. 43). First, a dataset using a network simulator is generated, comprising pairs of (scenario, performance), representing the model's inputs and outputs, respectively, as shown in Section 3.4. The scenario includes all configuration parameters of the network scenario, such as the number and type of slices, traffic flows in each slice, arrival processes, WFQ queue weights, topology, and routing configuration. performance includes the KPIs needed to validate whether the slice's SLA is met, such as per-flow delay or packet losses.

While throughput is also a critical performance metric, particularly for eMBB applications, it was not included in the initial implementation due to the focus on metrics that have more direct implications for latency-sensitive services. However, throughput is an important KPI for ensuring quality in eMBB slices, which typically require high data rates. Future work could extend the model to incorporate throughput prediction, or a post processing of the obtained predictions could be used for calculating an estimation of the throughput using the existing GNNetSlice model.

Next, a GNN model is trained to predict network performance based on parameters such as network topology, routing, traffic and network slice configuration. The dataset is split into training, validation and testing sets, and hyperparameters are adjusted until an acceptable accuracy is achieved, typically below 20% of error. To accomplish this, the model is trained multiple times, trying multiple hyperparameter values. Finally, the trained model would be ready to be deployed in a production network as part of the slice admission controller.

It is important to note that this approach is training the GNNetSlice model *offline*, meaning it is trained in the lab and later expected to be deployed in a production network. This approach is analogous to autonomous cars, which are trained by the vendor and then sold to customers already trained. The key reason for this approach is that the GNN model needs to learn various extreme scenarios, such as high delays, broken links, or high packet losses, and replicating these situations in a production network would be challenging due to potential disruptions and SLA degradation.

5.4.1 Dataset

The dataset used, which is a part of the generated dataset detailed in chapter 4 comprises 3498 samples, each consisting of the following components:

1. Topology: includes links and nodes. The example topology in Fig. 34 has 11 nodes representing a router, 3 final nodes representing the destination of each type of slice, connected to the source nodes. The quantity of these flows depends on the remaining capacities of each link of the path. This group of data includes the queue configurations, with features such as weight, utilization, losses, delay and max. occupancy.

2. Routing: the path of each flow is defined by a shortest path algorithm from source to destination. If the shortest path is already fully reserved, the next shortest path available is used.

3. Flow traffic: based on the standards defined in Table 7, specifying flow-level source-destination time and size distributions (e.g., AvgLnDelay, jitter, delay, δ, packet loss).

4. Slice reservations: origin and destination nodes, source nodes used, reserved bit rate ($V_{reserved}$), δ, slice type and unique identifier.

5. Link performance: performance measurements at the link level, such as bandwidth, utilization, losses and offered traffic intensity.

6. Flow performance: Per-flow QoS measurements such as dropped packets, average delay, jitter).

5.5 GNN libraries and architectures

To construct a high-performing model, selecting a well-documented GNN library is crucial. A GNN library is a software framework designed to facilitate the development, training, and evaluation of GNNs. The frameworks typically include several implemented architectures to develop models, as shown in this section. Several library options are available for creating such models, with the most commonly used ones including:

- Tensorflow/Keras [64]: TensorFlow provides a versatile platform for building and training neural networks, including GNNs. Keras, as a high-level API for TensorFlow, simplifies the implementation of GNN architectures.
- Pytorch/PyTorch Geometric [34]: PyTorch is another widely used deep learning framework with a dynamic computational graph. PyTorch Geometric is an extension library that simplifies the implementation of GNNs, offering various graph-related operations and pre-defined layers.
- Deep Graph Library (DGL) [91]: DGL is a dedicated library for building and training GNNs efficiently. It supports both TensorFlow and PyTorch backends, providing a high-level API for graph manipulation and GNN model development.
- Spektral [43]: Spektral is another GNN Python library offering seamless integration with TensorFlow and PyTorch. Providing essential GNN layers, graph operations, and support for tasks like pooling and signal processing, Spektral is versatile for various graph-based machine learning applications.

The GNN architecture refers to the specific structure and design of a GNN model, including its layers, nodes, and the way they are interconnected. Several types of architecture exist, depending on the problem to solve, such as node or edge regression, node or edge prediction, or graph classification. The most common GNN architectures are:

- Graph Convolutional Networks (GCNs) [55]: GCNs are one of the earliest and most popular GNN architectures. They aggregate information from neighboring nodes using a convolutional-like operation, enabling effective node representation learning.
- GraphSAGE (Graph Sample and Aggregation) [46]: GraphSAGE generalizes the idea of sampling and aggregating information from neighboring nodes. It allows the model to scale to large graphs by sampling a fixed-size neighborhood for each node.
- Graph Attention Networks (GATs) [89]: GATs introduce attention mechanisms to assign different weights to neighboring nodes during message passing. This enables the model to focus on more informative nodes, improving its expressive power.
- Graph Isomorphism Network (GIN) [94]: GIN is designed to be invariant to the order of nodes in a graph and can capture different levels of neighborhood information. It achieves this by using an aggregation function that is order-agnostic.

These are a subset of the diverse GNN available architectures. Research is continuously exploring new models to address specific challenges in graph-based learning tasks.

5.6 GNNetSlice model

The objective is to predict delay, jitter, and loss for the generated dataset. For constructing the GNNetSlice model uses the Deep Graph Library (DGL) for PyTorch [91], as it provides good documentation and support for multiples types of graphs. The overview of the most significant steps for creating the GNN model are the following:

1. Load the dataset using an adapted version of Barcelona Neural Networking Center's Application Programming Interface (API) [15]. Split it into training, validation, and testing sets, with percentages of 80%, 10% and 10%, respectively. Transform the original network topology graphs into input graphs [61].
2. Define the GNN model structure, and choose the hyperparameters values and relations
3. Select the input features and normalize using a min-max normalization, where the minimum value of the feature X_{min} is substracted from the actual value, and the result is divided by the difference between the maximum X_{max} and X_{min} (Eq. (7)).
4. Train and validate the model for n epochs. Predict the testing set labels to verify the correct training and accuracy. Optimize hyper-parameters to improve the training time and accuracy.

$$N_x = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{7}$$

5.6.1 Input graph

The GNNetSlice model has the ability to predict multiple attributes simultaneously using a single model. The model is trained to predict delay, jitter, and losses of the flows in the network. To perform this flow attribute prediction, link features need to be used. In the context of link attribute input and regression, there are limitations in the current state-of-the-art GNNs. To address this, the input graph is reshaped by transforming links into nodes. This reshaping creates a new input graph, allowing GNNs to indirectly predict link attributes as node attributes and bridging the gap between GNN capabilities and link attribute regression challenges [41]. More concretely, to predict flow features using link and queue features, it is needed to create an input graph that translates the original network topology graph to an input graph. This transformation involves converting individual links, queues and flows into nodes, creating an input graph structure for each sample. This results in three distinct node classifications: flow nodes, queue nodes, and link nodes. The goal is to establish a clear distinction and relation in data treatment between features at the flow level (e.g., end-to-end delay and packet losses), the queue level (e.g., WFQ weights and utilization), and the link level (e.g., link delay and utilization).

In Fig. 44 an example of how the GNN input graph is generated from the original network topology graph is shown. In the original network topology graph, the queues are represented as *Q_1_1*, *Q_2_1*, and *Q_2_2*. The original router nodes are *R1*, *R2*, and *R3*. *F1* and *F2* are two flows, and *L1* and *L2* are the links in the network. The status of a flow is tied to the states of all the links traversed by that flow, establishing the relation flow *traverses* link. Conversely, the status of a particular link is influenced by the conditions of all the flows that traverse that link, creating the relation link *composes* flow. In addition, a flow goes trough n queues, with the relation *uses*, and a queue serves paths using the relation *serves*. Finally, a link output port can host queues, establishing the relation *hosts*, and that queue is related to the link using the relation *resides*.

