Computer Communications 232 (2025) 108044

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

GNNetSlice: A GNN-based performance model to support network slicing in B5G networks

Miquel Farreras [a,*], Jordi Paillissé [a,b], Lluís Fàbrega [a], Pere Vilà [a]

[a] Institute of Informatics and Applications, Universitat de Girona, Girona, Spain
[b] UPC-BarcelonaTech, Barcelona, Spain

ARTICLE INFO

Dataset link: https://zenodo.org/records/10610616

Keywords:
Network modeling
Network slicing
Graph Neural Networks

ABSTRACT

Network slicing is gaining traction in Fifth Generation (5G) deployments and Beyond 5G (B5G) designs. In a nutshell, network slicing virtualizes a single physical network into multiple virtual networks or slices, so that each slice provides a desired network performance to the set of traffic flows (source–destination pairs) mapped to it. The network performance, defined by specific Quality of Service (QoS) parameters (latency, jitter and losses), is tailored to different use cases, such as manufacturing, automotive or smart cities. A network controller determines whether a new slice request can be safely granted without degrading the performance of existing slices, and therefore fast and accurate models are needed to efficiently allocate network resources to slices. Although there is a large body of work of network slicing modeling and resource allocation in the Radio Access Network (RAN), there are few works that deal with the implementation and modeling of network slicing in the core and transport network.

In this paper, we present GNNetSlice, a model that predicts the performance of a given configuration of network slices and traffic requirements in the core and transport network. The model is built leveraging Graph Neural Networks (GNNs), a kind of Neural Network specifically designed to deal with data structured as graphs. We have chosen a data-driven approach instead of classical modeling techniques, such as Queuing Theory or packet-level simulations due to their balance between prediction speed and accuracy. We detail the structure of GNNetSlice, the dataset used for training, and show how our model can accurately predict the delay, jitter and losses of a wide range of scenarios, achieving a Symmetric Mean Average Percentage Error (SMAPE) of 5.22%, 1.95% and 2.04%, respectively.

1. Introduction

In recent years, the surge of interest in Beyond 5G (B5G) networks has been fueled by their significant enhancements in bandwidth, latency, and innovative capabilities, promising benefits for both consumers and operators. These improvements aim to support the growing demands of connected devices and applications requiring high-speed, low-latency connectivity. Despite the considerable progress in this field, the implementation of these networks requires further development to fully unlock their potential.

Efficient management of network resources is one of the key requirements to successfully build next-generation networks. Resource management is still under discussion and development in B5G networks, particularly in the context of network slicing [1,2], which remains a subject of ongoing discussion and development. Network slicing involves the virtualization of physical infrastructure, allowing the network to be divided into multiple isolated virtual networks. Each slice can be tailored with different Quality of Service (QoS) and traffic parameters such as latency and bandwidth, enabling a wide range of use cases like Enhanced Mobile Broadband (eMBB), massive Internet of Things (mIoT), or Ultra-Reliable Low-Latency Communication (URLLC), as defined by 3GPP (TS-23.501 [3]).

However, effective resource management for diverse network slices is computationally expensive due to three main challenges. First, the increase in traffic volume: 5G/B5G networks are expected to handle more traffic than previous generations due to the incorporation of massive amounts of connected devices (e.g. smart meters, remote tags, etc.), as well as new services (e.g. AR/VR). Second, the variety of SLA demands of the different network slices and types of RAN networks connected to the core, that range from delay tolerant and low throughput slices to limited delay and high bandwidth slices. Third, a fast prediction time, in order to be able to dynamically adjust the network configuration parameters as network conditions evolve over time.

* Corresponding author.
E-mail address: miquel.farreras@udg.edu (M. Farreras).

https://doi.org/10.1016/j.comcom.2025.108044
Received 21 May 2024; Received in revised form 19 November 2024; Accepted 1 January 2025
Available online 11 January 2025
0140-3664/© 2025 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

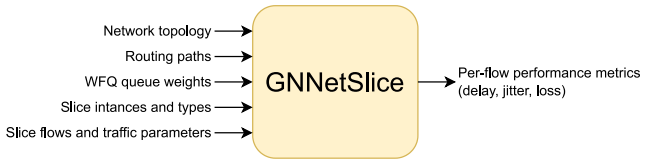

Fig. 1. Black-box representation of the GNNetSlice model.

To overcome these obstacles, resource management in B5G networks requires sophisticated prediction algorithms in combination with decision-making processes. In order to guarantee network Key Performance Indicators (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 Service Level Agreements (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.

In this context, several tools can estimate network performance, such as Queuing Theory or Discrete Event Simulators. Traditional network modeling is based on Queuing Theory, that leverages mathematical analysis of the network behavior, but it relies on simplifications that often do not reflect the complexity and dynamism of current network models [4]. In contrast, network simulators offer highly accurate results at the packet level. However, their processing time and usage of computer resources grows exponentially with the network size and parameter configuration, especially in case of frequently changing network conditions. Consequently, existing networks are challenging to analyze, monitor, and manage in near real-time with current solutions [5].

Numerous studies have investigated the application of Artificial Intelligence (AI) and Machine Learning (ML) techniques to B5G resource management. As depicted in Section 3, data-driven approaches such as Deep Learning, Reinforcement Learning, and Graph Neural Networks (GNNs) have demonstrated promising results in optimizing resource allocation, fault detection, and traffic management and prediction in B5G networks [6]. By leveraging the massive amounts of data generated by B5G networks, data-driven approaches can learn and adapt to changing network conditions and optimize resource usage. Their fast computation speed, along with the capacity to adapt to different scenarios, offer an interesting trade-off between Queuing Theory and network simulators.

In this paper, we present GNNetSlice, a novel approach to build a model to predict the network Key Performance Indicators (KPIs), which can be used to determine flow performance when allocating network slices to a given physical network. This usage is in line with the concept of a Digital Twin (DT) [7], a virtual replica of a physical network, that mimics one or more characteristics of the physical network. GNNetSlice predicts performance metrics (delay, jitter, losses) for network slices in core and transport networks. Fig. 1 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.

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 a DT [8].

Furthermore, we describe our approach to create the dataset, analyze the network properties, build the model, train it, and obtain predictions. We evaluate GNNetSlice and compare it with recent state-of-the-art GNN approaches, as well as the results of a packet-level simulator.

Our contributions 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.

The rest of the paper is structured as follows. Section 3 reviews methods for predicting network KPIs. Section 2 introduces the GNN and the network slicing concepts. The dataset generated for testing the models is outlined in Section 5, while GNNetSlice, our GNN-based solution for predicting KPIs of network slices, is presented in Section 6. The evaluation of the best GNN model is discussed in Section 7, and we conclude the paper in Section 8.

2. Background

2.1. Graph neural networks

A graph representation can better capture the relationships between nodes in a network, providing a direct mapping between the network topology and the graph representation. This allows the GNN model to predict multiple unseen topologies.

The expressive power of GNNs [9] allows them to model graphs with higher accuracy, offering advantages in accurate and fast network analysis [10]. GNNs have demonstrated success in solving combinatorial optimization problems [11] and can achieve relational reasoning [12]. GNNs have emerged as a robust approach to address various problems associated with graph-structured data, such as social networks, molecular structures, and computer networks [13,14]. Leveraging interactions between nodes and edges in a graph, GNNs excel in making predictions, as well as performing regression and classification tasks [15]. At the core of a GNN lies the concept of message-passing, where information exchange between neighboring nodes is employed to update the features of the target node.

In a GNN, a graph is represented as a set of nodes and edges, with each node and edge including associated features. The architecture consists of multiple layers, and at each layer, information is propagated through the graph, assimilating information from neighboring nodes and edges. The message-passing process involves three key steps: message transformation, aggregation, and update:

1. Message transformation functions dictate how nodes and edges generate messages.
2. Aggregation functions combine these messages.
3. Update functions integrate aggregated messages to update the features of the target node.

