3RD WORKSHOP ON MACHINE LEARNING IN NETWORKING (MaLeNe) PROCEEDINGS

SEPTEMBER 1, 2025

CO-LOCATED WITH
THE 6TH INTERNATIONAL CONFERENCE ON NETWORKED SYSTEMS (NETSYS 2025)
ILMENAU, GERMANY

Predicting Performance Metrics in Edge-Cloud Networks using Graph Neural Networks

Christian Maier, Nina Großegesse, Felix Strohmeier
Salzburg Research Forschungsgesellschaft mbH
<firstname.lastname@salzburgresearch.at>

***Abstract*—This paper explores the application of Graph Neural Networks (GNNs) for predicting performance metrics in edge-cloud networks. By modeling the edge-cloud network as a graph, where nodes represent devices, and edges represent communication links, GNNs effectively capture the complex interdependencies and interactions within the network. We demonstrate that GNNs can accurately predict key performance metrics such as latency and jitter, using data from real network conditions. Our findings highlight the potential of GNNs to enhance performance monitoring and optimization in edge-cloud environments, paving the way for more efficient resource management and energy-efficiency.**

***Index Terms*—Graph Neural Networks, Edge-Cloud-Networks, Performance Prediction**

I. Introduction

Machine learning models as digital twins in network management take the state of the network as input and produce predictions of performance metrics in the network as output. In the context of edge-cloud networks, important metrics are end-to-end latency, jitter and packet-loss of packet flows. This performance prediction is intended to replace measurements or estimation by simulations, as these are usually associated with considerable effort (in terms of network overhead or computing time).

The prediction of performance metrics can be done with a variety of different machine learning models, like decision trees or artificial neural networks. A common approach in the literature is the prediction of network performance metrics using Graph Neural Networks. GNNs have emerged as a powerful tool for predicting performance metrics in communication networks due to their ability to model complex relationships and dependencies among network components. By representing the network as a graph, where nodes correspond to devices or network elements and edges represent connections or interactions, GNNs can effectively capture the topological structure and dynamic behaviour of the network.

Recent studies have demonstrated that GNNs can predict various performance metrics, such as latency, throughput, and packet loss, by learning from historical data and network configurations. Their capacity to incorporate both node features (e.g., bandwidth, processing power) and edge features (e.g., link quality, distance) allows for a more nuanced understanding of how different factors influence overall network performance.

Additionally, GNNs can generalize well to unseen network topologies, making them suitable for real-time applications where network conditions may change frequently. This adaptability, combined with their ability to process large-scale data efficiently, positions GNNs as a promising approach for enhancing the performance prediction capabilities in modern communication networks, particularly in scenarios involving dynamic and heterogeneous environments (which is particularly the case for edge cloud networks).

Training a GNN to predict performance metrics in communication networks typically involves several steps. First, the network is represented as a graph, where nodes correspond to network elements (such as routers and switches) and edges represent the connections between them. Each node and edge is associated with features that capture relevant information, such as bandwidth and traffic load. For GNNs, it is not relevant to have a complete picture of the underlying network including all components, but only for those nodes and links that shall be analysed.

The training process begins with the collection of historical performance data, which serves as the ground truth for the metrics to be predicted. This data is used to create labelled training samples, where the input consists of the graph structure and associated features, while the output corresponds to the performance metrics (e.g., latency, throughput).

During training, the GNN learns to aggregate information from neighbouring nodes and edges through multiple layers of message passing. This process allows the model to capture both local and global patterns in the graph. The GNN is typically trained using supervised learning techniques, where a loss function (such as mean squared error) measures the difference between the predicted metrics and the actual values from the training data. Optimization algorithms, such as Adam or stochastic gradient descent, are employed to minimize this loss function by adjusting the model parameters.

Once trained, the GNN can be evaluated on a separate validation dataset to assess its predictive performance. Fine-tuning may be performed to improve accuracy, and the model can be deployed for real-time predictions in dynamic communication environments, adapting to changes in network conditions as new data becomes available. Overall, the training of GNNs for performance metric prediction leverages the unique graph structure of communication networks to enhance prediction

This work has been partly funded by the Austrian Federal Ministry of Innovation, Mobility, and Infrastructure (BMIMI) and the National Austrian Funding Agency (FFG).

accuracy and efficiency.