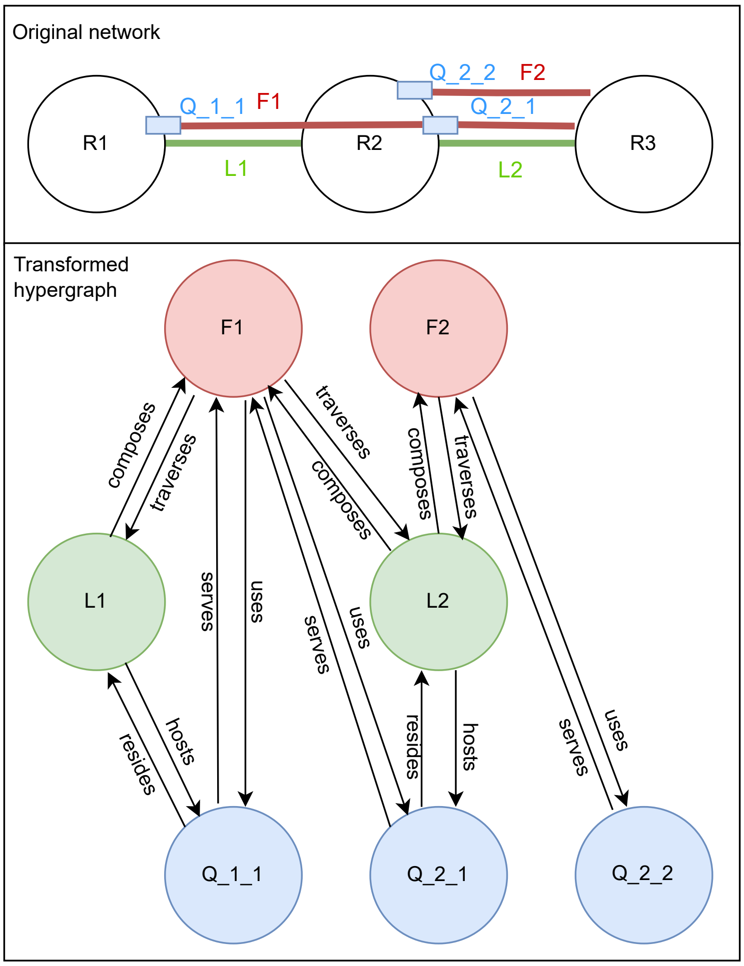

Figure 44: Example of an original network topology graph (top) transformation to the input graph (bottom).

5.6.2 Algorithm

The base model for prediction is a Relational Graph Convolutional Network (RGCN) [82]. This model supports multiple types of relations in the input graph, a crucial functionality for defining relations between flows, queues, and links. It consists of two graph convolutional layers, called GraphConv layers [55], each using a ReLu [63] activation function to regularize the outputs, where the hidden states of each node are trained and later predicted. The model calculates a hidden state for all the nodes in the input graph. Then, the HeteroGraphConv module, a module from DGL for computing convolutions on heterogeneous graphs, is used to create the RGCN model with the GraphConv layers. This module structure is applied to each type of relation defined in the graph, to later perform the message passing between related nodes. In addition, the aggregation function is applied, in this case the summation. Algorithm 1 describes the architecture of the model in detail while Fig. 45 shows the relations between the building modules of GNNetSlice, representing the algorithm more graphically. The algorithm operates on an heterogeneous input graph G, which consists of various types of nodes representing different components of the network. These components include flows (F), queues (Q), and links (L). The blue, orange and red arrows in Fig. 45 represent the relations between the three types of node.

The algorithm starts by initializing the hidden state variables h_f, h_q, and h_l for flows, queues, and links, respectively. They are initialized with the feature embeddings X_f, X_q, and X_l encoding the initial features of flows, queues, and links (lines 1-3).

Then, the message passing phase begins (lines 4-18), which consists of T iterations defined as a hyperparameter during the final hyperparameter optimization. In each iteration T, the algorithm updates the hidden states of flows f in the graph H, queues q, and links l, based on their interactions. For each flow f (lines 7-10), the algorithm updates its hidden state h_f using the function M, which takes into account the states of the set of queues and links the flow passes through. Similarly, for each queue q, the algorithm updates its hidden state h_q using the same function, considering the states of the flows passing through it and the link where it is located. Likewise, for each link l (lines 11-14), the algorithm updates its hidden state h_l, considering the states of the queues and the flows. Following the previous steps, for each link q, the hidden state h_l is updated (lines 15-18) with the states of the related links and flows. This is possible due the characteristic of the HeteroGraphConv module, where each type of relation has its own GraphConv layers, which are trained differently according to the relations seen during training. Therefore, M can be seen as 6 different configurations of the model, depending on the relation needed to update the hidden states.

This iterative process continues for T iterations, allowing the algorithm to refine the hidden states of flows, queues, and links based on their interactions within the network. Finally, the algorithm returns the updated node features X_F', using the readout function for the flows R_f (lines 20-21).

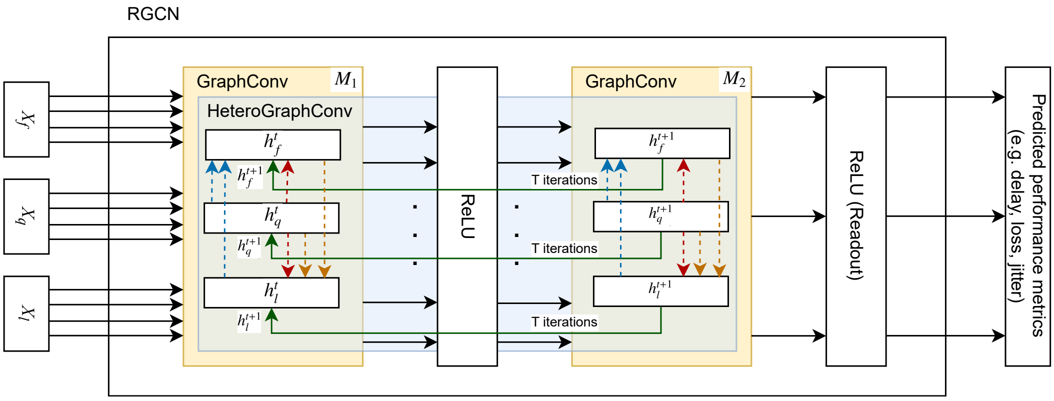

Figure 45: Schematic representation of GNNetSlice.

5.6.3 Input features

The selection of input features is crucial for the convergence of a ML model, in addition increased precision and also to improve the training speed [66]. The features used in the model are detailed in Table 10. Each type of node has four features. For nodes of type flow, Traffic and Packets are obtained directly from the simulation results, while Path length is calculated from the input graph, and δ is derived from the slice properties defined before the simulation. For nodes of type link, the features include Capacity (defined before the simulation), Utilization (U_l), AvgPktSize, and O_t. Finally, the nodes of type queue contain the Weight (w) in the WFQ queue (already defined before simulating), utilization (U_q), theoretically calculated losses (C_l), and theoretical MaxDelay (M_d). These queue-related features are derived using traditional Queuing Theory.

Feature engineering was employed to create new features from the existing data in the training dataset, aiding the model in generalizing and improving the prediction accuracy, as introduced in [26]. Three engineered input features are created for the GNNetSlice model. Firstly, O_t, as defined in Section 5.4. Secondly, Calculated losses (L), which are determined by the Eq. 8. L is defined as the O_t minus the product of weight w and capacity C_l of the output link, which is the carried traffic, and divided by O_t, resulting in a scalar feature assigned to the link state. Thirdly, MaxDelay M_d is created, which represents the maximum delay an average size packet might experience in a full queue. It is calculated using Eq. 9, where Q_s is the queue size in number of packets, multiplied by the AvgPktSize A_p in number of bits, and divided by the product of weight w and the output link capacity C_l.

$$L = \frac{(\sum_{f\in Q} T_f) - (w \times C_\ell)}{\sum_{f\in Q} T_f} \tag{8}$$

$$M_d = \frac{Q_s \times A_p}{w \times C_\ell} \tag{9}$$

Algorithm 1 Model architecture

Input: Hypergraph H, queue features X_q, link features X_l, flow features X_f
Output: Updated flow features X_f'

$h_f^0 \leftarrow H_f(X_f)$
$h_q^0 \leftarrow H_q(X_q)$
$h_l^0 \leftarrow H_l(X_l)$
for t = 0 to T-1 **do**
  $M_1 \leftarrow GraphConv(h^t)$
  $M_2 \leftarrow GraphConv(h^t)$
  for f in H **do**
    $h_f^{t+1} \leftarrow h_f^t + M_1(h_l^t, h_f^t)$
    $h_f^{t+1} \leftarrow h_f^{t+1} + M_2(h_q^t, h_f^{t+1})$
  end for
  for l in H **do**
    $h_l^{t+1} \leftarrow h_l^t + M_1(h_f^t, h_l^t)$
    $h_l^{t+1} \leftarrow h_l^{t+1} + M_2(h_q^t, h_l^{t+1})$
  end for
  for q in H **do**
    $h_q^{t+1} \leftarrow h_q^t + M_1(h_l^t, h_q^t)$
    $h_q^{t+1} \leftarrow h_q^{t+1} + M_2(h_f^t, h_q^{t+1})$
  end for
end for
X_f' $\leftarrow R_f(h_f^t)$
return X_f'

These engineered features can help to enhance the accuracy of GNNetSlice predictions, as they are guiding the model to an approximated estimation of the prediction. For example, while not giving exact values, the Symmetric Mean Average Percentage Error (SMAPE) (detailed in the next subsection) of L in eMBB, URLLC, and mIoT slices, in comparison to the real losses of each flow, stands at 5.34%, 14.92%, and 5.12%, respectively.

