Received 10 October 2025; revised 10 November 2025; accepted 5 December 2025. Date of publication 9 December 2025;
date of current version 24 December 2025.

Digital Object Identifier 10.1109/OJCOMS.2025.3641933

MHNet: A Multi-Head GNN Architecture for Efficient Network Modeling

SANDUSHAN RANAWEERA (Student Member, IEEE), REN PING LIU (Senior Member, IEEE), YING HE (Senior Member, IEEE), AND BEESHANGA JAYAWICKRAMA (Senior Member, IEEE)

School of Electrical and Data Engineering, University of Technology Sydney, Sydney, NSW 2007, Australia

CORRESPONDING AUTHOR: S. RANAWEERA (e-mail: sandushan.ranaweera@uts.edu.au)

ABSTRACT In recent years, Graph Neural Networks (GNNs) have emerged as a powerful tool for network modeling due to their ability to learn complex relationships in graph-structured data. However, the existing GNN-based network models are designed to predict only a single performance metric at a time, leading to computational inefficiencies. Moreover, existing approaches lack effective methodologies for interpreting the learned relationships in a networking context. We present MHNet, a multi-head GNN architecture capable of simultaneously predicting delay, jitter, and packet loss in network traffic flows. To train MHNet, we propose an adaptive optimization strategy that constructs a balanced update direction to update the weights of the model at each epoch using the normalized gradients of the individual loss functions correspond to performance metric outputs. To interpret the relationships learned by the model in the network graph, we construct a gradient-based analysis framework that integrates networking domain knowledge to assess the influence of input features on the prediction outputs. Experimental results show that MHNet achieves prediction accuracy comparable to the state-of-the-art RouteNet model across all metrics, while reducing inference-stage Floating Point Operations (FLOPs) cost by 67%. The interpretation analysis further reveals that MHNet mitigates oversmoothing and selectively focuses on the most relevant substructures of the network feature graph when predicting performance metrics for traffic flows.

INDEX TERMS Graph neural networks, network modeling, computational efficiency, explainable AI.

I. INTRODUCTION

THE RAPID growth in Internet traffic, driven by the widespread adoption of streaming services, remote work, and IoT devices, has placed a significant strain on network operators. Operators face various challenges, such as rising operational costs, latency issues, and potential service disruptions, which can negatively impact customer satisfaction. Modeling computer networks has become essential for understanding their dynamic behaviors, an area of interest for researchers since the early days of the Internet [1]. Accurate network modeling underpins various network engineering tasks, such as infrastructure planning [2] and congestion control [3]. Moreover, it plays a pivotal role in advancing emerging paradigms like Knowledge Defined Networking (KDN) [4] and Network Digital Twins (NDT) [5].

Traditional network modeling approaches primarily rely on analytical methods based on Queuing Theory (QT) [6] and packet-level simulators [7]. However, with the rapid evolution of the Internet, modern computer networks have become significantly more complex, exhibiting intricate traffic characteristics [8], [9], [10], [11], [12], [13], [14], [15]. This complexity poses challenges in adapting traditional modeling techniques to modern day networks. Analytical modeling techniques based on QT assume that the packet arrival process follows a Poisson distribution, which often fails to accurately represent modern networks' traffic patterns. On the other hand, although packet-level simulators offer highly accurate network performance evaluations, they have substantial computational costs. These costs scale linearly with the number of packets forwarded, requiring the simulation of millions of events per second for links with Gbps-level capacities. This computational burden makes packet-level simulators impractical for simulating modern large-scale networks.

Researchers have turned to Machine Learning (ML) to address these challenges to develop data-driven methods

capable of predicting performance metrics with reduced computational complexity. Early efforts to model computer networks using ML techniques involved Fully Connected Neural Networks (FCNs) [16] and Recurrent Neural Networks (RNNs) [17]. However, FCN models require fixed input dimensions, which constrains their ability to generalize across network topologies with different numbers of links, queues, and traffic flows. In contrast, RNNs can capture the sequential dependencies between links/queues and traffic flows traverse them, yet they fail to learn the interrelationships among the traffic flows within the network.

In recent years, Graph Neural Networks (GNNs) [18] have demonstrated their ability to capture complex relationships from graph-based data structures in various domains, including computer vision, traffic engineering, natural language processing, recommender systems, and chemistry [19]. Given that computer networks can naturally be represented as graphs, GNN has emerged as a powerful tool for network modeling [20], [21], offering significant benefits to network engineering tasks. For instance, a GNN-based admission control algorithm for 5G networks [22] enables real-time decision-making and outperforms traditional Software Defined Networking (SDN) shortest path algorithms. Similarly, an actor-critic Deep Re-inforcement Learning (DRL) routing optimization algorithm leveraging GNNs has been proposed in [23] to improve routing efficiency. In [24], an attention-based transformer GNN architecture evaluates network failure scenarios and predicts their impact, facilitating the identification of critical failure cases and reducing the search space for robust network design optimization.

However, despite their promising performance, GNNs face significant limitations in computational efficiency. Specifically, the existing models require separate training for predicting different performance metrics, such as delay, jitter, and packet loss [25], leading to high computational costs during both training and inference. Computationally efficient ML models are crucial for reducing operational costs and lowering carbon footprints, thereby promoting environmentally sustainable solutions [26]. Additionally, the lack of interpretation methods for GNNs that integrate networking knowledge makes them difficult to interpret, limiting their transparency in network modeling. This underscores the need for interpretability frameworks that can explain how GNNs predict performance metrics by identifying the influence of specific network input features.