The remainder of the paper is structured as follows. Section II surveys related work on performance prediction with machine learning approaches. Section III gives an overview of the background (both on the used measurement framework and on GNNs), Section IV provides a detailed description of the methodology and Section V presents the results of our evaluation. Finally, Section VI concludes the paper and provides an outlook to future work.

II. Related work

Performance metrics in edge cloud networks have attracted a lot of attention in recent years due to the increasing demand for low latency applications and the proliferation of Internet of Things (IoT) devices. Various studies have investigated different aspects of performance measurement, focusing on latency, bandwidth, reliability and resource utilization.

One of the pioneering works in the domain of predicting performance metrics of communication networks by a machine learning model which takes the graph structure of the network into account is by Rusek et al. [12]. They introduced a GNN-based framework called RouteNet for network performance prediction and demonstrated that GNNs could effectively capture the spatial and temporal dependencies in network data, leading to improved accuracy in predicting latency and throughput compared to traditional machine learning approaches. Their experiments showed that GNNs outperformed baseline models by leveraging the graph structure to learn from both node features and connectivity patterns. Building on this, there is a lot of further work that refine the RouteNet model (e.g. [5], [4]) and thus further improve accuracy of the predictions. In another significant contribution, Dai et al. [3] explored the use of GNNs for multi-task learning in communication networks. However, all these studies use simulation data and do not consider edge cloud networks in particular.

Furthermore, recent advancements have focused on integrating GNNs with reinforcement learning techniques to optimize network performance dynamically. For instance, Li et al. [8] developed a GNN-based reinforcement learning model that predicts performance metrics while simultaneously optimizing routing decisions in real-time. Their approach showcased the potential of combining predictive modelling with decision-making processes, leading to enhanced network efficiency and reduced latency. Li et al. [9] propose a distributed scheduler based on GNNs an reinforcement learning for edge clusters that minimizes the total completion time of ML tasks through co-optimizing task placement and fine-grained multi-resource allocation

III. Background

This section provides background information about the used performance measurement framework and about GNNs.

A. Performance Measurement Framework

Measuring performance on the network (IP) and transport (TCP/UDP) level can be done by various available tools (e.g. iperf3 [7], wireshark [6], ping [10]) and according to various metrics, such as IP performance metrics (IPPM), round-trip time (RTT), round-trips per minute (RPM), throughput or topology. Our approach to performance measurement is to use the MINER infrastructure as programmable orchestration framework [2]. MINER is a distributed Java application, where so-called "Toolproxies" are passively waiting for execution orders from a centralised measurement application. Communication between the nodes may be secured by a VPN. All configuration parameters are defined by the measurement application, which schedules the execution as soon as all the involved measurement tools are ready to run. Measurement tools can be standard tools integrated by the Toolproxy, or Miner-specific tools, such as the "IPPMTool". During and/or after the measurement execution, the results are collected, so that follow-up measurement analytics can take place centralised after the run.

After investigating different performance metrics and studying the related work in Section II, we considered latency, jitter, packet loss and throughput as the most relevant metrics in the edge-cloud continuum. For the analyses performed in this paper we measured latency, jitter on application and packet level, as well as packet loss by generating active UDP measurement flows with constant bitrates, using the Miner IPPMTool [2]. To receive a proper variation of measurement results, different packet sizes and rates are selected before starting the measurement flows.

B. Graph Neural Networks for Homogeneous Graphs

GNNs are machine learning models, which are made to process data in the form of graphs appropriately. Here, a graph is a mathematical object, consisting of nodes and (directed or undirected) edges between them. A frequently occurring task is the determination of quantities $y_v \in \mathbb{R}$ which are assigned to individual nodes v. In the context of a communication network, the nodes of the graph are typically the devices of the network, the edges represent communication links and a common quantity of a node is for example the utilization of a queue of the represented device. The task of a *generic* GNN can then be subdivided into two subtasks: First, a *hidden state vector* h_v needs to be computed for each node v. This vector h_v lies in some chosen state space $\mathbb{R}^m$ of dimension m and consists of information on the state of v at some level of granularity. It is computed by an iterative *message passing* scheme: After an initialisation of h_v with node-level features related to v, each node sends its state to all of its neighbours. Hence, each node v receives a certain number of states $h_{v_1}, \ldots, h_{v_k}$, where k is the number of neighbours of v. These states, together with the state h_v of v itself, are converted into an *aggregated message* m_v:

$$m_v = \sum_{i=1}^{k} M(h_v, h_{v_i}) \tag{1}$$

by a *message function* $M : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^n$. The dimension n of the codomain of M is again a chosen value. Using this aggregated message, an *update function* $U : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^m$ computes a new hidden state $U(h_v, m_v)$ for each node v. This message passing is repeated T times until the states of all nodes have (approximately) reached stationary values. Then, secondly, a *readout function* $R : \mathbb{R}^m \to \mathbb{R}$ computes y_v for each node v by applying R to h_v.