5.6.4 Model Training

To measure the prediction accuracy, the SMAPE metric is used. SMAPE quantifies prediction accuracy as the average percentage of relative errors (Eq. (10)). This relative error is obtained comparing the predicted value $\hat{y}_l$ with the actual value y_i in absolute value, the result of which is divided by the sum of $\hat{y}_l$ and y_i, both in absolute value and divided by two. Lower SMAPE scores indicate more accurate predictions. The key difference in comparison to the MAPE metric is that the MAPE degenerates into positive infinity when actual values are zero or close to zero. The usage of SMAPE or similar variants correcting this problem is essential, as many values for losses and jitter are zero in the simulations.

$$SMAPE = \frac{100\%}{n} \sum_{i=1}^{n} \frac{|\hat{y}_l - y_i|}{(|y_i| + |\hat{y}_l|)/2} \tag{10}$$

Table 10: Input features used in GNNetSlice.

	Feature	Definition
Flow	Traffic	Average bandwidth of the flow (bits/time unit)
	Packets	Packets generated by the flow (packets/time unit)
	Path length	Total number of hops of the flow, including first and last nodes
	δ	Resource provisioning parameter of the slice the flow belongs to
Link	Capacity	Bandwidth of the link (bits/time unit)
	Utilization U_l	Occupation of the link (in the range [0,1])
	AvgPktSize A_p	Average packet size of all outgoing packets through the link, in bits
	Offered Traffic Intensity O_t	Sum of Traffic of all flows passing through link divided by the link's Capacity, derived from Queuing Theory
Queue	Weight w	WFQ queue's weight (in the range [0,1])
	Utilization U_q	Occupation of the queue (in the range [0,1])
	Calculated losses L	Packet losses derived from Queuing Theory (in the range [0,1])
	MaxDelay M_d	Maximum queuing delay derived from Queuing Theory (in time units)

After hyperparameter tuning, the best GNNetSlice model has been trained during 95 epochs. It was found that, generally, larger values of the hyperparameters result in higher precision. Generally, when creating ML models, a good hyperparameter selection is crucial [99].

The training lasted 4 hours and 15 minutes to achieve the best SMAPE results for the three KPIs predicted in this work. Specifically, 95 epochs with batches of 1000 graphs were employed. A batch is a subset of the dataset used in one iteration of training a machine learning model, enabling more efficient computation and more stable convergence compared to processing the entire dataset at once. The training was performed using only the CPU, as the model performed slower on GPU than on the CPU. This occurs because the acceleration provided by the GPU does not sufficiently compensate for the overhead generated by copying the batches of data. This enhances the model's usability across a broader range of devices, reducing requirements without sacrificing training time or obtaining lower precision results. The training hardware consisted of an AMD Ryzen 5 5600X CPU 3.7 GHz and 64 GB of DDR4 RAM.

Table 11: Tested values for optimization of the hyperparameters.

Hyperparameter	Tested values	Optimized values
Batch size	1, 5, 10, 15, 20, 25, 50, 100, 200, 500, 1000	1000
Early stopping window	10, 20, 30, 40	40
Learning rate	0.1, 0.01, 0.005, 0.001	0.005
Layer size	4, 6, 8, 10, 20, 30, 40, 50, 60, 80, 100	80

5.7 Evaluation

For the evaluation of the GNNetSlice model a suite of established metrics is used, including SMAPE, Mean Square Error (MSE), Mean Average Error (MAE), and Coefficient of Determination (R^2). The SMAPE serves as a measure of accuracy, quantifying the average percentage difference between actual and predicted values, offering a symmetric approach that addresses issues related to scale and zero values, as introduced in Section 5.6. MSE and MAE provide insights into prediction errors by assessing the average squared and absolute differences, respectively, allowing to measure the model's performance with varying degrees of emphasis on error magnitudes and directions. Finally, R^2 is used as a concise measure to gauge how well a model explains the variability in the dependent variable, where higher and closer to 1 values indicates a stronger fit and better predictive performance.

The model is evaluated through a wide range of tests categorized as follows:

1. Prediction error of eMBB, URLLC, and mIoT flows.
2. 10-fold cross-validation using the entire dataset.
3. Prediction error depending on the topological properties, such as graph size and nodal degree, to demonstrate the generalization capabilities of the model.
4. Comparison with an state-of-the-art model for predicting network KPIs.
5. Inference and training speed.

Apart from 10-fold cross-validation, the tests are performed with the testing subset. This subset is obtained dividing the dataset by selecting samples randomly, in a proportion of 80% training, 10% validation and 10% testing.

5.7.1 Prediction error with different slice types

Firstly, the prediction error is evaluated across the different types of network slices: eMBB, URLLC, and mIoT. The performance of GNNetSlice is analyzed using the SMAPE. Table 12 presents the SMAPE errors for each slice type, indicating the level of prediction accuracy achieved.

Table 12: Symmetric Mean Average Percentage Error (SMAPE) of GNNetSlice with the different slice types.

	Delay	Losses	Jitter
eMBB	2.26	2.07	2.01
URLLC	2.17	1.99	1.91
mIoT	7.71	2.04	1.96
All slices	5.22	2.04	1.95

Furthermore, Fig. 46 illustrates the cumulative distribution functions (CDFs) of relative errors for each slice type. Each graph shows the distribution of prediction errors relative to the actual values. The error distribution for mIoT is slightly higher compared to eMBB and URLLC, although within acceptable bounds, as also seen in Table 12, suggesting that GNNetSlice is able to maintain reliable predictions across various service classes.

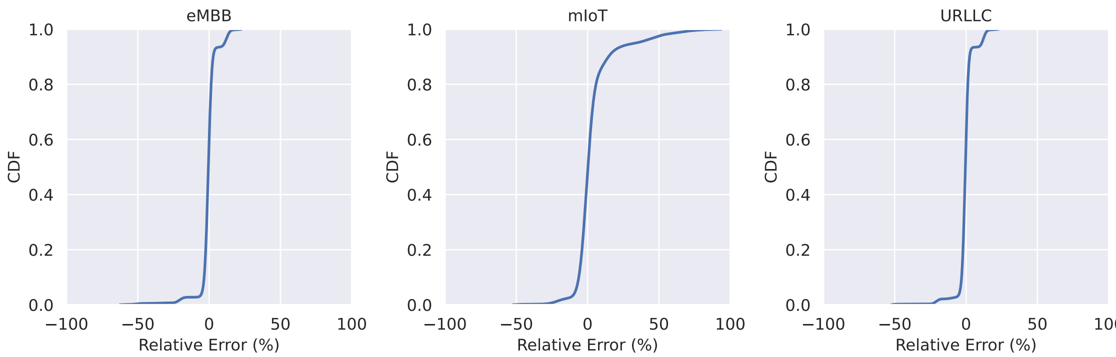

Figure 46: Cumulative Distribution Functions (CDF) of the relative error of the predictions testing the presented model.

Additionally, it was observed that the SMAPE errors for delay, jitter and losses, when analyzed with respect to the parameter δ, exhibit stability within a regular and bounded range. This underscores the robustness of GNNetSlice in maintaining consistent prediction accuracy for multiple over and underprovisioning scenarios.

Summing up, these results demonstrate GNNetSlice's effectiveness in accurately predicting network performance metrics across diverse network slices, with minor variations observed in prediction error across different slice types.

5.7.2 Cross-validation

Secondly, the results of a 10-fold cross-validation are presented, using the entire dataset to assess the generalization of GNNetSlice. The procedure of the 10-fold cross-validation is to separate the entire dataset in 10 random splits, and test the model with a different split each time, using the rest of splits available to train the model. The typical number of folds of a cross-validation is 10, as seen in [58]. The cross-validation results are depicted in Figure 47, where boxplots illustrate the distribution of SMAPE errors for delay, losses, and jitter prediction across the folds. The SMAPE errors for delay prediction range between 2% and 5%, indicating consistently accurate predictions within a narrow range of error. Similarly, the errors for losses prediction fall between 1.8% and 4.5%, while the errors for jitter prediction range from 0.8% to 3.3%. These results demonstrate the robustness of GNNetSlice in capturing and predicting various network performance metrics, generalizing well to unseen data.