In this paper, we propose MHNet, a multi-head GNN architecture designed to simultaneously predict delay, jitter, and packet loss in parallel. Specifically, MHNet incorporates a shared Message Passing Neural Network (MPNN) to compute hidden state embeddings for traffic flows, links, and queues within the network. These hidden states are then processed by three distinct readout branches to predict the three performance metrics concurrently. MPNNs are computationally intensive due to their iterative and repetitive operations required for hidden state updates. By utilizing a common MPNN, the computational complexity of the MHNet model is significantly reduced. To address the multi-task learning challenge of predicting three different performance metrics simultaneously, we employ an adaptive optimization algorithm that constructs a descent direction for weight updates using normalized gradients.

The main contributions of this paper are summarized as follows:

- We propose MHNet, a multi-head GNN architecture to predict delay, jitter, and packet loss of traffic flows simultaneously, employing a common backbone MPNN model to learn hidden states for links, queues, and traffic flows in a network.
- We develop an adaptive optimization algorithm to train the MHNet model for the multi-task learning task of simultaneously predicting the three performance metrics. This algorithm is designed to compute a unified descent direction for weight updates in each epoch, thereby mitigating bias toward the gradients of individual loss functions associated with larger gradients.
- To interpret the relationships learned by the model, we develop a gradient-based analysis framework that assesses the influence of input features on the model's prediction outputs. This method integrates domain-specific networking knowledge, enabling meaningful interpretation of the learned dependencies within the context of computer networks.

We evaluated the proposed MHNet model using the publicly available dataset [27], which contains traffic data generated by the BNNetSimulator [28]. Simulation results demonstrate that MHNet achieves performance comparable to the RouteNet model [25] across all three predicted performance metrics. Additionally, complexity analysis shows that MHNet requires 67% fewer Floating Point Operations (FLOPs) compared to RouteNet-F when predicting these metrics. A gradient-based interpretation of the model's learned representations further indicates that MHNet effectively mitigates the oversmoothing problem and is capable of identifying mathematical relationships between input features and performance metrics by learning the dependencies among links and queues based on the traffic flows they carry.

The rest of this paper is organized as follows. Section II presents an overview of related works in modeling computer networks using GNNs and an overview of Multi Task Learning (MTL) in ML context; we present our system model for MHNet in Section III. In Section IV, we construct an adaptive optimization algorithm to train the MHNet model. We present our framework to interpret the learned representations of the MHNet model in Section V. Section VI presents a performance evaluation of MHNet, and Section VII concludes the paper.

II. BACKGROUND AND RELATED WORK

In this section, we first illustrate the limitations of existing works for robust network modeling using GNNs and predicting performance metrics. Then we provide an overview on MTL and its relevance to the MHNet model.

A. NETWORK MODELING USING GNNs

The use of GNNs in modeling computer networks has attracted much attention from the networking research community in recent years. A GNN method utilizing monoid objects to model networks and predict network delay measurements is proposed in [29]. Similarly, a graph transformer-based GNN model is presented in [21] to predict traffic flow delays. In this approach, the graph is constructed with traffic flows as nodes, while edges represent interactions among flows—i.e., two nodes are connected if their flows share a common link or queue. To enhance the model's performance, the authors proposed a network context-aware encoding method to incorporate topological information. While these GNN models achieve promising accuracy in predicting network performance metrics, they require extensive training data, which involves significant computational costs due to the need for data collection from real-world networks or resource-intensive packet-level simulations. To mitigate this, a contrastive loss-based training mechanism combined with a fine-tuning stage using limited data is proposed in [30].

The aforementioned works primarily focus on modeling network delay. However, computer networks involve multiple performance metrics, such as delay, jitter, and packet loss. To address this, a series of studies introduced the RouteNet GNN framework [20], [25], [31], which is designed to predict multiple network performance metrics. The latest version of this family is RouteNet-F [25], which employs a MPNN to capture relationships between links, queues, and flows within a network. Based on their specifications, the model initializes feature embeddings for each link, queue, and flow using an encoder network. Following initialization, MPNN performs a fixed number of message-passing iterations. A readout Neural Network (NN) then extracts the performance metrics from the final embeddings. The training data for RouteNet is generated using the OMNET++ network simulator [32], with network topologies based on combinations of 14-node NSF [33], 24-node Geant2 [34], Germany50 [35], and 17-node GBN [36]. Simulation results demonstrate that RouteNet achieves superior accuracy compared to an QT-based benchmark model.

Despite its strengths, RouteNet-F is designed to predict one performance metric at a time, necessitating the training of separate models for delay, jitter, and packet loss. This approach significantly increases computational costs during both training and inference. Additionally, the absence of interpretability techniques for GNN models in the context of network modeling limits their transparency and further complicates their use in this domain.

B. MULTI TASK LEARNING

MTL is an approach in which a model learns multiple tasks jointly. The proposed MHNet architecture belongs to the category of hard parameter sharing, where some parameters are shared across tasks while others remain task-specific. In particular, the backbone network parameters are shared with all prediction tasks and the readout network parameters are dedicated to the prediction task associated with each performance metric. Over the recent years, MTL has shown great success in various domains such as natural language processing [37], computer vision [38], etc. Although some recent studies have explored MTL with GNNs [39], they are not directly applicable to network modeling, which requires customized message passing to capture inherent relationships in the network, such as propagating information between link, flow, and queue features. To the best of our knowledge, our work is the first to apply MTL in the context of GNN-based network modeling.