A generic GNN thus essentially consists of three functions: M, U and R. The mapping rules of these functions are given by the application of certain independent neural networks (like feed forward or recurrent neural networks). This justifies the name GNN and allows the execution of a training process: The internal parameters of the neural networks are updated via supervised learning in order to compute the target quantities y_v accurately. To achieve this, instances of graphs together with the target quantities y_v for all nodes have to be provided. Note that the way in which such a generic GNN is modelled enables an application to graphs of different sizes and structures (both during training and predicting).

Further details on this generic GNN architecture can be found in the literature [1, 11] and in the references provided there. Many GNN models deviate from the architecture of standard GNN models. Particularly, they consider heterogeneous graphs as input (i.e. with various types of nodes), and build a double message-passing phase to exchange the information between the different element types. In the next subsection we recall this architecture.

C. Graph Neural Networks for Heterogeneous Graphs

Heterogeneous graphs are graphs which consist of nodes of different types. For simplicity, we only consider heterogeneous graphs with nodes of two types A and B. The message passing explained in the previous subsection is then adapted to a so-called 2-stage message passing phase: In the first stage, nodes of type A send their messages to all nodes of type B to which they are connected. This is done in the same way as explained above. This results in updates of the hidden state vectors for all nodes of type B. After that, all nodes of type B send their messages to the nodes of type A, which the updates the states of the type A nodes. Notice that the message functions and update functions can be different for the two stages in the message passing phase. The readout function is then usually only applied to the nodes either of type A or of type B. In our approach (which we outline below), the nodes of the graphs correspond to links and flows in the physical network. Thus we will have two types of nodes. The readout function will then only be applied to nodes which correspond to flows, since we are eventually interested in calculating performance metrics of flows (mean latencies, mean jitter and packet loss).

IV. Methodology

This section describes our hardware testbed, the dataset which we created from this testbed, the architecture of our ML model and the approach taken in the training process.

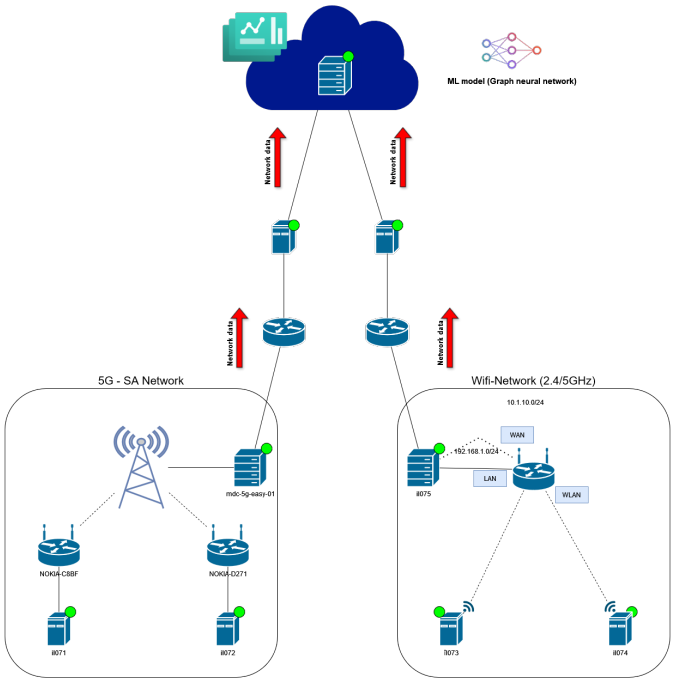

Fig. 1. Hardware testbed

A. Hardware Testbed

An overview diagram of our hardware setup is shown in Fig. 1. It consists of two local edge networks, one using 5G, the other one using Wi-Fi equipment. The 5G edge network is built using a 5G indoor base station located in our laboratory, connected to its 5G core located at the telecom provider premises. Two measurement endpoints are connected via wireless access routers to the indoor base station, a third measurement endpoint is installed in the 5G core. Measurement traffic flows are generated between all three nodes in both directions. More details on the measurements itself are available in Section IV-B.