Finally, a readout function executes the classification of regression task of the GNN. Fig. 2 illustrates the iterative nature of the message passing process across an input graph. This figure shows the architecture of a simple GNN, where nodes with distinct colors incorporate their features into vectors. The primary goal is to predict values for the target node labeled as “A”. The message transformation phase is applied initially, followed by the aggregation function and, finally, an update to the values of the target node. In cases where a specific node’s attributes are missing, a pooling technique is employed to gather information from it [16], enhancing the overall information flow within the network.

GNNs can provide predictions at different levels of detail:

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

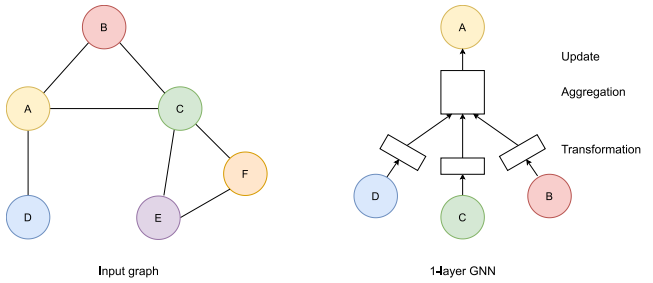

Fig. 2. Basic operation of a GNN, showing the input graph (left) and the equivalent 1-layer GNN (right) with the 3 key steps of message-passing for predicting node A's features [17].

2.2. Network slicing

Network slicing is commonly defined as a virtualization technology that allows dividing the resources of a physical network into multiple virtual networks or slices [18]. Each slice provides a desired network performance to the set of traffic flows (source–destination pairs) mapped to it, e.g., reduced latency or high bandwidth, depending on the use case. Its importance in B5G is strongly tied to the use cases it enables in different verticals, such as manufacturing, automotive, or smart cities. Moreover, there is a set of standardized slice types defined by 3GPP that aim to capture the majority of these use cases in 5G and B5G networks (3GPP TS-23.501 [3]). The slices are Enhanced Mobile Broadband (eMBB), Ultra-Reliable Low-Latency Communication (URLLC), and massive Internet of Things (mIoT). Those slices require basic functions: (i) ability to be dynamically created and destroyed, (ii) span end-to-end paths in the network, and (iii) be isolated between them.

These requirements make resource allocation essential to guarantee the QoS of each slice. A network controller determines whether a new slice request can be safely granted without degrading the performance of existing slices, and is in charge of determining which flows can be allocated to the network. There is a wide range of data models and interfaces that allow fine-grained control of network resources to create, modify, and delete network slices [19]. Due to the fact that slicing is implemented end-to-end (Radio Access Network (RAN), core, and transport network), most slicing architectures incorporate one or more controllers or orchestrators to coordinate the resource allocation.

In the data plane, network slicing is implemented with different mechanisms, combining resource reservation, traffic conditioning and queue scheduling algorithms. Packets are mapped to slices. Packets are mapped to slices using common tunneling and labeling protocols such as Ethernet Virtual Private Network (EVPN) or Multiprotocol Label Switching (MPLS), and routers are centrally controlled via Software Defined Networking (SDN) southbound protocols.

3. Related work

3.1. Traditional techniques for network performance prediction

In the literature, various techniques are explored to predict network performance, primarily relying on analytical modeling methods such as Queuing Theory [20], Markov chains [21,22], and similar approaches. However, despite these models often yield fast predictions, their accuracy degrades due to their dependence on unrealistic and static assumptions about packet arrival processes and real-world networks [23].

To obtain more realistic results, packet-level network simulators are commonly employed, thanks to their packet-level visibility and fine granularity. However, their computational complexity and execution times, especially with larger networks, cannot be used in scenarios that require near real-time predictions [24]. Nevertheless, these simulators generate highly accurate data that proves valuable for training different ML models, as demonstrated in [25].

3.2. 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 [26], 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 [27] 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 had a much better precision. Still, this was an initial approach to demonstrate the usability of the dataset.

Additionally, a DT approach in [28] predicts network traffic handling in a similar way to [26,29], but using Recurrent Neural Networks (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. Static and in-vehicle 5G data is used to train and predict using three different methods: Convolutional Neural Network (CNN), Gated Recurrent Unit (GRU), and LSTM, the last obtaining their best results.

RNNs are also employed in [30] to predict delays in 5G networks for the Internet of Things (IoT) and Tactile Internet. The model is trained in a simulated IoT system, collecting and later predicting the IoT traffic network delays.

3.3. 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 [31], where GNNs are used to enable reinforcement learning agents to adapt policies of autonomous mobility-on-demand. Another application in network slicing monitoring [32] 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 [33]. Finally, SDN end-to-end delay prediction in [34] 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, Deep Learning (DL), and GNNs, are summarized in Table 1. 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. While GNNs have emerged as excellent models for graph representation in networks, they face challenges due to a lack of datasets for model training, particularly in the field of network slicing. Our contributions address this gap by generating a network slicing dataset and developing a GNN model for predicting the KPIs of simulated network slices.

4. Scenario

The network consists of a core network and a transport network, with the RAN falling outside of the scope. We consider the following functions in the admission control block (Fig. 3): the Core Controller receives slice requests and decides if they can be allocated in the

Table 1
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
[26]	CNN	4D-CNN + MLPs	Real scenario	Private	N/A	N/A	Yes	No	No
[28]	RNN	SLSTM	Real data	On request	N/A	N/A	Yes	No	No
[30]	RNN	NARX	AnyLogic Simulator	Private	N/A	Yes	Yes	No	No
[27]	Classification models	SVM, DT, LR	Real data	Private	N/A	Yes	Yes	No	No
[31]	GNN	GCN + MLPs	Real data	Public	N/A	N/A	Yes	No	No
[32]	GNN	GAT + MLPs + genetic algorithm	Genetic Algorithm + greedy solution	Private	Yes	N/A	Yes	Yes	Yes
[34]	GNN	STGCN	OMNeT++ simulation	Private	Yes	No	Yes	No	No
[35]	GNN	MPNN + GRU	OPNET simulation	Private	N/A	Yes	Yes	No	Yes
[36]	GNN	GCN/MTDRL	Open dataset, Tor [37] + OpenDayLight scenario	Private	N/A	N/A	Yes	No	Yes
[38]	GNN	MPNN + GRU	OMNeT++ simulation	Public	Yes	No	Yes	No	No
GNNet-Slice	GNN	GRCN	OMNeT++ simulation	Public	Yes	Yes	Yes	No	Yes

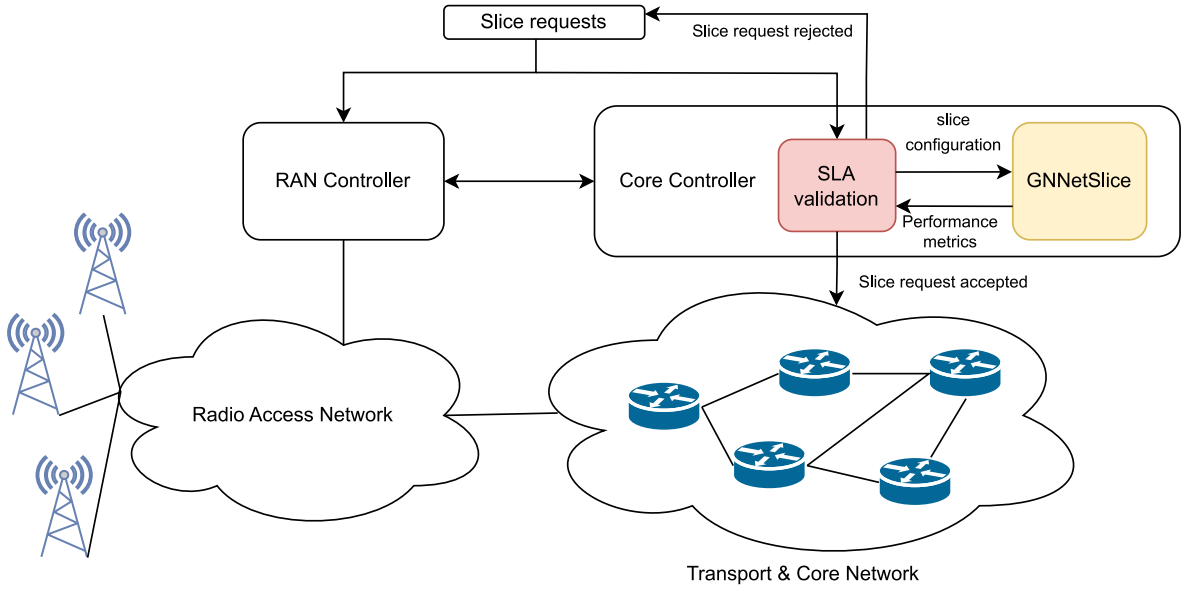

Fig. 3. Functional block diagram of the slice admission controller.

network. The slice admission module incorporates GNNetSlice, that acts as a Slice Performance Model, 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. The Core Controller syncs the configurations with the RAN Controller, to align the configurations for the changing network requirements.