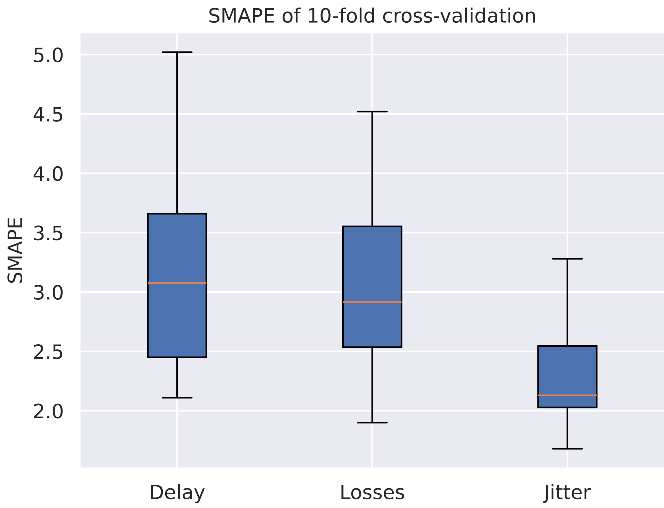

Figure 47: SMAPE errors for delay, losses, and jitter prediction with the 10-fold cross-validation.

5.7.3 Topological generalization

Thirdly, the performance of GNNetSlice is evaluated using two topological properties: the number of nodes of each original graph obtained from the Internet Topology Zoo, and the average nodal degree. The results of this analysis are presented in Fig. 48. The first column illustrates the SMAPEs of GNNetSlice when predicting flow delay, losses and jitter across different graph sizes, from 5 to 100 nodes. Across all scenarios, the model consistently demonstrates low levels of error, indicating its robustness in accurately predicting the desired flow KPIs. The second column focuses on the SMAPE of the flow predictions, categorized by the average nodal degree of the original graph. Despite the variety of configurations, with an average nodal degree from 1.5 to 4.5, GNNetSlice maintains stable and low error rates. Some outliers are observed, mainly because of the large number of graphs inside the range of 10-19 nodes and nodal degree 2-2.5. Another example is an increased SMAPE in the range 3-3.5, as the number of samples is much lower in that range. Even in this cases, the prediction error stays in an acceptable range.

Overall, this analysis provides an evidence of GNNetSlice's effectiveness in producing accurate predictions. The stable and low error rates observed across diverse number of nodes and nodal degrees highlight the model's ability to generalize well and make reliable predictions.

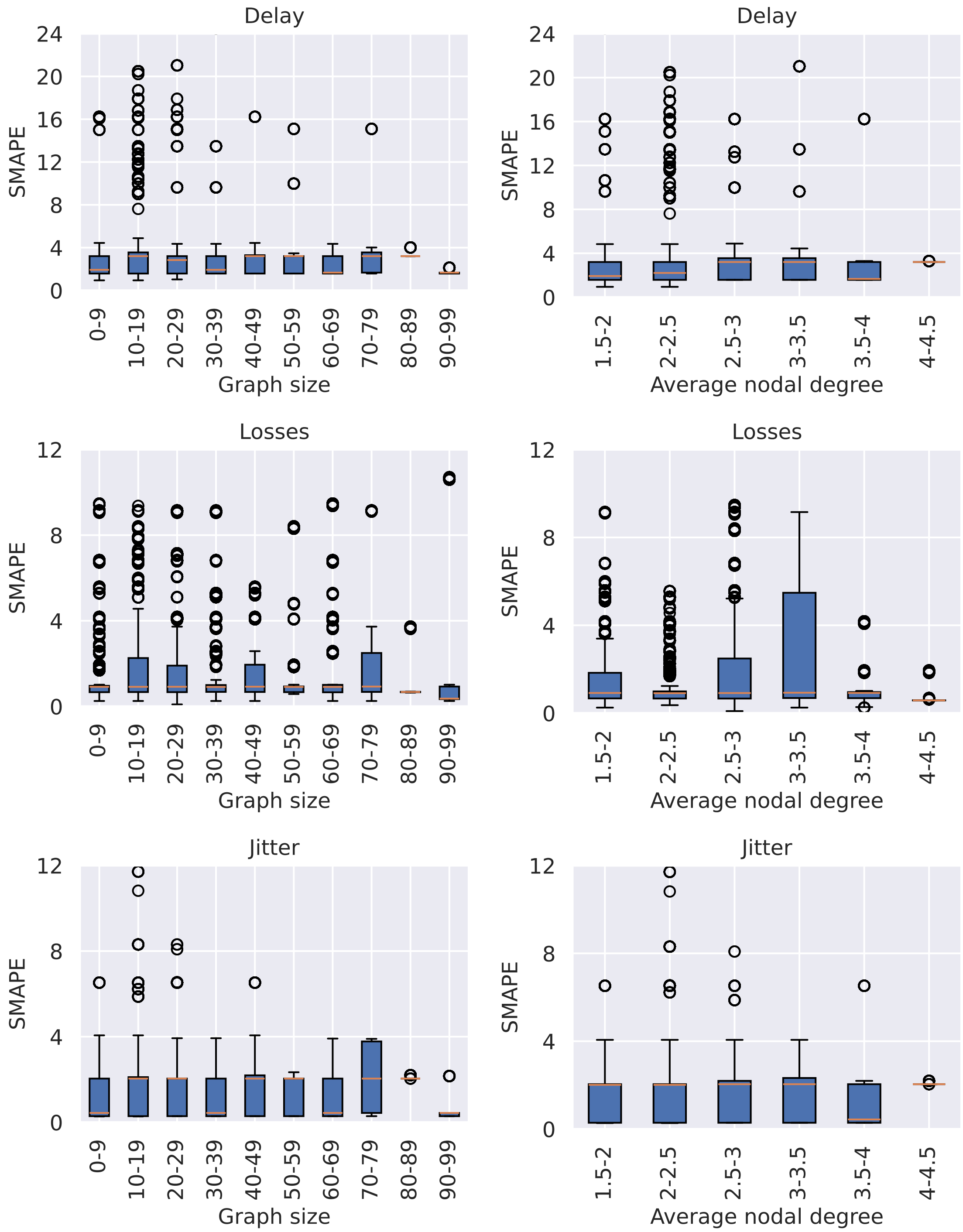

Figure 48: SMAPE of the predictions of GNNetSlice, per number of nodes and average nodal degree.

5.7.4 Interpretation of error values in real-world applications

While the percentage errors reported for GNNetSlice are low, it is essential to interpret these in the context of realistic application requirements, particularly for delay-sensitive services.

For example, in eMBB applications such as video streaming or infotainment, a 2-5% prediction error for latency may not result in significant user-perceived performance degradation. For instance, if the expected delay is 50 ms for a video streaming application, a 5% error corresponds to approximately 2.5 ms, which would be tolerable within the quality expectations for those services.

In the case of URLLC applications, which are critical for use cases such as industrial automation, remote surgery, or autonomous vehicles, even small prediction errors can be problematic. In URLLC scenarios, the expected latency is often below 1 ms, and a 5% error would translate to an additional 50 microseconds, which is a relatively low time difference.

In summary, GNNetSlice offers good prediction accuracy for critical applications requiring low-latency communications. The current prediction error ranges can presumably be tolerated for most consumer and time-critical services.

5.7.5 Comparison with State of the Art models

Fourthly, the performance of GNNetSlice is evaluated with a state-of-the-art model, RouteNet-Fermi [33], in predicting network performance metrics such as delay, losses, and jitter. Both models are trained and tested using the same subsets of the simulated data.

Table 13 presents a comprehensive comparison of SMAPE, MAE, MSE, and R^2 values obtained by both models across the three prediction tasks with the provided dataset. Remarkably, GNNetSlice consistently outperforms RouteNet-Fermi across all evaluated metrics. GNNetSlice demonstrates lower SMAPE, MAE and MSE values, indicating more accurate predictions with reduced errors. Additionally, the higher R^2 values for GNNetSlice indicate a stronger correlation between predicted and actual values, signifying superior explanatory power. These findings confirm the capabilities of GNNetSlice to model complex network slicing scenarios.

Table 13: Comparison of the the most common error metrics of the predictions between GNNetSlice and Routenet-Fermi.