The other edge network built on 5GHz-Wi-Fi-Technology (”Wi-Fi 5”), which also has three MINER measurement points installed. All measurement nodes are located in our laboratory, two on wireless nodes, and one node connected by wire to the wireless router. Also in this part of the network, measurement traffic flows are generated between all three nodes in both directions.

To be more precise, for each ordered pair of measurement points from the same local network, a packet flow with packets of constant size and constant inter packet time between two packets is generated. Each of the local edge networks is connected to the cloud node via an intermediate on-premise node.

B. Dataset

To create a dataset in our hardware testbed, we use the following approach: We randomly choose packet sizes and inter packet times for each packet flow in our local edge networks. Since there are six packet flows in each local network, this

```
directed: true
multigraph: false
graph: {}
nodes:
  0:
    entity: "node"
    f_v: 0
    id: "il072"
  1:
    entity: "node"
    f_v: 0
    id: "il071"
  2:
    entity: "node"
    f_v: 0
    id: "mdc-5g-easy-01"
links:
  0:
    f_e:
      0: "1046"
      1: "17.317"
    owdApp: "23.084415584415584"
    owdPcap: "23004.324675324675"
    jitterApp: "-0.03246753246753246"
    jitterPcap: "7.307359307359308"
    source: "il072"
    target: "il071"
  1:
    f_e:
      0: "1368"
      1: "91.773"
    owdApp: "22.25"
    owdPcap: "22264.476744186046"
    jitterApp: "-0.011363636363636364"
    jitterPcap: "-72.38823529411765"
    source: "il072"
    target: "mdc-5g-easy-01"
  2:
    f_e:
      0: "805"
      1: "34.413"
    owdApp: "20.689655172413794"
    owdPcap: "20607.82327586207"
    jitterApp: "-0.03879310344827586"
    jitterPcap: "29.745689655172413"
    source: "il071"
    target: "il072"
  3:
    f_e:
      0: "1433"
      1: "64.365"
    owdApp: "20.830645161290324"
    owdPcap: "20629.120967741936"
    jitterApp: "-0.04838709677419355"
    jitterPcap: "-51.016129032258064"
    source: "il071"
    target: "mdc-5g-easy-01"
  4:
    f_e:
      0: "683"
      1: "63.453"
    owdApp: "0.9920634920634921"
    owdPcap: "998.436507936508"
    jitterApp: "-0.007936507936507936"
    jitterPcap: "-8.976190476190476"
    source: "mdc-5g-easy-01"
    target: "il072"
  5:
    f_e:
      0: "1463"
      1: "42.156"
    owdApp: "1.6421052631578947"
    owdPcap: "1548.1736842105263"
    jitterApp: "-0.02631578947368421"
    jitterPcap: "-27.189473684210526"
    source: "mdc-5g-easy-01"
    target: "il071"
```


Fig. 2. An example data sample from a measurement in the local 5G-network. The nodes of the communication network are the nodes of the graph. The links of the graph are the packet flows in the communication network. For each packet flow, the parameters (packet size s and inter-packet-time t) are summarized in the list $f_e = [s, t]$.

results in twelve parameters for each measurement. The packet sizes are varied from 300 to 1500 Bytes and the packet rates between 10 and 100 packets per second resulting in bitrates of 20 to 1200 kbps for the measurement flows. The chosen packet sizes and inter packet times define the configuration of the network traffic. For each configuration, we measure the performance metrics explained in Sec. III (i.e. latency, jitter and packet loss, where latency and jitter are measured both on application and on packet level) for a measurement interval of 10 seconds, where we then use the mean value for latency and jitter. The data set is then saved as a JSON file. An example data sample is shown in Fig. 2.

Our data set in the 5G network consists of measurements from February 26, 2024 to December 31, 2024. Every day between 9 p.m. and 5 a.m., a ten-second measurement was taken every 4 minutes. We only took measurements at night and only every four minutes, because other measurements are taken in this network during the day and in other time slots (which are not relevant for this paper). As some of the measurements failed (for various reasons, e.g. power outages or disruptions in the 5G network), the pre-processed data does not contain a measurement result for every resulting measurement time.

For organizational reasons, the measurements in the Wifi-network ran from 22 May, 2024 to 31 December, 2024. The time between two measurements was also 4 minutes here (to be consistent with the 5G measurements), but measurements were taken throughout the whole day (to obtain a data set similar in size to that in the 5G network).