The weighted sum approach is the most common approach used to construct a loss function for MTL problems. In the weighted sum approach, a single loss function is constructed as,

$$\mathcal{L}(\mathcal{W}, \mathcal{D}) = \sum_{k=1}^{N} \lambda_k \mathcal{L}_k(\mathcal{W}, \mathcal{D}), \tag{1}$$

where $\mathcal{W}$ denotes the weights of the NN, $\mathcal{D}$ denotes the data samples in the dataset, $\mathcal{L}_k$ denotes loss functions representing multiple objectives and λ_k denotes some positive scaling coefficients. When it comes to the training of ML models for MTL tasks, gradient-based algorithms are utilized as optimization algorithms. In general, the weights are updated in each training step as follows:

$$\mathcal{W}^{(t+1)} = \mathcal{W}^{(t)} - \gamma \boldsymbol{d}^{(t)}, \tag{2}$$

where $\mathcal{W}^{(t)}$ is a weight parameter at the t^{th} training step, γ is a step size and $-\boldsymbol{d}^{(t)}$ is some descent direction at the t^{th} training step. Usually, the descent direction is constructed using the gradient information. This makes selecting the scaling coefficients highly challenging [40], as they directly impact the gradient of the loss function, i.e.,

$$\nabla\mathcal{L}(\mathcal{W}, \mathcal{D}) = \sum_{k=1}^{N} \lambda_k \nabla\mathcal{L}_k(\mathcal{W}, \mathcal{D}). \tag{3}$$

The search for optimal coefficients is typically performed through an exhaustive grid search, which incurs significant computational cost during the training process. In Section IV, we analyze the effects of these scaling coefficients and devise an efficient training algorithm to construct a descent direction for optimizing the MHNet model for the multi-task learning problem of predicting delay, jitter, and packet loss for traffic flows, while balancing the scales of the gradients of the loss functions from individual prediction outputs.

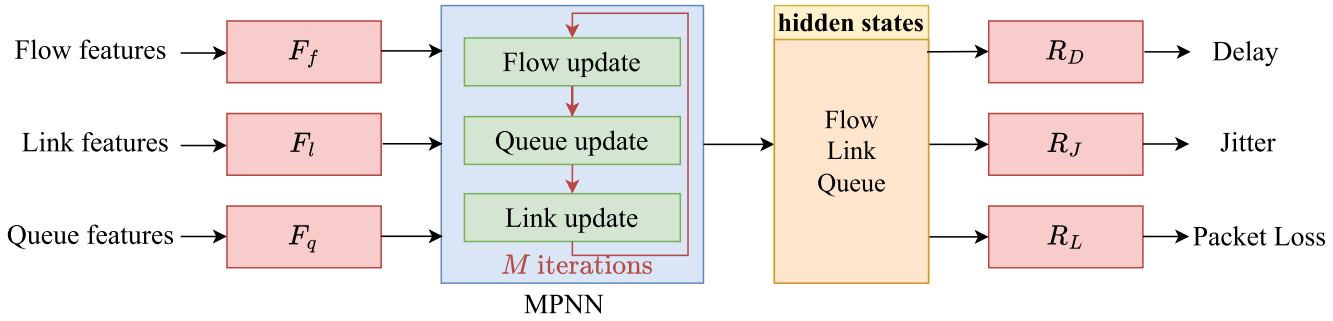


FIGURE 1. MHNet GNN model overview. The FCNs F_f, F_l, and F_q transform input feature vectors corresponding to flows, links and queues in to hidden states of a fixed dimension. These hidden states are iteratively updated using a three stage MPNN for M iterations. The three readout branches consisting of the readout networks (R_D, R_J, and R_L) map the final hidden states traffic flows to delay, jitter and packet loss predictions.

III. SYSTEM MODEL

MHNet constructs a GNN model to predict delay, jitter, and packet loss of traffic flows as a single model. Fig. 1 presents an overview of the MHNet GNN model. In MHNet, a computer network is modeled as a heterogeneous graph consisting of three types of nodes: traffic flows, queues and links. The edges of the heterogeneous graph denote the dependencies between the traffic flows, queues and links. For the notational convenience, we define a network as a collection of a set of traffic flows $\mathcal{F} = \{f_i : \ i \in (1, \ldots, n_f)\}$, a set of queues $\mathcal{Q} = \{q_i : \ i \in (1, \ldots, n_q)\}$, and a set of links $\mathcal{L} = \{l_i : \ i \in (1, \ldots, n_l)\}$. The number of flows, queues and links in the network are denoted by n_f,n_q, and n_l, respectively. We denote the tuples consisting of queues and links traversed by a traffic flow using $P(f_i)$. Specifically, $P(f_i) = \{(q_{i,1}, l_{i,1}), \ldots, (q_{i,K_i}, l_{i,K_i})\}$ where K_i denotes the path length of the flow f_i. Also, we denote all of the flows passing through the queue q_i using $Q_f(q_i)$ and $L_q(l_i)$ denotes all the queues injecting traffic to the link l_i.