	GNNetSlice			RouteNet-Fermi		
	Delay	Losses	Jitter	Delay	Losses	Jitter
SMAPE	5.22	2.04	1.95	65.5	6.87	4.86
MAE	0.0042	0.0022	0.0019	0.259	0.071	0.124
MSE	0.0016	0.0003	0.0002	11.868	0.005	0.008
R^2	0.89	0.95	0.88	0.56	0.76	0.84

5.7.6 Inference and training speed

During the testing stage, the model demonstrated an average prediction time of 26 ms per sample for delay, jitter, and losses. This short prediction time suggests that GNNetSlice could be effectively used in near real-time scenarios. For example, this prediction speed allows network operators to assess an entire network performance rapidly enough to apply the decision of accepting or rejecting a new network slice instance. This exemplifies the capabilities of AI/ML approaches in B5G network management. While time-critical services are present in B5G networks, the creation and management of network slices can be a near real-time service, making this prediction time perfectly

A notable strength of the proposed model is its remarkable efficiency during training. Each simulation, aimed at achieving slightly more accurate results, required an average of 1.7 hours for training. The model achieves swift convergence and learning, exclusively using CPU resources. This not only contributes to lower resource consumption but also facilitates rapid updates, enabling the model to adapt quickly to evolving datasets from real-world scenarios. Offline training could be performed while the production model is making predictions in the production network, enabling the system to learn from newly observed statistics from the network.

5.7.7 Reproducibility of the results

Reproducibility is a crucial aspect of any data-driven model. The GNNetSlice model was designed with reproducibility in mind, and its results can be reproduced under specific conditions using the provided dataset, tools, and model architecture. The dataset generated for this research is publicly available to ensure that other researchers can access and replicate the exact experiments.

The simulations used to generate the dataset in this thesis are based on packet-level simulations of core and transport networks with network slicing configurations. By following the methodology outlined in Chapter 4, researchers can reproduce the dataset and subsequently the model results. The provided DataNet API script ensures that each simulation configuration can be loaded to the model with ease, ensuring that the simulation environment can be replicated in a lab setting.

In addition, the usage of datasets is not limited to simulations, as real world data can be easily adapted with the usage of the DatanetAPI Python Code, which can be used as a plugin for reading any kind of dataset and loading it to GNNetSlice. The only constraint is to provide the same features to the model as done during the developement of the model, which can be seen in Section 5.6.

Moreover, the use of well-documented libraries like PyTorch Geometric and the Deep Graph Library (DGL) for implementing GNNetSlice further enhances reproducibility if new changes or adaptations are needed.

One challenge in transitioning to real-world datasets is the presence of noisy data and unforeseen network conditions that are not captured by simulators. In real-world networks, variability in hardware performance, unexpected traffic surges, and intermittent failures can introduce noise that may not have been present in a controlled simulation environment.

Future work could focus on hybrid approaches, where the model is initially trained on synthetic datasets from simulations but later fine-tuned with real-world measurements. This hybrid methodology could ensure that the model is trained by both the precision of simulated environments and the unpredictability of real-world traffic.

In conclusion, the reproducibility of GNNetSlice is well-supported in simulated environments, and with appropriate adjustments, the model is adaptable to real-world datasets.

5.8 Conclusions

This section introduced GNNetSlice, a model built to predict the performance of network slicing in core and transport networks in B5G deployments. The model predicts essential performance metrics (delay, jitter, losses) of all network flows. This predictive capability empowers the network controller to make decisions regarding flow allocations, ensuring compliance with the SLAs of the associated network slices. This makes it possible to safely test various slice configurations before deploying them in production.

The proposed methodology relies on a data-driven approach, achieving a good balance between speed and accuracy. Initially, a dataset is constructed, capturing the performance metrics of a network implementing network slicing across the three primary use cases (eMBB, URLLC, and mIoT). This dataset was generated using a packet-level simulator. Subsequently, this work has designed and trained a Graph Neural Network to predict these metrics. The results demonstrate that GNNetSlice can be used effectively to provide accurate predictions for delay, jitter and losses across diverse network scenarios. Notably, GNNetSlice exhibits low resource requirements, near-real-time prediction capabilities, and an efficient training process.

6 Conclusions and future work

The research conducted in this thesis has demonstrated the potential of AI/ML models, specifically GNNs, in advancing the management and optimization of B5G networks. More concretely, as the research evolved, the adaptability of GNNs was discovered, as seen in the proposed models GAIN and GNNetSlice. These models showcase the capability of GNNs to accurately predict KPIs such as per-path network delay, jitter, and packet losses, which are crucial for network slicing and general network performance management. These contributions represent an advancement in the use of ML for B5G networks, offering a practical solution for optimizing network resource allocation.

The main contributions of this work are:

1. **GAIN model**: The GAIN model is built starting from RouteNet, incorporating enhancements such as feature normalization, feature selection, and hyper-parameter optimization. These modifications enable GAIN to achieve superior scalability and generalization, particularly when predicting per-path network delay in larger network topologies. GAIN's lower implementation overhead, reduced resource requirements, and faster training times make it a practical solution for predicting KPIs in real-world network scenarios.

2. **Dataset generation**: The simulations created have proven to be of high quality, serving as a foundation for training neural network models. However, limitations such as the exclusion of RAN, static routing configurations, and traffic patterns need to be addressed in future work to make the dataset even more realistic. Despite these limitations, the dataset sets a benchmark for future advancements in network slicing management and optimization.

3. **GNNetSlice model**: GNNetSlice is designed to predict performance metrics (delay, jitter, losses) for network slices in B5G networks, contributing in optimizing network slicing management. The data-driven approach employed ensures a balance between speed and accuracy, with the model demonstrating low resource requirements and near-real-time prediction capabilities. GNNetSlice's ability to provide near-real-time predictions while maintaining high accuracy makes it an efficient tool for optimizing network slicing management.

The strongest results in this work include GNNetSlice's ability to predict delay, jitter, and packet loss with low error rates. In a head-to-head comparison with the state-of-the-art RouteNet-Fermi model, GNNetSlice consistently outperformed in terms of SMAPE, MAE, and R^2 values. Specifically, GNNetSlice achieved a SMAPE error below 5% for delay and losses in most scenarios, showing strong predictive accuracy that supports its real-world applicability. One of the key advantages of GNNetslice is the ability to predict with this level of precision while using fewer computational resources. In addition, the generation of a dataset for comparing different models predicting network slice acpkpi has been a good contribution for the research community. The first contribution of this thesis, the improvement of the RouteNet model in the ITU Challenge, GAIN, served as a foundation for the rest of the work in this thesis, providing the required learning for analyzing graph datasets and processing them into new GNN models.

However, despite the promising results, there are limitations to be addressed. For instance, the models have primarily been tested in simulated environments, and their performance in real-world scenarios requires further validation. The lack of datasets, nonetheless, has been partially solved by this thesis, providing a useful tool for developing new models and perform an initial validation of the predictions.

The GNN models have a significant potential to develop interdisciplinary applications. Similar GNN-based approaches can be adapted for other complex systems, such as water networks, transportation networks, supply chain management, and smart grid optimization, demonstrating the versatility of GNNs.

In the topic of this thesis, the evolution of GNNs in network management seems promising. As B5G networks evolve and the transition to 6G networks begins, the foundational work presented in this thesis can be useful in the development of more advanced and efficient network management solutions. The continuous refinement and validation of AI and ML models will likely provide even more powerful tools for network optimization, enabling innovations that will enhance the performance, reliability and security of future communication networks.

As a future work, many improvements and new advancements could be achieved. A list of possible future research ideas could be:

- **Limitations of the dataset**: Future work should focus on generating datasets from real-world networks, including dynamic routing and additional traffic patterns, to validate the models under adverse conditions.
- **Exploration of new GNN architectures:** Continue exploring alternative and new GNN architectures, such as Spatio-Temporal GNNs, and improvements, including advanced feature engineering and optimization of input attributes. This exploration could lead to models that are more efficient, accurate, or with better generalization capabilities.
- **Integration of new input graphs:** Test the integration of additional relations into the input graphs, for example incorporating temporal features to capture the evolution of network states.
- **Creation of closed-loop control applications:** Application of the models in closed-loop control contexts, where GNN predictions inform decision-making algorithms such as traffic admission control. This integration could enable real-time, adaptive network management strategies that dynamically respond to changing network conditions.

- **Achieve real-world validation:** Validation and testing of the models on real-world testbeds to test the models' reliability in practical deployments. Such validation is crucial to ensure that the models can perform well under real network conditions, as simulations might not capture all the complexities and variabilities that simulations might not capture.
- **Achieve model explainability:** Investigate the explainability of GNN models to ensure transparency and interpretability in decision-making processes. Understanding how GNNs obtain their predictions can help network operators trust these algorithms and facilitate the integration of these models into production networks.
- **Perform security considerations:** Exploration of the security threats in GNN applied to network management, to ensure a robust protection against potential vulnerabilities. Ensuring that GNN models are secure is critical to maintain the integrity and reliability of network management systems.