For each measurement and each transmitted packet, we saved the latency and jitter (both at Pcap and application level) and whether the packet was successfully transmitted. We thus obtained five time series for each packet flow. From these time series, we calculate statistical mean values for the latency and jitter. Fig. 3 and Fig. 4 shows an example of a histogram of the measurement results both for the 5G network and the WiFi network. For this calculation, however, we have neglected the first and last second of the respective time series (in order not to take into account any transient behaviour)

Since we did not have access to the 5G core, there is a synchronization issue in our latency measurements in the 5G network. However, this is irrelevant for our investigations. Synchronization of the computers in the laboratory takes place via their own network to a local, GPS-synchronized time server, not via the "network under test". The core was synchronized to an external time server. In general, synchronization is only important for one-way-delay measurements.

C. Architecture

We use a graph neural network model to predict performance metrics of packet flows in edge-cloud networks, cf. Fig. 5. As already explained, the packet flows consist of packets of constant size with constant inter-packet times between two packets.

The structure of the used 2-stage Graph neural network model is as follows: The nodes of the graph correspond to links and flows in the data set. This means that the graphs which we consider consist of nodes of two types: links and flows. For each link, there is a directed edge to each flow which uses this link. For each flow, there is a directed edge to each link which is used by this flow. Observe that the links of the first type are unordered, while the links of the second type are ordered (in the order in which the flow passes the individual links).

The architecture of the graph neural network is based on a two-stage message passing as described in Sec. III. The dimension of the hidden state vector space is 16. The message function is given by $M(h_v, h_w) = h_w$ for both stages, i.e. nodes simply send their hidden state to the nodes in their neighbourhood. The update function for updating states of links is a feed forward neural network with two hidden layers, each one consisting of 16 neurons with RELU activation functions. The update function for flow updates is a recurrent neural network consisting of GRU cells. The message passing is repeated for $T = 4$ times. The readout function is a feed

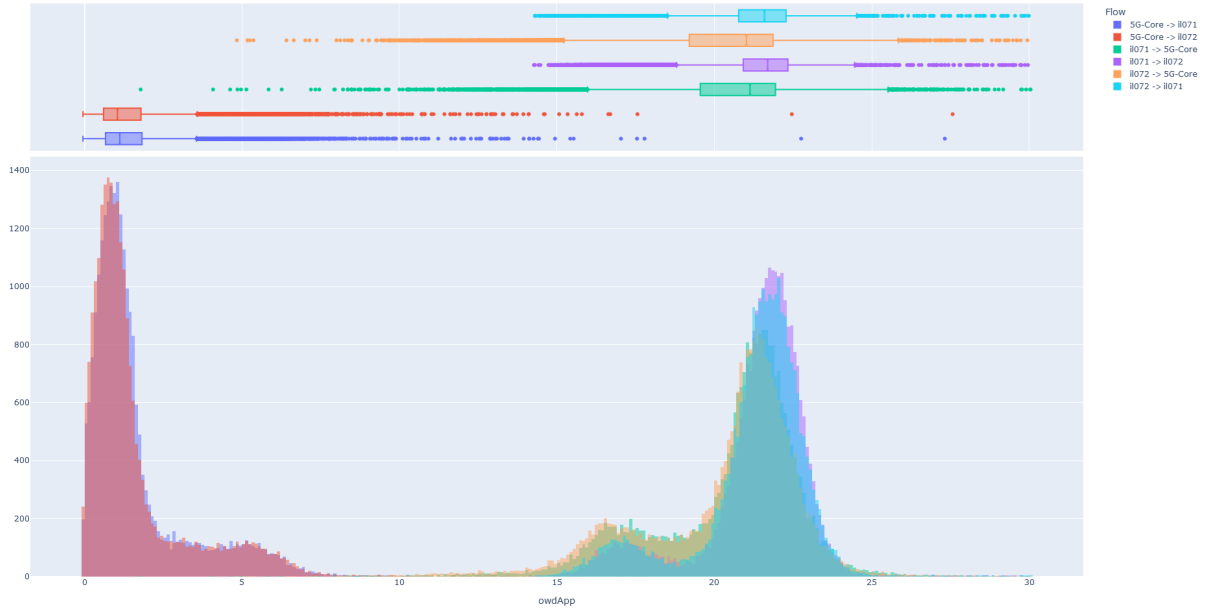

Fig. 3. Histograms of the measurement data of latency on application level in the 5G local network