We consider three distinct slice types, specifically those defined by the 3rd Generation Partnership Project (3GPP) [39]: eMBB, URLLC, and mIoT. Each slice type is designed for different use cases (Table 2). 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.

We assume that both the transport and core networks implement a packet labeling mechanism. Output links in the routers map these labeled packets to queues using a WFQ algorithm. WFQ queues facilitate the isolation of the slice instances, as the weight for each slice instance is determined during the admission step [40]. This isolation among queues guarantees the allocated bandwidth for each slice, and implies that excess traffic in one queue does not affect traffic in other queues; each queue has a guaranteed minimum portion of the link's capacity. If a slice experiences a reduction in traffic, the rest of the queues can benefit from the extra capacity. The weights of each WFQ queue are adjusted to allocate a given portion of link capacity to the slice. Algorithms are dynamically adjusted to meet the traffic requirements of each slice. Each WFQ queue serves a different slice, spanning from the originating RAN node(s) to the transport network's exit point.

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 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. 4, 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

Table 2
Slice types considered and flow characteristics [41,42].

Type of flow	Maximum delay	Maximum packet loss	Intended use	Bitrate	Packet size
eMBB	10 ms	10%	Broadcasting, media delivery, gaming, general usage	18 Mbps	Average: 6000 bits Minimum: 800 bits Maximum: 18000 bits
URLLC	0.5 ms	0.001%	High reliability, ultra-low latency, high availability	3.5 Mbps	Average: 800 bits Minimum: 256 bits Maximum: 1600 bits
mIoT	Up to 10s	1%	Long battery, low cost devices, extreme coverage	0.2 Mbps	Average: 160 bits Minimum: 320 bits Maximum: 480 bits

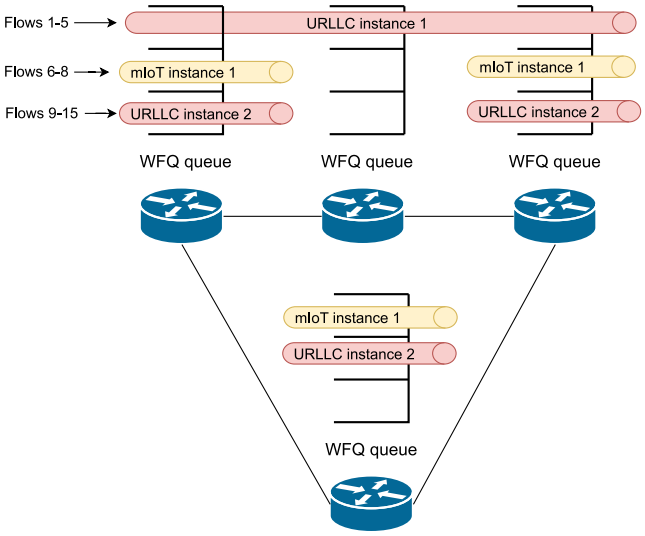

Fig. 4. Mapping of flows to slices and slices to WFQ queues across several paths.

of these instances, as the weight for each slice instance is determined during the admission step.

More details on the step-by-step methodology performed for generating the scenario and the dataset can be found in [43].

5. Methodology

In order to build the GNNetSlice GNN-based model for estimating slice performance, we adopt a data-driven approach, typical in ML workflows (Fig. 5). First, we generate a dataset using a network simulator, comprising pairs of `(scenario, performance)`, representing the model's inputs and outputs, respectively. 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.

Next, we train a GNN model 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 our 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.

5.1. Packet level simulations

The ground truth is generated using a packet-level simulator, a modified version of OMNeT++ [38]. The 130 network topologies are obtained from the Internet Topology Zoo [44], simulating a wide range of slicing configurations and traffic intensities for each topology. Traffic originates from different nodes connected just after the RAN and exits the network in a set of nodes selected as output nodes on the other end. A simplified example can be observed in Fig. 6, where multiple source nodes generate input traffic to the network and an exit node is placed at the right side. The destinations of the generated flows are common for each type of flow, having three different types of destination located at the outgoing port of the exit node (eMBB, mIoT, URLLC). The different simulation scenarios are generated using the following steps:

1. Retrieve a network topology from the Internet Topology Zoo [44].
2. Randomly allocate link capacities from a predefined range of 25 to 500 Mbps.
3. Designate at least two nodes as sources. A maximum of N/3 nodes should serve as source nodes, where N represents the total number of nodes in the graph.
4. Define the three destinations (eMBB, mIoT, URLLC). The remaining nodes are categorized as transport nodes.
5. Randomly set the number of slice instances between 1 and N.
6. Specify the slice type and traffic for each slice.
7. Assign one eMBB flow for each source node, while determining the others based on link 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 link capacity is complete in any link of the path.
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 bitrate ($V_{reserved}$) of the slices that traverse each link.
9. Eliminate any empty slices.
10. Execute the simulation using the network simulator.

The slices exhibit the following characteristics:

- Three different network slices types (eMBB, mIoT and 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 source nodes.
- Each slice has at least one assigned flow.
- At least one eMBB flow is included for each simulated source node, serving as the standard network usage. Other flows for each source node and slice are randomly assigned.

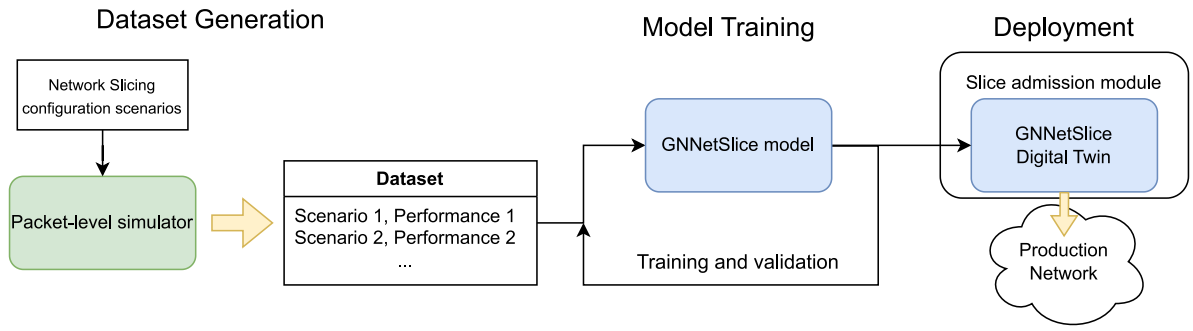

Fig. 5. Machine Learning workflow, from dataset generation to model deployment, for constructing the GNN-based Slice Performance model.