The research presented in this thesis represents a step forward in the use of GNN models for network slicing and B5G network management. The proposed GAIN and GNNetSlice models have demonstrated high accuracy and scalability, providing practical solutions for predicting network KPIs in a near-real-time manner. As B5G networks continue to evolve, and with the future transition to 6G networks, the contributions of this thesis can serve as a foundation for more advanced and efficient network management solutions. Further research in the areas of real-world validation, hybrid datasets and testbeds, and advanced GNN architectures will be essential for achieving the full potential of GNN-based network management.

Bibliography

[1] 3GPP. *System Architecture for the 5G System (5GS)*. 3GPP TS 23.501 version 16.7.0 Release 16. 3rd Generation Partnership Project (3GPP), 2020. URL: https://www.3gpp.org/ftp/Specs/archive/23_series/23.501/23501-g70.zip.

[2] 3GPP. *TS 23.501*. URL: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3144.

[3] 5GPPP. *5GPPP Architecture Working Group View on 5G Architecture*. 2021. DOI: 10.5281/zenodo.5155657.

[4] 5GPPP. *AI and ML – Enablers for Beyond 5G Networks*. 2021. DOI: 10.5281/zenodo.4299895.

[5] Ali R. Abdellah et al. "Machine learning algorithm for delay prediction in IoT and tactile internet". In: *Future Internet* (12 Dec. 2021). ISSN: 19995903. DOI: 10.3390/fi13120304.

[6] Hamed Ahmadi et al. "Networked Twins and Twins of Networks: An Overview on the Relationship Between Digital Twins and 6G". In: *IEEE Communications Standards Magazine* 5.4 (2021), pp. 154–160. DOI: 10.1109/MCOMSTD.0001.2000041.

[7] Elif Ak et al. "T6CONF: Digital Twin Networking Framework for IPv6-Enabled Net-Zero Smart Cities". In: *IEEE Communications Magazine* 61.3 (2023), pp. 36–42. DOI: 10.1109/MCOM.003.2200315.

[8] Paul Almasan et al. *Digital Twin Network: Opportunities and Challenges*. 2022. DOI: http://dx.doi.org/10.48550/arXiv.2201.01144. eprint: 2201.01144.

[9] Alia Asheralieva, Dusit Niyato, and Yoshikazu Miyanaga. "Efficient Dynamic Distributed Resource Slicing in 6G Multi-Access Edge Computing Networks with Online ADMM and Message Passing Graph Neural Networks". In: *IEEE Transactions on Mobile Computing* (2023). ISSN: 15580660. DOI: 10.1109/TMC.2023.3262514.

[10] Mohsen Attaran and Bilge Gokhan Celik. "Digital Twin: Benefits, use cases, challenges, and opportunities". In: *Decision Analytics Journal* 6 (2023), p. 100165. ISSN: 2772-6622. DOI: https://doi.org/10.1016/j.dajour.2023.100165.

[11] Peter W. Battaglia et al. "Relational inductive biases, deep learning, and graph networks". In: *CoRR* (2018). DOI: http://dx.doi.org/10.48550/arXiv.1806.01261.

[12] Dario Bega et al. "AZTEC: Anticipatory Capacity Allocation for Zero-Touch Network Slicing". In: *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. 2020, pp. 794–803. DOI: 10.1109/INFOCOM41043.2020.9155299.

[13] Dario Bega et al. "DeepCog: Cognitive Network Management in Sliced 5G Networks with Deep Learning". In: *Proceedings - IEEE INFOCOM* (2019). ISSN: 0743166X. DOI: 10.1109/INFOCOM.2019.8737488.

[14] Irwan Bello et al. "Neural Combinatorial Optimization with Reinforcement Learning". In: *CoRR* (2016). DOI: https://doi.org/10.48550/arXiv.1611.09940. eprint: 1611.09940.

[15] BNN. *DataNet API Documentation*. [Online; accessed 31-01-2022]. 2021. URL: https://github.com/BNN-UPC/datanetAPI/tree/challenge2021.

[16] BNN. *Dataset description*. [Online; accessed 26-10-2022]. 2021. URL: https://bnn.upc.edu/challenge/gnnet2021/dataset/.

[17] BNN. *RouteNet - Graph Neural Networking challenge*. [Online; accessed 31-01-2022]. 2021. URL: https://github.com/BNN-UPC/GNNetworkingChallenge/tree/2021_Routenet_TF.

[18] BNN-UPC. *NetworkModelingDatasets - datasets v4*. https://github.com/BNN-UPC/NetworkModelingDatasets/tree/master/datasets_v4. [Accessed 17-Jun-2023].

[19] Kaylani Bochie et al. "A survey on deep learning for challenged networks: Applications and trends". In: *Journal of Network and Computer Applications* 194 (2021), p. 103213. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2021.103213.

[20] Tomas Borovicka et al. "Selecting Representative Data Sets". In: *Advances in Data Mining Knowledge Discovery and Applications*. Ed. by Adem Karahoca. Rijeka: IntechOpen, 2012. Chap. 2. DOI: 10.5772/50787.

[21] Facundo Bre, Juan Gimenez, and Víctor Fachinotti. "Prediction of wind pressure coefficients on building surfaces using Artificial Neural Networks". In: *Energy and Buildings* 158 (Nov. 2017). DOI: 10.1016/j.enbuild.2017.11.045.

[22] Beata Bylina and Jarosław Bylina. "Using Markov chains for modelling networks". In: *Annales UMCS Informatica AI* (2005), pp. 27–34. URL: http://www.annales.umcs.lublin.pl/.

[23] Chia Yu Chang et al. "Performance Isolation for Network Slices in Industry 4.0: The 5Growth Approach". In: *IEEE Access* (2021), pp. 166990–167003. ISSN: 21693536. DOI: 10.1109/ACCESS.2021.3135827.

[24] Florin Ciucu and Jens Schmitt. "Perspectives on Network Calculus - No Free Lunch, but Still Good Value". In: *Computer Communication Review* (Oct. 2012). DOI: 10.1145/2342356.2342426.

[25] *SOFGNN*. Concordia University and Computer Research Institute of Montreal. [Online; accessed 26-01-2022]. 2021. URL: https://github.com/ITU-AI-ML-in-5G-Challenge/ITU-ML5G-PS-001-SOFGNN-Graph-Neural-Networking-Challenge.

[26] Guozhu Dong and Huan Liu. *Feature Engineering for Machine Learning and Data Analytics*. 1st. USA: CRC Press, Inc., 2018. ISBN: 1138744387. DOI: https://doi.org/10.1201/9781315181080.

[27] Tianjian Dong et al. "Intelligent Joint Network Slicing and Routing via GCN-Powered Multi-Task Deep Reinforcement Learning". In: *IEEE Transactions on Cognitive Communications and Networking* 8 (2 June 2022), pp. 1269–1286. ISSN: 23327731. DOI: 10.1109/TCCN.2021.3136221.

[28] David Duvenaud et al. "Convolutional Networks on Graphs for Learning Molecular Fingerprints". In: *CoRR* (2015). DOI: https://doi.org/10.48550/arXiv.1509.09292.

[29] *PARANA. Improved GNN Generalization to Larger 5G Networks By Fine-Tuning Predictions From Queueing Theory*. Federal University of São Paulo. [Online; accessed 26-01-2022]. 2021. URL: https://github.com/ITU-AI-ML-in-5G-Challenge/ITU-ML5G-PS-001-PARANA.

[30] A. Fernandez. "On the isolation of several work-conserving scheduling policies". In: *Proceedings Eight International Conference on Computer Communications and Networks (Cat. No.99EX370)*. 1999, pp. 188–192. DOI: 10.1109/ICCCN.1999.805515.

[31] Miquel Ferriol-Galmés et al. "Building a Digital Twin for network optimization using Graph Neural Networks". In: *Computer Networks* 217 (2022), p. 109329. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2022.109329.

[32] Miquel Ferriol-Galmés et al. "RouteNet-Erlang: A Graph Neural Network for Network Performance Evaluation". In: *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*. 2022, pp. 2018–2027. DOI: 10.1109/INFOCOM48880.2022.9796944.

[33] Miquel Ferriol-Galmés et al. *RouteNet-Fermi: Network Modeling with Graph Neural Networks*. 2022. DOI: https://doi.org/10.48550/arXiv.2212.12070. eprint: 2212.12070 (cs.NI).