MHNet implements a MPNN following [25] to learn hidden state embeddings for traffic flows, links, and queues by capturing dependencies between them. Intuitively, the state of a flow is determined by the states of the links and queues it traverses, while the state of a queue is influenced by the flows passing through it. Similarly, the state of a link is affected by the queues connected to it, creating an interdependent relationship among flows, queues, and links. After learning the hidden state embeddings, delay, jitter and packet loss for each traffic flow are predicted using three branches of readout NNs. The following sections describe the implementation of MHNet architecture to model computer networks and predict their performance metrics.

A. INITIALIZING HIDDEN STATES

When using a GNN to model relationships between links, queues and flows, it is essential to give their characteristics to the GNN as inputs. MHNet uses the encoding schemes below to construct feature vectors for links, queues, and flows.

$$\boldsymbol{x}_l = \langle \ell, \mathbf{1}_s \rangle \text{ for links,} \tag{4a}$$
$$\boldsymbol{x}_q = \langle b, \mathbf{1}_p, w \rangle \text{ for queues,} \tag{4b}$$
$$\boldsymbol{x}_f = \langle t, \boldsymbol{v}_d \rangle \text{ for flows,} \tag{4c}$$

where ℓ denotes the average link load, $\mathbf{1}_s$ denotes a one-hot vector corresponds to the scheduling policy at the output port of the link, b denotes the buffer size of the queue, $\mathbf{1}_p$ a one-hot vector corresponds to the priority of the queue, w denotes the weight of the queue to handle Weighted Fair Queuing or Deficit Round Robin configurations, t denotes the average traffic volume of the flow, and $\boldsymbol{v}_d$ denotes a vector of parameters specific to the traffic distribution which specifies the distribution of inter-arrival times of the packets. A FCN is used to map $\boldsymbol{x}_l, \boldsymbol{x}_q$ and $\boldsymbol{x}_f$ into their corresponding hidden state space, i.e.,

$$\boldsymbol{h}_l^{(0)} = F_l(\boldsymbol{x}_l), \tag{5a}$$
$$\boldsymbol{h}_q^{(0)} = F_q(\boldsymbol{x}_q), \tag{5b}$$
$$\boldsymbol{h}_f^{(0)} = F_f(\boldsymbol{x}_f), \tag{5c}$$

where F_l, F_q, and F_f are denoting FCNs corresponding to links, queues, and flows respectively.

B. IMPLEMENTING MPNN

MHNet leverages a MPNN to capture the intricate dependencies among traffic flows, network links, and queues. It models the following mathematical relationship between the hidden states: $\boldsymbol{h}_{f_i}, \boldsymbol{h}_{q_i}$ and $\boldsymbol{h}_{l_i}$.

$$\boldsymbol{h}_{f_i}^{(m+1)} = U_f\left(\boldsymbol{h}_{f_i}^{(m)}, \left\{\boldsymbol{h}_q^{(m)} || \boldsymbol{h}_l^{(m)}; (q, l) \ \in P(f_i)\right\}\right), \tag{6a}$$
$$\boldsymbol{h}_{q_i}^{(m+1)} = U_q\left(\boldsymbol{h}_{q_i}^{(m)}, \sum_{\forall f_i \in Q_f(q_i)} \boldsymbol{y}_{f_i,q_i}^{(m)}\right) \tag{6b}$$
$$\boldsymbol{h}_{l_i}^{(m+1)} = U_l\left(\boldsymbol{h}_{l_i}^{(m)}, \left\{\boldsymbol{h}_q^{(m+1)}; q \in L_q(l_i)\right\}\right), \tag{6c}$$

where U_f, U_q, and U_l are RNN layers are constructed from Gated Recurrent Unit (GRU) cells to implement the functions to update hidden states, m is the message passing iteration index which increments to 0 to $M-1$, and M is the total number of message passing iterations. $\boldsymbol{y}_{f_i,q}^{(m)}$ denotes the output corresponding to $\boldsymbol{h}_q^{(m)} || \boldsymbol{h}_l^{(m)}$ from the RNN layer U_f for the flow f_i, and $||$ denotes the concatenation operation. A graphical illustration of implementing U_f, U_q, and U_l is presented in Fig. 2. This message-passing mechanism can flexibly handle graphs with varying numbers of flows,

queues, and links, as well as different topological relations among them. By using GRU-based recurrent layers, the update functions can naturally process input sequences of different lengths, allowing MHNet to capture dependencies across paths of varying lengths and complex queue–link relationships. This ensures that the model remains scalable and adaptable to diverse network scenarios without requiring structural modifications.

A single message passing iteration corresponds to executing the steps outlined in Equations (6a), (6b), and (6c) sequentially. Specifically, (6a) runs for all traffic flows f_i, followed by running (6b) for all queues q_i. Finally, the step (6c) runs for all links l_i. Although the three update phases are executed sequentially to preserve the dependency structure among flows, queues, and links, each phase is inherently parallelizable. Specifically, the GRU updates for all flows in (6a), all queues in (6b), and all links in (6c) can be performed concurrently within their respective sets using batch inference. While designing customized dependency-aware scheduling strategies for the computational graph of the model to pipeline these updates could further improve inference latency, such optimization lies beyond the scope of this paper.