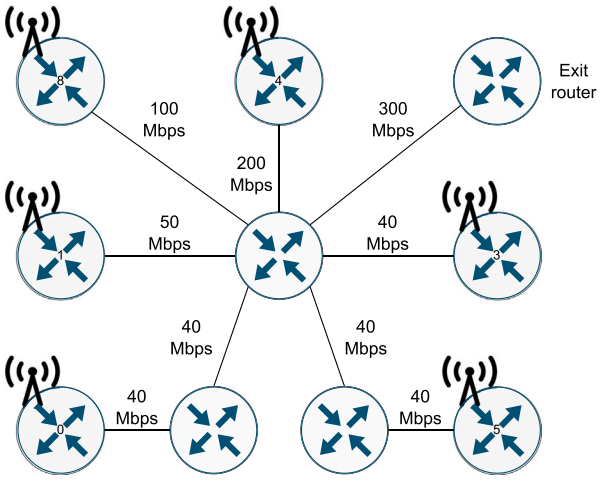

Fig. 6. An example of the Ai3 network topology: routers with antennas (the RAN part) would be the source nodes of the transport network while the exit router would be the destination node.

- Flows are created to fill link capacities, and reservations are made based on each flow's required average bit rate.
- The bit rate reservation for each slice is allocated on each output port using WFQs. This allows the implementation of resource isolation of each slice from the others, a key characteristic for network slices.
- Each slice instance contains different flows with the traffic characteristics outlined in Table 2.
- A standard proportion of 1 eMBB flow, 7 URLLC flows and 2 mIoT flows is used, derived from [41].

Finally, simulations allow different scenarios, ranging from under provisioning to overprovisioning, controlled by a tunable parameter $\delta \in [0, 1]$ that adjusts the amount of under and overprovisioning as shown in Eq. (1). $V_{reserved}$ represents the actual reserved bitrate in the simulation, while δ is the resource provisioning parameter, randomly assigned with a value $[0, 1]$ for flows within the same slice. V_{min} and V_{max} denote the minimum and maximum bitrate for each flow, respectively. This configuration allows for a flexible range from underprovisioning at V_{min} to overprovisioning at V_{max}.

In line with the methodology in [43], δ is used to expose the networks to different traffic intensities, allowing to observe how KPIs vary under different provisioning conditions. Specifically, the minimum and maximum bitrates are set based on the average flow bitrate, with underprovisioning defined as 10% below the average and overprovisioning as 20% above. This approach provides a simulation of traffic conditions across different scenarios and allows a comprehensive analysis of network performance.

$$V_{reserved} = V_{min} + \delta(V_{max} - V_{min}) \quad (1)$$

After generating the necessary files (routing, traffic, and topology) for simulating the scenarios, the OMNeT++ simulator is utilized to produce samples for the dataset.

5.2. Dataset

The used dataset from [43] comprises 7900 samples, each consisting of the following components:

1. Topology: includes links and nodes. The example topology in Fig. 6 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 2, 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, traffic reservation ($V_{reserved}$), 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.

In terms of individual samples, the node count ranges from 17 to 765 nodes. Simulation durations span between 1.4 h and 16 h, while the memory utilization varies from 161 MB to 27 GB.

Since the ability to generalize is critical in ML-based algorithms, we have performed an extensive analysis of the entire dataset, separated by links, flows, and queues for each slice type. This way, we can ensure that the algorithm is presented with a wide range of network scenarios, that include varying degrees of network load, packet loss or link bandwidth. The following subsections analyze the distribution of different flow, link and queue features, and more details can be found in [43].

Fig. 7 shows the values distribution of several flow features. 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 δ values are mostly uniform for each type of slice, while packet loss is lower in the URLLC slices.

We can see that the features 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.

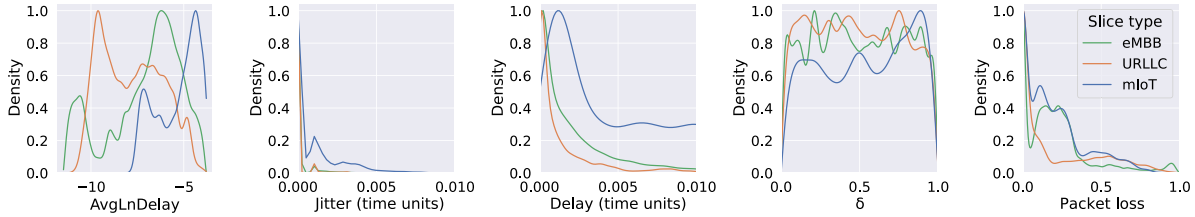

Fig. 7. Probability Density Function (PDF) of several flow features for each slice type.

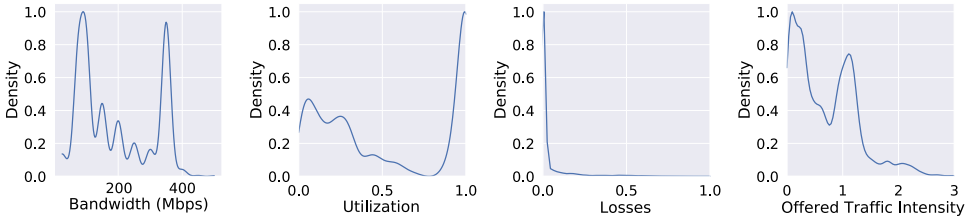

Fig. 8. Probability Density Function (PDF) of several link features.

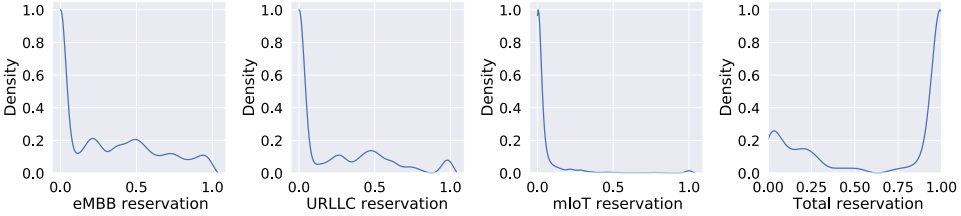

Fig. 9. Probability density function (PDF) of the reservations of the links.

Fig. 8 illustrates the values distribution of several link features, showing high utilization, as it was intended in the scenarios, low losses in most cases, varied bandwidth 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. (2). 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.

$$O_t = \frac{\sum_{f \in N_\ell} T_f}{C_\ell} \tag{2}$$

Fig. 9 shows the values distribution of link's reservations. The majority of links have a total reservation close to 1, meaning that the algorithm to fill the network as much as possible with reservations is working correctly. The percentage of reservations in each link is higher in the case of eMBB as expected, because this type of slices has higher bandwidth and the requirement of having at least one eMBB slice for each source node.

Finally, Fig. 10 shows several queue features values distribution. It can be seen that most values for the weight attribute of the WFQ queues are under 50%. The utilization of the queues is higher for the eMBB slices, while it is very low for the mIoT case. The losses are lower in the URLLC queues analyzed as also seen in the links, while the delay is also lower in the URLLC. The max. occupancy measured in number of packets is on average a little bit higher for the URLLC slices, but the total max. occupancy is reached more commonly by the mIoT slice types. Moreover, in some occasions the queues are full, which is the expected behavior in the situations where there are packet losses. The queues have a capacity of 31 packets, and it can be seen that the max. occupancy in number of packets has a peak between 25 and 31 packets.