[34] Matthias Fey and Jan E. Lenssen. "Fast Graph Representation Learning with PyTorch Geometric". In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*. 2019. DOI: https://doi.org/10.48550/arXiv.1903.02428.

[35] Miquel Ferriol Galmés et al. "Scaling Graph-based Deep Learning models to larger networks". In: *CoRR* (2021). DOI: https://doi.org/10.48550/arXiv.1903.02428.

[36] Daniele Gammelli et al. “Graph Neural Network Reinforcement Learning for Autonomous Mobility-on-Demand Systems”. English. In: *Proceedings of the 60th IEEE Conference on Decision and Control, CDC 2021*. United States: Institute of Electrical and Electronics Engineers Inc., 2021, pp. 2996–3003. DOI: 10.1109/CDC45484.2021.9683135.

[37] Zihang Gao. “5G Traffic Prediction Based on Deep Learning”. In: *Computational intelligence and neuroscience* (2022), p. 3174530. ISSN: 16875273. DOI: 10.1155/2022/3174530.

[38] Zhun Ge, Jiacheng Hou, and Amiya Nayak. “GNN-based End-to-end Delay Prediction in Software Defined Networking”. In: *2022 18th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. 2022, pp. 372–378. DOI: 10.1109/DCOSS54816.2022.00066.

[39] Justin Gilmer et al. “Neural Message Passing for Quantum Chemistry”. In: *CoRR* (2017). DOI: https://doi.org/10.48550/arXiv.1704.01212.

[40] GAIN. *GAIN 5 GitHub code*. Girona-Antwerp Intelligence for Networking. [Online; accessed 10-10-2022]. 2022. URL: https://github.com/mfarreras/gain-gnn.

[41] Liyu Gong and Qiang Cheng. *Exploiting Edge Features in Graph Neural Networks*. 2019. arXiv: 1809.02709 [cs.LG].

[42] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[43] Daniele Grattarola and Cesare Alippi. “Graph Neural Networks in TensorFlow and Keras with Spektral [Application Notes]”. In: *Comp. Intell. Mag.* 16.1 (Feb. 2021), pp. 99–106. ISSN: 1556-603X. DOI: 10.1109/MCI.2020.3039072. URL: https://doi.org/10.1109/MCI.2020.3039072.

[44] Wanqing Guan et al. “A Service-Oriented Deployment Policy of End-to-End Network Slicing Based on Complex Network Theory”. In: *IEEE Access* PP (Apr. 2018), pp. 1–1. DOI: 10.1109/ACCESS.2018.2822398.

[45] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. “Exploring Network Structure, Dynamics, and Function using NetworkX”. In: *Proceedings of the 7th Python in Science Conference*. Ed. by Gaël Varoquaux, Travis Vaught, and Jarrod Millman. Pasadena, CA USA, 2008, pp. 11–15. DOI: http://dx.doi.org/10.25080/TCWV9851.

[46] William L. Hamilton, Rex Ying, and Jure Leskovec. “Inductive Representation Learning on Large Graphs”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 1025–1035. ISBN: 9781510860964. DOI: https://doi.org/10.48550/arXiv.1706.02216.

[47] Dan Hendrycks and Thomas G. Dietterich. "Benchmarking Neural Network Robustness to Common Corruptions and Perturbations". In: *CoRR* (2019). DOI: `https://doi.org/10.48550/arXiv.1903.12261`.

[48] Weihua Hu et al. "Open graph benchmark: Datasets for machine learning on graphs". In: *Advances in neural information processing systems* (2020), pp. 22118–22133. DOI: `https://doi.org/10.48550/arXiv.2005.00687`.

[49] ITU. *AI for Global Summit. ITU AI/ML in 5G Challenge: Graph Neural Networking Challenge 2021*. [Accessed 19-01-2022]. URL: `https://aiforgood.itu.int/about/aiml-in-5g-challenge/`.

[50] Jure Leskovec. *CS224W: Machine Learning with Graphs*. Stanford University, Oct. 12, 2021. URL: `http://cs224w.stanford.edu` (visited on 11/17/2022).

[51] Latif U. Khan et al. "Digital-Twin-Enabled 6G: Vision, Architectural Trends, and Future Directions". In: *IEEE Communications Magazine* 60.1 (2022), pp. 74–80. DOI: `10.1109/MCOM.001.21143`.

[52] Latif U. Khan et al. "Federated Learning for Digital Twin-Based Vehicular Networks: Architecture and Challenges". In: *IEEE Wireless Communications* (2023), pp. 1–8. DOI: `10.1109/MWC.012.2200373`.

[53] Ali Safari Khatouni, Francesca Soro, and Danilo Giordano. "A Machine Learning Application for Latency Prediction in Operational 4G Networks". In: *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. 2019, pp. 71–74. URL: `https://ieeexplore.ieee.org/document/8717807`.

[54] Meejoung Kim. "Analysis of feasible region in network slicing: Markov chain approach". In: *Computer Communications* 212 (2023), pp. 420–431. ISSN: 0140-3664. DOI: `https://doi.org/10.1016/j.comcom.2023.08.014`.

[55] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *Proceedings of the 5th International Conference on Learning Representations* (ICLR). ICLR '17. Palais des Congrès Neptune, Toulon, France, 2017. URL: `https://openreview.net/forum?id=SJU4ayYgl`.

[56] Bruno Klaus, Aquino Afonso, and Lilian Berton. "QT-RouteNet: Improved GNN generalization to larger 5G networks by fine-tuning predictions from queuing theory". In: *ITU Journal on Future and Evolving Technologies* (2022). URL: `https://www.itu.int/pub/S-JNL-VOL3.ISSUE2-2022-A12`.

[57] S. Knight et al. "The Internet Topology Zoo". In: *Selected Areas in Communications, IEEE Journal on* 29.9 (Oct. 2011), pp. 1765–1775. ISSN: 0733-8716. DOI: `10.1109/JSAC.2011.111002`.

[58] Max Kuhn and Kjell Johnson. *Applied Predictive Modeling*. Springer, 2013. ISBN: 978-1-4614-6848-6. URL: `https://doi.org/10.1007/978-1-4614-6849-3`.

[59] Arash Habibi Lashkari et al. “Characterization of tor traffic using time based features”. In: *ICISSP 2017 - Proceedings of the 3rd International Conference on Information Systems Security and Privacy*. Vol. 2017-January. SciTePress, 2017, pp. 253–262. ISBN: 9789897582097. DOI: 10.5220/0006105602530262.

[60] Xin Li et al. “Network Slicing for 5G: Challenges and Opportunities”. In: *IEEE Internet Computing* (Aug. 2018), pp. 1–1. DOI: 10.1109/MIC.2018.326150452.

[61] Deep Graph Library. *Heterogeneous Graphs*. https://docs.dgl.ai/en/latest/guide/graph-heterogeneous.html. [Accessed 12-Jul-2024].

[62] Jorg Liebeherr, Yashar Ghiassi-Farrokhfal, and Almut Burchard. “On the Impact of Link Scheduling on End-to-End Delays in Large Networks”. In: *IEEE Journal on Selected Areas in Communications* (2011), pp. 1009–1020. DOI: 10.1109/JSAC.2011.110511.

[63] Lu Lu. “Dying ReLU and Initialization: Theory and Numerical Examples”. In: *Communications in Computational Physics* 28.5 (June 2020), pp. 1671–1706. DOI: 10.4208/cicp.oa-2020-0165. URL: https://doi.org/10.4208%2Fcicp.oa-2020-0165.

[64] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[65] B. Martini, M. Gharbaoui, and P. Castoldi. “Intent-based network slicing for SDN vertical services with assurance: Context, design and preliminary experiments”. In: *Future Generation Computer Systems* 142 (May 2023), pp. 101–116. ISSN: 0167739X. DOI: 10.1016/j.future.2022.12.033.

[66] Sunil Kumar Maurya, Xin Liu, and Tsuyoshi Murata. “Feature selection: Key to enhance node classification with graph neural networks”. In: *CAAI Transactions on Intelligence Technology* 8.1 (2023), pp. 14–28. DOI: https://doi.org/10.1049/cit2.12166.

[67] A. Minokuchi and S. Isobe. “5G Core Network Standardization Trends.” In: *NTT DOCOMO Technical Journal, Technology Reports, Vol 19-3*. 2020, pp. 48–54. URL: https://www.docomo.ne.jp/english/binary/pdf/corporate/technology/rd/technical_journal/bn/vol19_3/vol19_3_006en.pdf.