When predicting performance metrics, MHNet runs multiple message-passing iterations to capture dependencies between the flows, links, and queues in the network. The number of message-passing iterations M is treated as a configurable hyperparameter during the training stage.

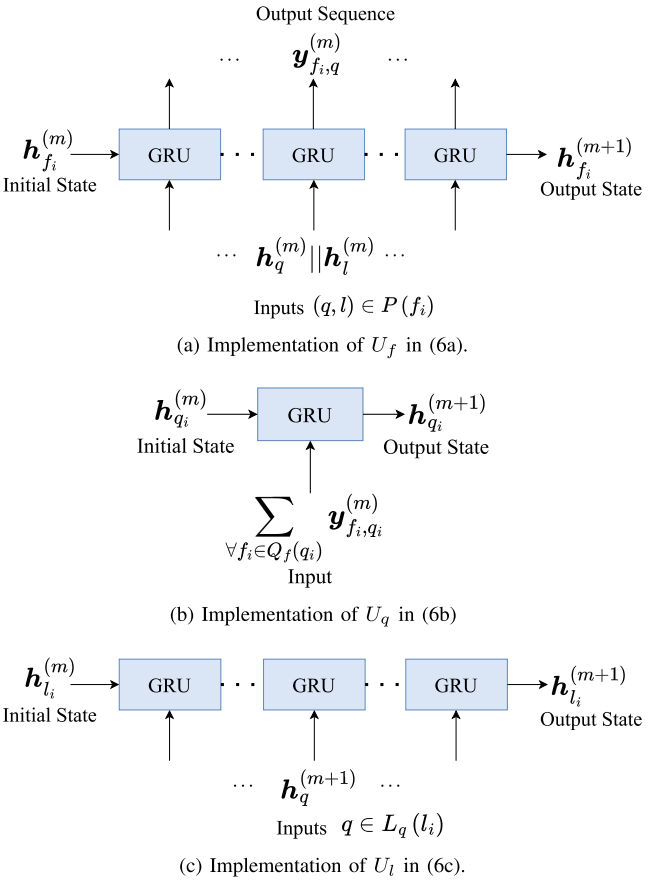


FIGURE 2. Illustration of implementing U_f, U_q, and U_l in (6).

C. IMPLEMENTING READOUT NETWORKS

In classical network modeling, end-to-end delay is expressed as the sum of propagation, transmission, queuing, and processing delays [41]. Following [25], [30], we omit propagation and processing delays as they are negligible, and we model delay as the summation of transmission and queuing delays, jitter as the variation in queuing occupancy, and packet loss directly predicted from the flow hidden states.

To predict the average delay, jitter, and packet loss of traffic flows using the learned hidden state embeddings, MHNet implements three FCNs. Unlike in RouteNet [25], where only one performance metric is predicted by the single readout network, MHNet has three branches of readout networks to predict all three performance metrics using the hidden state embeddings learned by a common MPNN backbone network. The operation of the readout networks can be expressed mathematically as follows.

$$D_i = \sum_{\forall (l,q) \in P(f_i)} \frac{\mathcal{S}_{f_i}}{\kappa_l} + \frac{R_D\left(\boldsymbol{y}_{f_i,q}^{(M-1)}\right)}{\kappa_l}, \tag{7a}$$

$$J_i = \sum_{\forall (l,q) \in P(f_i)} \frac{R_J\left(\boldsymbol{y}_{f_i,q}^{(M-1)}\right)}{\kappa_l}, \tag{7b}$$

$$L_i = R_L\left(\boldsymbol{h}_{f_i}^{(M)}\right), \tag{7c}$$

where κ_l denotes link capacity of link l, M denotes the total number of message passing iterations and $\mathcal{S}_{f_i}$ denotes the average packet size of the traffic flow f_i. R_D, R_J, and R_L denote FCNs, which correspond to delay, jitter, and packet loss readout networks. The summation in the delay prediction expression consists of two terms in (7a). The first term $\frac{\mathcal{S}_{f_i}}{\kappa_l}$ corresponds to the average transmission delay of a packet in a link while the second term $\frac{R_D(\boldsymbol{y}_{f_i,q}^{(M-1)})}{\kappa_l}$ corresponds to the average queuing delay. Specifically, the prediction from delay readout network R_D serves as the average queue occupancy, and it is divided by the capacity κ_l of the link connected to the output port of the queue to get the average queuing delay. The term resulting from the jitter readout network $R_J(\boldsymbol{y}_{f_i,q}^{(M-1)})$ serves as the average variation of queuing occupancy, and it is divided by the capacity κ_l of the link connected to the output port of the queue to get the delay variance. The output layers of R_D and R_J are having Rectified Linear Unit (ReLU) activation function, i.e.,

$$r(x) = \max(0, x), \tag{8}$$

to ensure positive output for delay and jitter predictions. For packet loss predictions, the output of R_L is taken directly.

In R_L, the sigmoid activation function, i.e.,

$$\sigma(x) = \frac{1}{1+e^{-x}}, \tag{9}$$

is applied to the last layer to ensure the output lies between 0 and 1. We use the ReLU activation function for all internal readout network layers.

IV. TRAINING ALGORITHM

The training process of MHNet presents a multi-objective optimization problem, as it involves simultaneously optimizing multiple objective functions. Specifically, there are three distinct loss functions for three performance metric predictions. Let us begin by constructing the loss functions for the model.