6. GNNetSlice model

Our objective is to predict delay, jitter, and loss for the generated dataset. For constructing the GNNetSlice model we have used the Deep Graph Library (DGL) for PyTorch [46] is used, 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) [47]. 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 [48].
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 (Eq. (3)).
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.

$$N_x = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{3}$$

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. Relational Graph Convolutional Networks (RGCNs) [49], the type of model used for building GNNetSlice, are designed to process and update node information based on the relationships between nodes, making them suitable

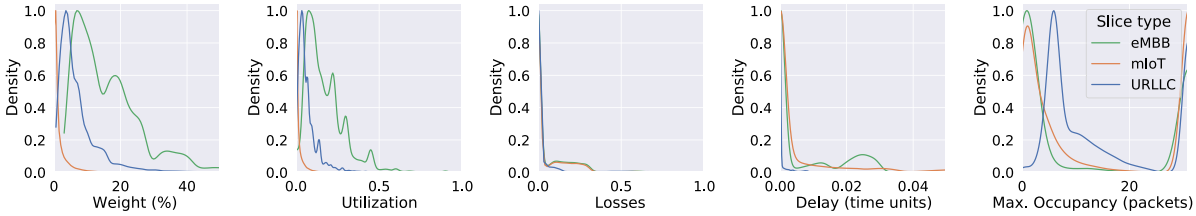

Fig. 10. Probability Density Function (PDF) of several queue features for each slice type.

for tasks like node classification and link prediction. However, RGCNs are not built to predict values directly associated with edges, such as link delay or utilization, because they do not have a way to learn or represent edge-specific information directly GNNs. To overcome this, we transform link attributes into nodes within the graph, allowing the RGCN to handle link properties indirectly by treating them as node attributes. This approach enables the model to work with heterogeneous graphs, to perform edge prediction tasks.

To address this problem, the input graph is reshaped by transforming links into nodes. This reshaping creates an input graph, allowing GNNs to indirectly predict link attributes as node attributes and bridging the gap between GNN capabilities and link attribute regression challenges [50]. More concretely, to predict flow features using link and queue features, we need 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. 11 we show an example of how the GNN input graph is generated from the original network topology graph. 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*.

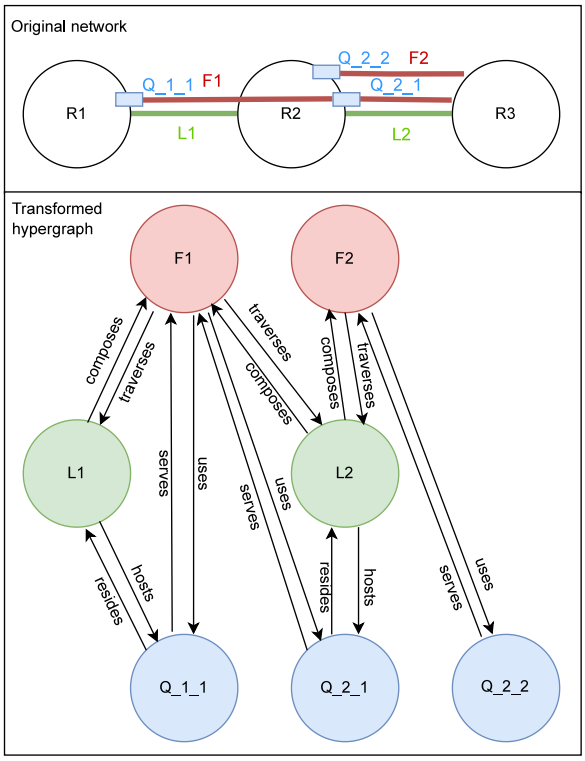

Fig. 11. Example of an original network topology graph (top) transformation to the input graph (bottom).

6.2. Algorithm

The GNNetSlice model uses a RGCN for predictions. This model supports multiple relation types in the input graph, a crucial functionality for defining relations between flows, queues, and links, enabling the usage of the designed input graph node types and features. RGCNs are chosen mainly because they can be used in scenarios involving multi-relational data, where edges have different types or relationships, as they are designed to incorporate heterogeneous edge types into the node update process. In contrast, Graph Isomorphism Networks (GINs) [51] excel at distinguishing graph structures, which could use the original network graph without modifications, but these are typically limited to homogeneous graphs. In our case, network slicing involves diverse node types that impact flow attributes differently. Thus, RGCN is chosen, as it can learn from these varied relations, providing predictions by modeling the diverse interactions between nodes.

GNNetSlice consists of two GraphConv layers [52], which are graph convolutional layers, each using a ReLu [53] 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. 12 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. 12 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 4–14), which consists of T iterations. In each iteration, the algorithm updates the hidden states of flows h_f, queues h_q, and links h_l, based on their interactions

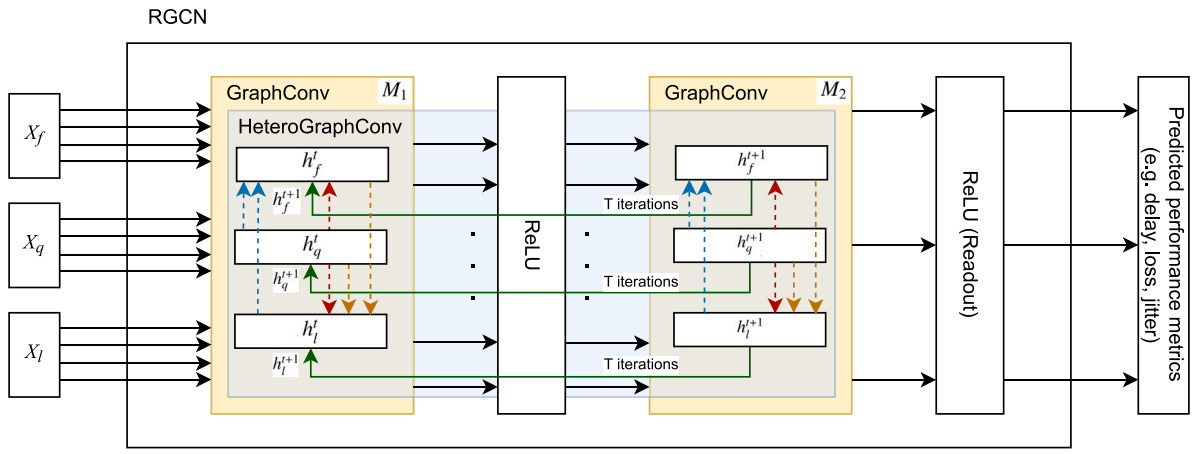

Fig. 12. Schematic representation of GNNetSlice.

within the graph H. Each hidden state captures contextual information needed to predict the KPIs.

For each flow f (lines 6–8), the algorithm updates its hidden state h_f using the function M, which aggregates information from the queues and links through which the flow passes. This enables h_f to capture the behavior of queues and links depending on the traffic of the flow.

Similarly, for each queue q, the algorithm updates its hidden state h_q using the same function M, now considering the states of the flows passing through it and the link where it is located. This allows h_q to capture information about local congestion and flow intensity, that will be used to update future h_f states, which are used for the predictions.

Likewise, for each link l (lines 9–11), 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 12–14) 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. These hidden states collectively enable the network to output accurate predictions of traffic-related KPIs.

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 15–16).

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'

1: $h_f^0 \leftarrow H_f(X_f)$
2: $h_q^0 \leftarrow H_q(X_q)$
3: $h_l^0 \leftarrow H_l(X_l)$
4: **for** t = 0 to T-1 **do**
5: $M_1 \leftarrow GraphConv(h^t)$
6: $M_2 \leftarrow GraphConv(h^t)$
7: **for** f in H **do**
8: $h_f^{t+1} \leftarrow h_f^t + M_1(h_l^t, h_f^t)$
9: $h_f^{t+1} \leftarrow h_f^{t+1} + M_2(h_q^t, h_f^{t+1})$
10: **for** l in H **do**
11: $h_l^{t+1} \leftarrow h_l^t + M_1(h_f^t, h_l^t)$
12: $h_l^{t+1} \leftarrow h_l^{t+1} + M_2(h_q^t, h_l^{t+1})$
13: **for** q in H **do**
14: $h_q^{t+1} \leftarrow h_q^t + M_1(h_l^t, h_q^t)$
15: $h_q^{t+1} \leftarrow h_q^{t+1} + M_2(h_f^t, h_q^{t+1})$
16: $X_f' \leftarrow R_f(h_f^t)$
17: **return** X_f'

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 [54]. The features used in the model are detailed in Table 3. 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 (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 [55]. Three engineered input features are created for the GNNetSlice model. Firstly, O_t, as defined in Section 5. Secondly, Calculated losses (L), which are determined by the Eq. (4). 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. (5), where Q_s is the queue size in number of packets, multiplied by the AvgPktSize A_p in number of bits and in the queue, and divided by the product of weight w and the output link capacity C_l.

$$L = \frac{(\sum_{f \in Q} T_f) - (w \times C_\ell)}{\sum_{f \in Q} T_f} \tag{4}$$

$$M_d = \frac{Q_s \times A_p}{w \times C_\ell} \tag{5}$$

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) 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.