[68] Ryan L. Murphy et al. “Relational Pooling for Graph Representations”. In: *CoRR* (2019). DOI: https://doi.org/10.48550/arXiv.1903.02541.

[69] Nicolò Navarin and Duc Van Tran. “Universal readout for graph convolutional neural networks”. In: *International Joint Conference on Neural Networks (IJCNN)*. 2019. DOI: 10.1109/IJCNN.2019.8852103. URL: https://ieeexplore.ieee.org/document/8852103.

[70] N Alliance - NGMN 5G P. "Description of network slicing concept". In: (2016). URL: https://www.ngmn.org/wp-content/uploads/Publications/2016/161010_NGMN_Network_Slicing_framework_v1.0.8.pdf.

[71] Dimitrios Pliatsios et al. "Realizing 5G vision through Cloud RAN: technologies, challenges, and trends". In: *EURASIP Journal on Wireless Communications and Networking* 2018 (May 2018). DOI: 10.1186/s13638-018-1142-1.

[72] Filippo Poltronieri et al. "Value is King: The MECForge Deep Reinforcement Learning Solution for Resource Management in 5G and Beyond". In: *J. Netw. Syst. Manage.* 30.4 (Oct. 2022). ISSN: 1064-7570. DOI: 10.1007/s10922-022-09672-6. URL: https://doi.org/10.1007/s10922-022-09672-6.

[73] Aiko Pras, Juergen Schonwalder, and Mark Burgess. "Key research challenges in network management". In: *IEEE Communications Magazine* (2007). URL: https://ieeexplore.ieee.org/abstract/document/4342832/.

[74] Junfei Qiu et al. "A survey of machine learning for big data processing". In: *Eurasip Journal on Advances in Signal Processing* 2016 (1 Dec. 2016). ISSN: 16876180. DOI: 10.1186/s13634-016-0355-x.

[75] JA Rahim, R Nordin, and OA Amodu. "Open-Source Software Defined Networking Controllers: State-of-the-Art, Challenges and Solutions for Future Network Providers". In: *Computers, Materials and Continua* (2024). URL: https://cdn.techscience.cn/files/cmc/2024/TSP_CMC-80-1/TSP_CMC_47009/TSP_CMC_47009.pdf.

[76] Anouar Rkhami et al. "MonGNN: A neuroevolutionary-based solution for 5G network slices monitoring". In: *Proceedings - Conference on Local Computer Networks, LCN* (2021), pp. 185–192. DOI: 10.1109/LCN52139.2021.9524880.

[77] Krzysztof Rusek. *GitHub - Queuinx: A library for performance evaluation in Jax*. https://github.com/krzysztofrusek/queuinx. [Accessed 17-Jun-2023].

[78] Krzysztof Rusek et al. "RouteNet: Leveraging Graph Neural Networks for Network Modeling and Optimization in SDN". In: *IEEE Journal on Selected Areas in Communications* (2020), pp. 2260–2270. ISSN: 15580008. DOI: 10.1109/JSAC.2020.3000405. eprint: 1910.01508.

[79] Kamal Saluja. "Comparative Analysis of Network Simulators-A Literature Review". In: *International Journal of Research in Computer Applications and Information Technology* 3 (6 2015), pp. 32–39. ISSN: 2348-0009. URL: www.iaster.com.

[80] Ryoma Sato. "A Survey on The Expressive Power of Graph Neural Networks". In: *CoRR* (2020). DOI: https://doi.org/10.48550/arXiv.2003.04078. eprint: 2003.04078.

[81] Franco Scarselli et al. “The graph neural network model”. In: *IEEE Transactions on Neural Networks* (2009), pp. 61–80. ISSN: 10459227. DOI: 10.1109/TNN.2008.2005605.

[82] Michael Schlichtkrull et al. *Modeling Relational Data with Graph Convolutional Networks*. 2017. arXiv: 1703.06103 [stat.ML].

[83] S. Shalev-Shwartz and S. Ben-David. *Understanding machine learning: From theory to algorithms*. CS Huji, 2014. URL: https://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning/understanding-machine-learning-theory-algorithms.pdf.

[84] S Singh and RK Jha. “A survey on software-defined networking (SDN): Architecture for next-generation network”. In: *Journal of Network and Systems Management* (2017). DOI: 10.1007/s10922-016-9393-9. URL: https://link.springer.com/article/10.1007/s10922-016-9393-9.

[85] Paola Soto et al. “ATARI: A graph convolutional neural network approach for performance prediction in next-generation WLANs”. In: *Sensors* (2021). DOI: https://doi.org/10.3390/s21134321.

[86] Prashant Subedi et al. “Network slicing: a next generation 5G perspective”. In: *Eurasip Journal on Wireless Communications and Networking* 2021 (1 Dec. 2021). ISSN: 16871499. DOI: 10.1186/s13638-021-01983-7.

[87] GraphML Team. *The GraphML File Format*. 2002. URL: http://graphml.graphdrawing.org/ (visited on 03/24/2014).

[88] N Kumar V Thirupathi CH Sandeep. “A Comprehensive Review on SDN Architecture, Applications, and Major Benefits of SDN”. In: *International Journal of Advances in Science, Engineering and Technology* (2019). URL: https://www.researchgate.net/publication/339771419_A_COMPREHENSIVE_REVIEW_ON_SDN_ARCHITECTURE_APPLICATIONS_AND_MAJOR_BENIFITS_OF_SDN.

[89] Petar Veličković et al. “Graph Attention Networks”. In: *6th International Conference on Learning Representations* (2017). DOI: https://doi.org/10.48550/arXiv.1710.10903.

[90] Haozhe Wang et al. “A Graph Neural Network-based Digital Twin for Network Slicing Management”. In: *IEEE Transactions on Industrial Informatics* (2 2020), pp. 1367–1376. ISSN: 19410050. DOI: 10.1109/TII.2020.3047843.

[91] Minjie Wang et al. “Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks”. In: *arXiv preprint arXiv:1909.01315* (2019).

[92] E. Weingartner, H. vom Lehn, and K. Wehrle. “A Performance Comparison of Recent Network Simulators”. In: *2009 IEEE International Conference on Communications*. 2009, pp. 1–5. DOI: 10.1109/ICC.2009.5198657.

[93] Yiwen Wu, Ke Zhang, and Yan Zhang. "Digital Twin Networks: A Survey". In: *IEEE Internet of Things Journal* 8.18 (2021), pp. 13789–13804. DOI: 10.1109/JIOT.2021.3079510.

[94] Keyulu Xu et al. "How Powerful are Graph Neural Networks?" In: *CoRR* (2018). DOI: https://doi.org/10.48550/arXiv.1810.00826.

[95] Xiaolong Xu et al. "Service Migration Across Edge Devices in 6G-Enabled Internet of Vehicles Networks". In: *IEEE Internet of Things Journal* 9.3 (2022), pp. 1930–1937. DOI: 10.1109/JIOT.2021.3089204.

[96] Zhiyuan Xu et al. "Experience-driven networking: A deep reinforcement learning based approach". In: *IEEE INFOCOM 2018-IEEE*. IEEE. 2018. DOI: https://doi.org/10.48550/arXiv.1801.05757.

[97] G Hinton Y LeCun Y Bengio. "Deep learning". In: *Nature* (2015). DOI: 10.1038/nature14539. URL: https://www.nature.com/articles/nature14539.

[98] Qingqing Yang et al. "DeepQueueNet: towards scalable and generalized network performance estimation with packet-level visibility". In: *Proceedings of the ACM SIGCOMM 2022 Conference*. SIGCOMM '22. Amsterdam, Netherlands: Association for Computing Machinery, 2022, pp. 441–457. ISBN: 9781450394208. DOI: 10.1145/3544216.3544248. URL: https://doi.org/10.1145/3544216.3544248.

[99] Tong Yu and Hong Zhu. *Hyper-Parameter Optimization: A Review of Algorithms and Applications*. 2020. arXiv: 2003.05689 [cs.LG].

[100] Anis Zarrad and Izzat Alsmadi. "Evaluating network test scenarios for network simulators systems". In: *International Journal of Distributed Sensor Networks* 13 (10 Oct. 2017), pp. 1–17. ISSN: 15501477. DOI: 10.1177/1550147717738216.

[101] Cheng Zhou et al. *Digital Twin Network: Concepts and Reference Architecture*. Internet draft. Internet Engineering Task Force, July 2022. 29 pp. URL: https://datatracker.ietf.org/doc/draft-irtf-nmrg-network-digital-twin-arch/01/.