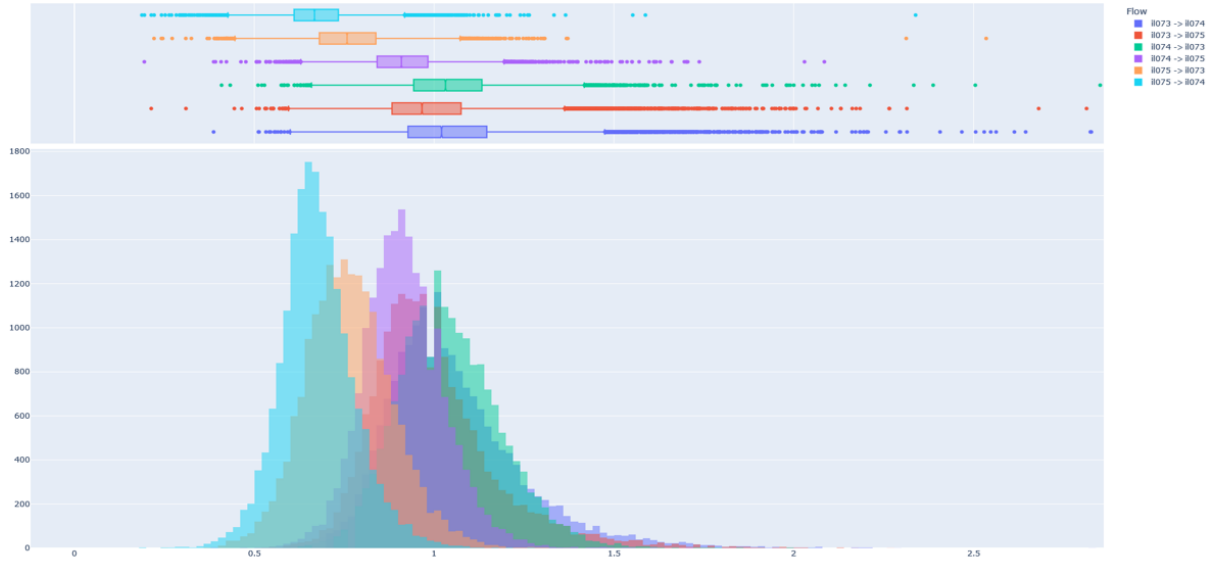

Fig. 4. Histograms of the measurement data of latency on application level in the WiFi local network

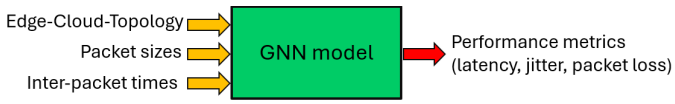

Fig. 5. Overview of the approach

forward neural network as well and with the same structure as the update function for links.

D. GNN Training

For the implementation of the GNN model, we used the iGNNition framework. iGNNITION is a TensorFlow-based framework for fast prototyping of GNNs. It provides a codeless programming interface, where users can implement their own GNN models in a YAML file. IGNNITION also incorporates a set of tools and functionalities that guide users during the design and implementation process of the GNN.

V. Evaluation

We evaluated the following scenarios: (A) One GNN model for each local edge network and (B) a common GNN model for both edge networks. For scenario (A), each model is only trained with the data from the corresponding edge network. For scenario (B), we use the whole data (both from the 5G network

and from the Wifi-Network) to train the model. In both scenarios, data was scaled in a preprocessing. The training of the two separate GNN models for each local network reached a satisfying performance for both latency and jitter prediction. For example, Fig. 6 shows the training process where the GNN model is trained to predict the latency on application level in the 5G network. For the other metrics we achieved similar results (both in the 5G and in the Wi-Fi network).

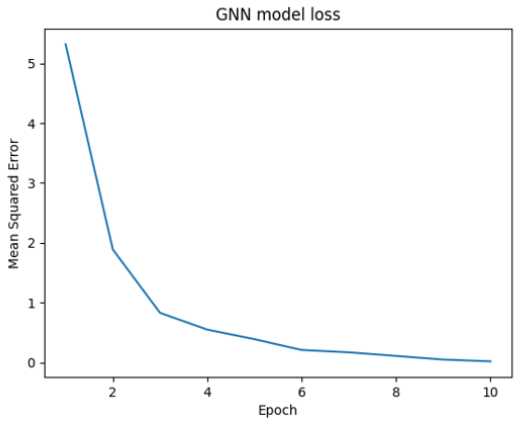

Fig. 6. Loss function of the GNN-model trained for predicting latency on application level in the local 5G network. The GNN-model for the local Wifi network achieved a similar performance.