Table 3
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)

Table 4
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

6.4. Model training

We use MSE (Mean Squared Error) as the training loss metric. To measure and compare the prediction accuracy in the test phase, the Symmetric Mean Average Percentage Error (SMAPE) is used as metric, as it is symmetrical in comparison to Mean Average Percentage Error (MAPE), and human-understandable. SMAPE quantifies prediction accuracy as the average percentage of relative errors (Eq. (6)). 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.

$$SMAPE = \frac{100\%}{n} \sum_{i=1}^{n} \frac{|\hat{y}_l - y_i|}{(|y_i| + |\hat{y}_l|)/2} \tag{6}$$

After hyper-parameter tuning, our best GNNetSlice model has been trained during 95 epochs. We found that, generally, larger values of the hyperparameters result in higher precision. Generally, when creating ML models, a good hyperparameter selection is crucial [56] (see Table 4).

The training lasted 4 h and 15 min (95 epochs with batches of size 1000) to achieve the best SMAPE results for the three KPIs predicted in this work. The training of was performed using only the CPU, as the model performed slower on Graphics Processing Unit (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 subpar results. The training hardware consisted of an AMD Ryzen 5 5600X CPU 3.7 GHz and 64 GB of DDR4 RAM.

Table 5
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

7. Evaluation

For the evaluation of the GNNetSlice model we use a suite of established metrics including Symmetric Mean Average Percentage Error (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 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 test,s 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 selecting samples randomly, in a proportion of 80% training, 10% validation and 10% testing.

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 5 presents the SMAPE errors for each slice type, indicating the level of prediction accuracy achieved.

Furthermore, Fig. 13 illustrates the cumulative distribution functions (CDFs) of the relative error in predictions for each slice type: eMBB, mIoT, and URLLC. The relative error was calculated as the percentage difference between predicted and actual values relative to the actual values. Each CDF curve represents the proportion of predictions falling below a given relative error threshold, showing the accuracy distribution across the three slice types. In all cases, the curves indicate that most relative errors are close to zero, demonstrating the model's high predictive accuracy. The mIoT slice shows a slightly broader error

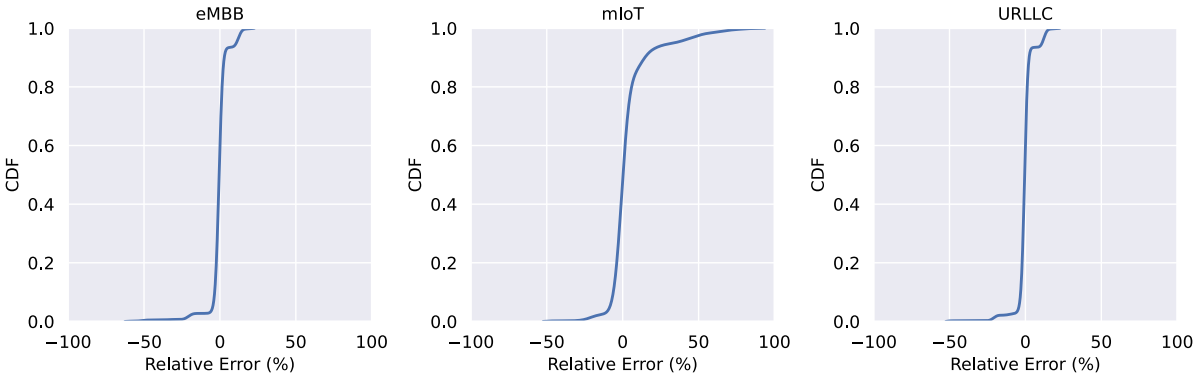

Fig. 13. Cumulative Distribution Functions (CDF) of the relative error of the predictions testing the presented model.

distribution compared to eMBB and URLLC, yet it remains within an acceptable range, as further seen by the results in Table 5. This suggests that GNNetSlice is able to maintain reliable predictions across various service classes.

Additionally, we 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.

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 cross-validation results are depicted in Fig. 14, where boxplots illustrate the distribution of SMAPE errors for delay, losses, and jitter prediction across the folds. 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 [57].

Overall, the cross-validation reveals a stable performance of GNNetSlice across different network performance metrics. Specifically, 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.

7.3. Topological generalization

Thirdly, we evaluate the performance of GNNetSlice 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. 15. 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.

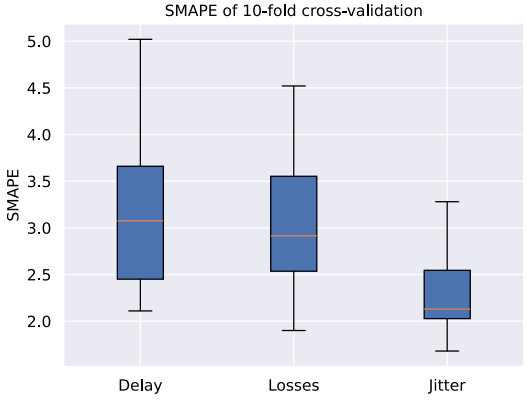

Fig. 14. SMAPE errors for delay, losses, and jitter prediction with the 10-fold cross-validation.

Table 6
Ablation study GNNetSlice model SMAPE results on test dataset for different modifications, including the average error change for each modification.

Modification	Delay	Loss	Jitter	Change
GNNetSlice (no changes)	5.22	2.04	1.95	N/A
Remove GraphConv layer	12.16	13.79	9.30	+357.5%
Remove queue input data	9.26	13.54	7.02	+280.9%
Remove ReLU	11.11	14.07	6.92	+301.3%

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.

7.4. Ablation study

An ablation study is applied on the GNNetSlice model, to check the importance of the different model components. The SMAPE for *delay*, *loss*, and *jitter* is analyzed under three different modifications of key components of the network: removing one layer of the two GraphConv layers, removing input data from the queues in the input graph, and finally the deletion of the intermediate ReLU activation function. The impact of these modifications is shown in Table 6.

The ablation study demonstrates that each component removed plays a critical role in the GNNetSlice performance. Removing any of these components leads to increased prediction errors for delay, loss, and jitter, with the greatest impact observed when a layer is removed.

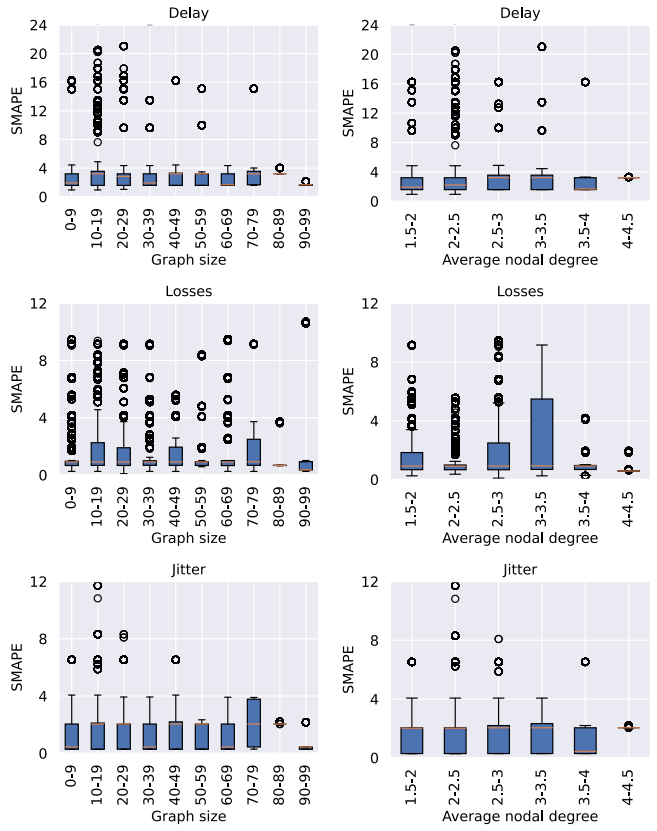

Fig. 15. SMAPE of the predictions of GNNetSlice, per number of nodes and average nodal degree.