A. CONSTRUCTION OF THE LOSS FUNCTIONS

The three predictions of the MHNet model have different characteristics. For example, packet loss can be expressed as a number between 0 and 1, while delay and jitter are often expressed in time units. Considering the nature of these measurements, we use the Mean Absolute Percentage Error (MAPE) loss function, i.e.,

$$\mathcal{L}_{\text{MAPE}} = \left| \frac{y_{\text{pred}} - y_{\text{true}}}{y_{\text{true}}} \right| \times 100, \tag{10}$$

for the delay and jitter predictions. The MAPE loss function is employed for delay and jitter predictions because it quantifies the prediction error in relative terms, normalizing the absolute difference between predicted and true values by the ground truth value. We chose Binary Cross Entropy (BCE) loss function, i.e.,

$$\mathcal{L}_{\text{BCE}} = -\big[y_{\text{true}} \log\big(y_{\text{pred}}\big) + (1 - y_{\text{true}}) \log\big(1 - y_{\text{pred}}\big)\big], \tag{11}$$

for the packet loss prediction. The BCE loss function is used for packet loss prediction since average packet loss can be interpreted as the parameter of a Bernoulli distribution, making BCE a natural choice to measure the difference between predicted and observed packet loss values.

We can combine these three loss functions into a single loss function using the weighted sum approach as below.

$$\begin{aligned}\mathcal{L}(\mathcal{W}, \mathcal{D}) = {} & \lambda_D \mathcal{L}_D(\mathcal{W}, \mathcal{D}) \\ & + \lambda_J \mathcal{L}_J(\mathcal{W}, \mathcal{D}) \\ & + \lambda_L \mathcal{L}_L(\mathcal{W}, \mathcal{D}),\end{aligned} \tag{12}$$

where $\mathcal{W}$ is all the weights of the MHNet model. The batch of input data samples is denoted by $\mathcal{D}$. The functions $\mathcal{L}_D$, $\mathcal{L}_J$, and $\mathcal{L}_L$ represent the individual loss terms for delay, jitter, and packet loss prediction outputs. The positive coefficients λ_D, λ_J, and λ_L serve as balancing factors, adjusting the contribution of each loss to account for differences in scale.

B. OPTIMIZING THE MODEL WEIGHTS

During the training process of any NN, the weights are updated using the gradient of the loss function. In general, we can write the parameter update step of a NN during the training stage as

$$\mathcal{W}^{(t+1)} = \mathcal{W}^{(t)} - \gamma \boldsymbol{d}^{(t)}, \tag{13}$$

where $\boldsymbol{d}^{(t)}$ denotes some descent direction devised from the gradient information.

In the MHNet model, the gradient of the loss function with respect to the parameters $\mathcal{W}$ of the model consists of three parts, each coming from the loss functions of three performance metric predictions, i.e.,

$$\begin{aligned}\nabla_{\mathcal{W}} \mathcal{L}(\mathcal{W}, \mathcal{D}) = {} & \lambda_D \nabla_{\mathcal{W}} \mathcal{L}_D(\mathcal{W}, \mathcal{D}) \\ & + \lambda_J \nabla_{\mathcal{W}} \mathcal{L}_J(\mathcal{W}, \mathcal{D}) \\ & + \lambda_L \nabla_{\mathcal{W}} \mathcal{L}_L(\mathcal{W}, \mathcal{D}).\end{aligned} \tag{14}$$

Since the parameter update direction during training is determined by the gradient of (12), which combines the gradients of the individual loss functions $\mathcal{L}_D$, $\mathcal{L}_J$, and $\mathcal{L}_L$ through their respective scaling factors, the choice of these factors has a direct impact on the optimization process of model parameters [42, Sec. 9.2]. For example, assigning equal weights to all three objectives can cause the update direction to be dominated by the losses with larger gradient norms, thereby degrading overall performance. Hence, it is essential to appropriately adjust the weights so that the gradients from delay, jitter, and packet loss objectives contribute in a balanced manner to model parameter updates.

We use Algorithm 1 to train the MHNet model by dynamically assigning the weighting coefficients λ_D, λ_J, and λ_L in each training epoch using the gradient information. First, we calculate the gradients of individual loss functions with respect to the parameters $\mathcal{W}$ of the model (line 2 of Algorithm 1). Then, we calculate the norm G of the gradient vector obtained by summing the gradients of the three individual loss functions (line 3 of Algorithm 1). Next, the norms of the gradients of the individual loss functions, corresponding to delay, jitter, and packet loss predictions, are calculated (lines 4-6 in Algorithm 1). Afterwards, we assign the values for the scaling coefficients λ_D, λ_J, and λ_L by dividing G by the gradient norms of the individual loss functions, corresponding to delay, jitter, and packet loss, respectively (lines 7–9 of Algorithm 1). Then, the descent direction is computed by taking the weighted sum of the gradients of the individual loss functions with respect to all of the parameters of the model using these calculated coefficients. $d_w^{(t)}$ denotes the component of the descent direction vector $\boldsymbol{d}^{(t)}$ corresponding to the weight parameter w in the t^{th} training step. The weights of the model are updated using the Adam algorithm [43] using the resulting descent direction (lines 11 - 17 of Algorithm 1). The constants β_1, β_2 are treated as training hyperparameters, γ is the step size, and $\epsilon \approx 10^{-7}$ is a small positive constant used to ensure numerical stability of the algorithm.