However, as soon as we tried to train a joint model for both networks, we did not achieve sufficient performance (even with various attempts to tune the hyperparameters). Our explanation for this is that the distributions of the data in the two networks are too different (as can be observed for example from the graphs Fig. 3 and Fig. 4). Even scaling the data has not brought any improvement so far.

VI. Conclusion

This paper illustrates the significant potential of Graph Neural Networks as a powerful tool for performance prediction and network management in edge-cloud systems. By leveraging the inherent graph structure of these networks, GNNs provide accurate insights into critical metrics like latency and jitter, enabling more informed decision-making for resource allocation and optimization. The results underscore the promise of GNN-based approaches to advance the efficiency, reliability, and sustainability of edge-cloud infrastructure, ultimately contributing to more responsive and energy-efficient network environments. However, so far we have only achieved sufficient performance in a simple scenario (one machine learning model for each local network). We will continue to try to train a joint model for both local networks in the future. It would also be interesting to consider other metrics (such as energy efficiency) as well, or to consider more complex network scenarios.

References

[1] Guillermo Bernárdez, José Suárez-Varela, Albert López, Bo Wu, Shihan Xiao, Xiangle Cheng, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Is machine learning ready for traffic engineering optimization? *IEEE International Conference on Network Protocols (ICNP)*, 2021.

[2] Christof Brandauer and Thomas Fichtel. Miner-a measurement infrastructure for network research. In *2009 5th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities and Workshops*, pages 1–9. IEEE, 2009.

[3] Yueyue Dai, Xiaoyang Rao, Bruce Gu, Youyang Qu, Huiran Yang, and Yunlong Lu. Graph learning-based multi-user multi-task offloading in wireless computing power networks. *IEEE Internet of Things Journal*, 2025.

[4] Miquel Ferriol-Galmés, Jordi Paillisse, José Suárez-Varela, Krzysztof Rusek, Shihan Xiao, Xiang Shi, Xiangle Cheng, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Routenet-fermi: Network modeling with graph neural networks. *IEEE/ACM transactions on networking*, 31(6):3080–3095, 2023.

[5] Miquel Ferriol-Galmés, Krzysztof Rusek, José Suárez-Varela, Shihan Xiao, Xiang Shi, Xiangle Cheng, Bo Wu, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Routenet-erlang: A graph neural network for network performance evaluation. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, pages 2018–2027. IEEE, 2022.

[6] Wireshark Foundation. Wireshark: The world's foremost network protocol analyzer. https://www.wireshark.org/, 1998. Accessed: 2025-05-28.

[7] The iPerf Development Team. iperf3: A tcp, udp, and sctp network bandwidth measurement tool. https://iperf.fr/, 2014. Accessed: 2025-05-28.

[8] Kai Li, Wei Ni, Xin Yuan, Alam Noor, and Abbas Jamalipour. Deep-graph-based reinforcement learning for joint cruise control and task offloading for aerial edge internet of things (edgeiot). *IEEE Internet of Things Journal*, 9(21):21676–21686, 2022.

[9] Yihong Li, Xiaoxi Zhang, Tianyu Zeng, Jingpu Duan, Chuan Wu, Di Wu, and Xu Chen. Task placement and resource allocation for edge machine learning: A gnn-based multi-agent reinforcement learning paradigm. *IEEE Transactions on Parallel and Distributed Systems*, 34(12):3073–3089, 2023.

[10] Mike Muuss. The PING program. https://ftp.arl.army.mil/~mike/ping.html, 1983. Accessed: 2025-05-28.

[11] David Pujol-Perich, José Suárez-Varela, Miquel Ferriol, Shihan Xiao, Bo Wu, Albert Cabellos-Aparicio, and Pere Barlet-Ros. IGNNITION: Bridging the gap between graph neural networks and networking systems. *IEEE Network, in press*, 2021.

[12] Krzysztof Rusek, José Suárez-Varela, Paul Almasan, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Routenet: Leveraging graph neural networks for network modeling and optimization in sdn. *IEEE Journal on Selected Areas in Communications*, 38(10):2260–2270, 2020.