Table 7
Comparison of 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

This analysis suggests that the original configuration of the model is optimal for minimizing prediction errors, as all the main elements are required to maintain good accuracy.

7.5. Comparison with state of the art models

Next, we compare the performance of GNNetSlice with a state-of-the-art model, RouteNet-Fermi [38], 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 7 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 behaviors.

7.6. Inference and training speed

During the testing stage, our model demonstrated an average prediction time of 26 ms per sample for delay, jitter, and losses. Each simulation, aimed at achieving slightly more accurate results, required an average of 1.7 h. This exemplifies the capabilities of AI/ML approaches in B5G network management.

A notable strength of our proposed model is its remarkable efficiency during 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.

8. Conclusions and future work

This paper introduces GNNetSlice, a model built to predict the performance of network slicing in core and transport segment in 5G and B5G networks. 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.

Our methodology relies on a data-driven approach, achieving a good balance between speed and accuracy. Initially, we constructed a dataset 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, we 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.

In terms of future work, some encouraging improvements are further optimization of the attributes used, advanced feature engineering, the incorporation of additional relations into the input graph, and testing GNNetSlice on data from real testbed.

CRediT authorship contribution statement

Miquel Farreras: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Jordi Paillissé:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Lluís Fàbrega:** Writing – review & editing, Validation, Supervision, Investigation, Conceptualization. **Pere Vilà:** Writing – review & editing, Validation, Supervision, Investigation, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

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. Jordi Paillisse is funded by European Union-Next Generation EU, Ministry of Universities and Recovery, Transformation and Resilience Plan, through a call from Universitat Politècnica de Catalunya (Grant Ref. 2022UPC-MSC-93871).

Data availability

The model developed and the data that support the findings of this study are publicly available in GitHub https://zenodo.org/records/10610616.

References

[1] X. Li, M. Samaka, H.A. Chan, D. Bh, L. Gupta, C. Guo, R. Jain, Network slicing for 5G: Challenges and opportunities, IEEE Internet Comput. (2018) http://dx.doi.org/10.1109/MIC.2018.326150452, 1–1.

[2] P. Subedi, A. Alsadoon, P.W. Prasad, S. Rehman, N. Giweli, M. Imran, S. Arif, Network slicing: a next generation 5G perspective, Eurasip J. Wirel. Commun. Netw. 2021 (2021) http://dx.doi.org/10.1186/s13638-021-01983-7.

[3] 3GPP, TS 23.501, 2023, URL https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3144. [Online; Accessed 20- July 2023].

[4] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C.H. Liu, D. Yang, Experience-driven networking: A deep reinforcement learning based approach, in: IEEE INFOCOM 2018-IEEE, IEEE, 2018, pp. 1871–1879.

[5] B. Martini, M. Gharbaoui, P. Castoldi, Intent-based network slicing for SDN vertical services with assurance: Context, design and preliminary experiments, Future Gener. Comput. Syst. 142 (2023) 101–116, http://dx.doi.org/10.1016/j.future.2022.12.033.

[6] K. Bochie, M.S. Gilbert, L. Gantert, M.S. Barbosa, D.S. Medeiros, M.E.M. Campista, A survey on deep learning for challenged networks: Applications and trends, J. Netw. Comput. Appl. 194 (2021) 103213, http://dx.doi.org/10.1016/j.jnca.2021.103213, URL https://www.sciencedirect.com/science/article/pii/S1084804521002149.

[7] C. Zhou, H. Yang, X. Duan, D. Lopez, A. Pastor, Q. Wu, M. Boucadair, C. Jacquenet, Digital Twin Network: Concepts and Reference Architecture, Internet Draft, Internet Engineering Task Force, 2022, URL https://datatracker.ietf.org/doc/draft-irtf-nmrg-network-digital-twin-arch/01/.

[8] Y. Wu, K. Zhang, Y. Zhang, Digital twin networks: A survey, IEEE Internet Things J. 8 (18) (2021) 13789–13804, http://dx.doi.org/10.1109/JIOT.2021.3079510.

[9] R. Sato, A survey on the expressive power of graph neural networks, 2020, CoRR arXiv:2003.04078.

[10] P. Tam, I. Song, S. Kang, S. Ros, S. Kim, Graph neural networks for intelligent modelling in network management and orchestration: A survey on communications, Electron. (Switzerland) 11 (2022) http://dx.doi.org/10.3390/electronics11203371.

[11] I. Bello, H. Pham, Q.V. Le, M. Norouzi, S. Bengio, Neural combinatorial optimization with reinforcement learning, 2016, CoRR arXiv:1611.09940.

[12] P.W. Battaglia, J.B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V.F. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, Ç. Gülçehre, H.F. Song, A.J. Ballard, J. Gilmer, G.E. Dahl, A. Vaswani, K.R. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M.M. Botvinick, O. Vinyals, Y. Li, R. Pascanu, Relational inductive biases, deep learning, and graph networks, 2018, CoRR arXiv:1806.01261.

[13] J. Gilmer, S.S. Schoenholz, P.F. Riley, O. Vinyals, G.E. Dahl, Neural message passing for quantum chemistry, 2017, CoRR.

[14] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, R.P. Adams, Convolutional networks on graphs for learning molecular fingerprints, 2015, CoRR.

[15] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Trans. Neural Netw. 20 (1) (2009) 61–80, http://dx.doi.org/10.1109/TNN.2008.2005605.

[16] R.L. Murphy, B. Srinivasan, V.A. Rao, B. Ribeiro, Relational pooling for graph representations, 2019.

[17] J. Leskovec, CS224w: Machine learning with graphs, 2021, URL http://cs224w.stanford.edu.

[18] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, H. Flinck, Network slicing and softwarization: A survey on principles, enabling technologies, and solutions, IEEE Commun. Surv. Tutor. 20 (3) (2018) 2429–2453, http://dx.doi.org/10.1109/COMST.2018.2815638.

[19] M. Chahbar, G. Diaz, A. Dandoush, C. Cérin, K. Ghoumid, A comprehensive survey on the E2E 5g network slicing model, IEEE Trans. Netw. Serv. Manag. 18 (1) (2021) 49–62, http://dx.doi.org/10.1109/TNSM.2020.3044626.

[20] F. Ciucu, J. Schmitt, Perspectives on network calculus - no free lunch, but still good value, Comput. Commun. Rev. (2012) http://dx.doi.org/10.1145/2342356.2342426.

[21] B. Bylina, J. Bylina, Using Markov chains for modelling networks, Ann. UMCS Inform. AI (2005) 27–34, URL http://www.annales.umcs.lublin.pl/.

[22] M. Kim, Analysis of feasible region in network slicing: Markov chain approach, Comput. Commun. 212 (2023) 420–431, http://dx.doi.org/10.1016/j.comcom.2023.08.014, URL https://www.sciencedirect.com/science/article/pii/S0140366423002992.

[23] M. Ferriol-Galmés, K. Rusek, J. Suárez-Varela, S. Xiao, X. Shi, X. Cheng, B. Wu, P. Barlet-Ros, A. Cabellos-Aparicio, RouteNet-Erlang: A graph neural network for network performance evaluation, in: IEEE INFOCOM 2022 - IEEE Conference on Computer Communications, 2022, pp. 2018–2027, http://dx.doi.org/10.1109/INFOCOM48880.2022.9796944.

[24] E. Weingartner, H. vom Lehn, K. Wehrle, A performance comparison of recent network simulators, in: 2009 IEEE International Conference on Communications, 2009, pp. 1–5, http://dx.doi.org/10.1109/ICC.2009.5198657.