Algorithm 1 Optimization Algorithm

Input: $m_w^{(0)} = 0, v_w^{(0)} = 0\ \forall w \in \mathcal{W}$; $\mathcal{W}^{(0)}$; $\mathcal{T} > 0$; $\beta_1, \beta_2 \in [0, 1)$; $\epsilon \approx 10^{-7}$; $\gamma > 0$
Output: $\mathcal{W}^{(*)}$

repeat
  Calculate $\nabla_{\mathcal{W}} \mathcal{L}_y\big(\mathcal{W}^{(t)}, \mathcal{D}^{(t)}\big)\ \forall y \in \{D, J, L\}$
  $G \leftarrow \Big\| \sum_{y \in \{D,J,L\}} \nabla_{\mathcal{W}} \mathcal{L}_y\big(\mathcal{W}^{(t)}, \mathcal{D}^{(t)}\big) \Big\|$
  $G_D = \big\| \nabla_{\mathcal{W}} \mathcal{L}_D\big(\mathcal{W}^{(t)}, \mathcal{D}^{(t)}\big) \big\|$
  $G_J = \big\| \nabla_{\mathcal{W}} \mathcal{L}_J\big(\mathcal{W}^{(t)}, \mathcal{D}^{(t)}\big) \big\|$
  $G_L = \big\| \nabla_{\mathcal{W}} \mathcal{L}_L\big(\mathcal{W}^{(t)}, \mathcal{D}^{(t)}\big) \big\|$
  $\lambda_D \leftarrow \frac{G}{G_D}$ **if** $G_D \neq 0$ **else** 0;
  $\lambda_J \leftarrow \frac{G}{G_J}$ **if** $G_J \neq 0$ **else** 0;
  $\lambda_L \leftarrow \frac{G}{G_L}$ **if** $G_L \neq 0$ **else** 0;
  $\boldsymbol{d}^{(t)} = \sum_{y \in \{D,J,L\}} \lambda_y \nabla_{\mathcal{W}} \mathcal{L}_y\big(\mathcal{W}^{(t)}, \mathcal{D}^{(t)}\big)$
  for each $w \in \mathcal{W}$ **do**
    $m_w^{(t)} \leftarrow \beta_1 m_w^{(t-1)} + (1 - \beta_1) d_w^{(t)}$;
    $v_w^{(t)} \leftarrow \beta_2 v_w^{(t-1)} + (1 - \beta_2) \Big(d_w^{(t)}\Big)^2$;
    $\hat{m}_w^{(t)} \leftarrow \frac{m_w^{(t)}}{1 - \beta_1^t}$;
    $\hat{v}_w^{(t)} \leftarrow \frac{v_w^{(t)}}{1 - \beta_2^t}$;
    $w^{(t+1)} \leftarrow w^{(t)} - \gamma \frac{\hat{m}_w^{(t)}}{\sqrt{\hat{v}_w^{(t)}} + \epsilon}$;
  end for
  $t \leftarrow t + 1$
until $t \geq \mathcal{T}$

In each training step, Algorithm 1 performs a single forward pass followed by three backward passes. During the forward pass, the model predicts all three performance metrics and computes their respective losses. In the subsequent backward passes, the gradients for each individual loss are back propagated through the network to calculate the gradients with respect to the weights of the model. Despite involving three backward passes, the overall computational complexity remains lower than training three separate models, each of which would require its own set of three forward and backward passes. Additionally, the use of an adaptive weighting mechanism eliminates the need for grid search over the scaling coefficients λ_D, λ_J, and λ_L, further reducing the computational cost during training. At inference time, the model needs only a single forward pass to predict all performance metrics, making it substantially more efficient than employing separate MPNNs for each metric.

V. INTERPRETING THE LEARNED RELATIONSHIPS

In this section, we construct a gradient-based interpretation scheme to interpret the learned relationships by MHNet. Beyond assessing its predictive accuracy, it is crucial to analyze and interpret the model's ability to capture the underlying relationships between network entities. From a mathematical standpoint, MHNet can be seen as a differentiable function, which allows us to leverage gradient information to analyze how input features influence the model's predictions. By examining the gradients of the output with respect to the input features, we can identify which inputs the model relies on most heavily when making predictions [44]. This type of analysis has been widely used in applications such as image classification, where visualizing gradient-based saliency maps helps reveal which regions of an image most impact model predictions [45]. In the context of modeling networks with GNNs, we expect the predictions of delay, jitter, and packet loss for a given traffic flow to primarily depend on the specific links and queues it traverses, consistent with our intuition on the behavior of networks. Additionally, the performance metrics of a flow may be influenced by other flows that share common links, which are referred to as neighboring flows.