[25] Q. Yang, X. Peng, L. Chen, L. Liu, J. Zhang, H. Xu, B. Li, G. Zhang, DeepQueueNet: towards scalable and generalized network performance estimation with packet-level visibility, in: Proceedings of the ACM SIGCOMM 2022 Conference, SIGCOMM '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 441–457, http://dx.doi.org/10.1145/3544216.3544248.

[26] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, X. Costa-Perez, AZTEC: Anticipatory capacity allocation for zero-touch network slicing, in: IEEE INFOCOM 2020 - IEEE Conference on Computer Communications, 2020, pp. 794–803, http://dx.doi.org/10.1109/INFOCOM41043.2020.9155299.

[27] A.S. Khatouni, F. Soro, D. 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.

[28] Z. Gao, 5G traffic prediction based on deep learning, Comput. Intell. Neurosci. (2022) 3174530, http://dx.doi.org/10.1155/2022/3174530.

[29] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, X. Costa-Perez, DeepCog: Cognitive network management in sliced 5G networks with deep learning, in: Proceedings - IEEE INFOCOM, IEEE, 2019, http://dx.doi.org/10.1109/INFOCOM.2019.8737488.

[30] A.R. Abdellah, O.A. Mahmood, R. Kirichek, A. Paramonov, A. Koucheryavy, Machine learning algorithm for delay prediction in IoT and tactile internet, Futur. Internet (2021) http://dx.doi.org/10.3390/fi13120304.

[31] D. Gammelli, K. Yang, J. Harrison, F. Rodrigues, F.C. Pereira, M. Pavone, Graph neural network reinforcement learning for autonomous mobility-on-demand systems, in: Proceedings of the 60th IEEE Conference on Decision and Control, CDC 2021, Institute of Electrical and Electronics Engineers Inc., United States, 2021, pp. 2996–3003, http://dx.doi.org/10.1109/CDC45484.2021.9683135.

[32] A. Rkhami, Y. Hadjadj-Aoul, G. Rubino, A. Outtagarts, MonGNN: A neuroevolutionary-based solution for 5G network slices monitoring, in: Proceedings - Conference on Local Computer Networks, LCN, IEEE, 2021, pp. 185–192, http://dx.doi.org/10.1109/LCN52139.2021.9524880.

[33] M. Ferriol-Galmés, J. Suárez-Varela, J. Paillissé, X. Shi, S. Xiao, X. Cheng, P. Barlet-Ros, A. Cabellos-Aparicio, Building a digital twin for network optimization using graph neural networks, Comput. Netw. 217 (2022) 109329, http://dx.doi.org/10.1016/j.comnet.2022.109329, URL https://www.sciencedirect.com/science/article/pii/S1389128622003681.

[34] Z. Ge, J. Hou, A. 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, http://dx.doi.org/10.1109/DCOSS54816.2022.00066.

[35] A. Asheralieva, D. Niyato, Y. Miyanaga, Efficient dynamic distributed resource slicing in 6G multi-access edge computing networks with online ADMM and message passing graph neural networks, IEEE Trans. Mob. Comput. (2023) http://dx.doi.org/10.1109/TMC.2023.3262514.

[36] T. Dong, Z. Zhuang, Q. Qi, J. Wang, H. Sun, F.R. Yu, T. Sun, C. Zhou, J. Liao, Intelligent joint network slicing and routing via GCN-powered multi-task deep reinforcement learning, IEEE Trans. Cogn. Commun. Netw. 8 (2022) 1269–1286, http://dx.doi.org/10.1109/TCCN.2021.3136221.

[37] A.H. Lashkari, G.D. Gil, M.S.I. Mamun, A.A. Ghorbani, 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, http://dx.doi.org/10.5220/0006105602530262.

[38] M. Ferriol-Galmés, J. Paillisse, J. Suárez-Varela, K. Rusek, S. Xiao, X. Shi, X. Cheng, P. Barlet-Ros, A. Cabellos-Aparicio, RouteNet-Fermi: Network modeling with graph neural networks, 2022, arXiv:2212.12070.

[39] 5GPPP, AI and ML – enablers for beyond 5G networks, 2021, http://dx.doi.org/10.5281/zenodo.4299895.

[40] 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, http://dx.doi.org/10.1109/ICCCN.1999.805515.

[41] C.Y. Chang, T.G. Ruiz, F. Paolucci, M.A. Jimenez, J. Sacido, C. Papagianni, F. Ubaldi, D. Scano, M. Gharbaoui, A. Giorgetti, L. Valcarenghi, K. Tomakh, A. Boddi, A. Caparros, M. Pergolesi, B. Martini, Performance isolation for network slices in industry 4.0: The 5growth approach, IEEE Access (2021) 166990–167003, http://dx.doi.org/10.1109/ACCESS.2021.3135827.

[42] A. Minokuchi, S. Isobe, 5G core network standardization trends, in: NTT DOCOMO Technical Journal, Technology Reports, Vol 19-3, 2020, pp. 48–54.

[43] M. Farreras, J. Paillissé, L. Fàbrega, P. Vilà, Generation of a network slicing dataset: The foundations for AI-based B5G resource management, Data Brief 55 (2024) 110738, http://dx.doi.org/10.1016/j.dib.2024.110738, URL https://www.sciencedirect.com/science/article/pii/S2352340924007054.

[44] S. Knight, H. Nguyen, N. Falkner, R. Bowden, M. Roughan, The internet topology zoo, IEEE J. Sel. Areas Commun. 29 (9) (2011) 1765–1775, http://dx.doi.org/10.1109/JSAC.2011.111002.
[45] A.A. Hagberg, D.A. Schult, P.J. Swart, Exploring network structure, dynamics, and function using networkx, in: G. Varoquaux, T. Vaught, J. Millman (Eds.), Proceedings of the 7th Python in Science Conference, Pasadena, CA USA, 2008, pp. 11–15.
[46] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, Z. Zhang, Deep graph library: A graph-centric, highly-performant package for graph neural networks, 2019, arXiv preprint arXiv:1909.01315.
[47] BNN, DataNet API documentation, 2021, URL https://github.com/BNN-UPC/datanetAPI/tree/challenge2021. [Online; Accessed 31 Januuary 2022].
[48] Deep Graph Library Developers, 1.5 heterogeneous graphs, 2024, URL https://docs.dgl.ai/en/latest/guide/graph-heterogeneous.html. (Accessed 18 November 2024).
[49] M. Schlichtkrull, T.N. Kipf, P. Bloem, R. van den Berg, I. Titov, M. Welling, Modeling relational data with graph convolutional networks, 2017, arXiv:1703.06103.
[50] L. Gong, Q. Cheng, Exploiting edge features in graph neural networks, 2019, arXiv:1809.02709.
[51] K. Xu, W. Hu, J. Leskovec, S. Jegelka, How powerful are graph neural networks? 2018, CoRR abs/1810.00826. arXiv:1810.00826. URL http://arxiv.org/abs/1810.00826.
[52] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, 2017, arXiv:1609.02907.
[53] L. Lu, Dying ReLU and initialization: Theory and numerical examples, Commun. Comput. Phys. 28 (5) (2020) 1671–1706, http://dx.doi.org/10.4208/cicp.oa-2020-0165.
[54] S.K. Maurya, X. Liu, T. Murata, Feature selection: Key to enhance node classification with graph neural networks, CAAI Trans. Intell. Technol. 8 (1) (2023) 14–28, http://dx.doi.org/10.1049/cit2.12166.
[55] G. Dong, H. Liu, Feature Engineering for Machine Learning and Data Analytics, first ed., CRC Press, Inc., USA, 2018.
[56] T. Yu, H. Zhu, Hyper-parameter optimization: A review of algorithms and applications, 2020, arXiv:2003.05689.
[57] M. Kuhn, K. Johnson, Applied Predictive Modeling, Springer, 2013.