A. ANALYZING THE IMPACT OF INPUT FEATURES ON MODEL PREDICTIONS

Let y_i be a performance metric prediction output of the model for some traffic flow f_i. We can calculate the gradient of this output y_i with respect to any input feature vector $\boldsymbol{x} \in \mathbb{R}^n$ to the model, i.e.,

$$\nabla_{\boldsymbol{x}} y_i = \left\langle \frac{\partial y_i}{\partial x_1}, \ldots, \frac{\partial y_i}{\partial x_n} \right\rangle, \tag{15}$$

by back propagating the gradients to the input feature vectors. Let $\boldsymbol{u} \in \mathbb{R}^n$ be a unit vector. We can get the directional derivative in the direction of $\boldsymbol{u}$ as

$$\nabla_{\boldsymbol{x}}^{\boldsymbol{u}} y_i = \nabla_{\boldsymbol{x}} y_i \cdot \boldsymbol{u} = \|\nabla_{\boldsymbol{x}} y_i\| \cos\phi, \tag{16}$$

where $\|\cdot\|$ denoting the ℓ^2 norm. ϕ denotes the angle between the gradient vector $\nabla_{\boldsymbol{x}} y_i$ and the unit vector $\boldsymbol{u}$. Therefore, for any direction vector $\boldsymbol{u}$,

$$\big|\nabla_{\boldsymbol{x}}^{\boldsymbol{u}} y_i\big| \leq \|\nabla_{\boldsymbol{x}} y_i\|, \tag{17}$$

where $|\cdot|$ denotes the absolute value. In other words, the rate at which the output prediction changes with respect to variations in the feature vector $\boldsymbol{x}$ is bounded by $||\nabla_{\boldsymbol{x}} y_i||$. Therefore, input features that strongly influence the model's predictions can be identified by examining feature vectors with larger gradient norms.

B. COMBINE WITH NETWORKING KNOWLEDGE

We denote the set of links traversed by a traffic flow f_i as $L(f_i)$, and the set of neighboring flows of f_i as $N_f(f_i)$. The neighboring links $N_l(f_i)$ are defined as the links traversed by the flows in $N_f(f_i)$ but not by f_i itself. For each flow f_i in the network, we compute the gradient norms of the feature vectors corresponding to the model's predictions of delay, jitter, and packet loss.

Intuitively, we expect the features corresponding to the links in $L(f_i)$, as well as those associated with the flows in $N_f(f_i)$, to exhibit higher gradient norms. Moreover, the features of flow f_i and its neighboring flows $N_f(f_i)$ are anticipated to have higher gradient magnitudes, consistent with this intuition. Based on this intuition, we define the following numerical metrics to assess the model's ability to

capture meaningful relationships between the input features and the predicted performance metrics.

$$\mathrm{IoU}_f = \frac{\mathcal{N}\big(G_f(f_i) \cap T_f(f_i)\big)}{\mathcal{N}\big(G_f(f_i) \cup T_f(f_i)\big)}, \tag{18}$$

$$\mathrm{IoU}_l = \frac{\mathcal{N}(G_l(f_i) \cap T_l(f_i))}{\mathcal{N}(G_l(f_i) \cup T_l(f_i))}, \tag{19}$$

where $T_f(f_i) = \{f_i\} \cup N_f(f_i)$, $T_l(f_i) = L(f_i) \cup N_l(f_i)$, $\mathcal{N}(\cdot)$ denotes the number of elements in a set, $\cap$ represents the intersection operator, and $\cup$ denotes the union operator. The set $G_f(f_i)$ contains the top $\mathcal{N}(T_f(f_i))$ flows with the highest gradient norms, while $G_l(f_i)$ includes the top $\mathcal{N}(T_l(f_i))$ links with the highest gradient norms. Logically, a higher value of IoU_f indicates that the features of flow f_i and its neighboring flows contribute significantly to the model's prediction outputs for f_i. Similarly, a higher IoU_l suggests that the model's predictions for performance metrics related to f_i are primarily influenced by the links in $L(f_i)$ and those in $N_l(f_i)$, aligning with our intuition based on networking knowledge.

The IoU_f and IoU_l metrics quantify the extent to which the sets of links and flows, identified based on gradient information and topological information, respectively, overlap with one another. To obtain a more quantitative assessment of feature importance, we define the following two metrics. This is based on the observation that $\|\nabla_{\boldsymbol{x}} y_i\|$ can be interpreted as an importance score, reflecting the degree to which the model depends on the feature vector $\boldsymbol{x}$ when predicting the performance metrics.

$$\Gamma_f = \frac{\sum_{\forall f \in T_f(f_i)} \left\| \nabla_{\boldsymbol{x}_f} y_i \right\|}{\sum_{\forall f \in \mathcal{F}} \left\| \nabla_{\boldsymbol{x}_f} y_i \right\|} \tag{20}$$

$$\Gamma_l = \frac{\sum_{\forall l \in T_l(f_i)} \left\| \nabla_{\boldsymbol{x}_l} y_i \right\|}{\sum_{\forall l \in \mathcal{L}} \left\| \nabla_{\boldsymbol{x}_l} y_i \right\|} \tag{21}$$

In these metrics, values closer to 1 suggest that the gradients of the features corresponding to links and flows in $T_l(f_i)$ and $T_f(f_i)$ dominate when the model makes predictions. Collectively, these four metrics offer a comprehensive view of how input features influence the model's outputs.

VI. PERFORMANCE EVALUATION

We conduct a comprehensive evaluation of the proposed MHNet model in terms of both prediction accuracy and computational efficiency. Its performance is compared against several baseline models as well as ablated variants of MHNet. The rest of this section details the simulation setup and presents the corresponding experimental results.

A. EXPERIMENT SETUP

MHNet was implemented in Python with TensorFlow, employing 32-bit floating-point precision for all computations. The functions F_l, F_q, and F_f are implemented as FCNs, each consisting of one hidden layer with 32 units and an output layer with 32 units. The readout networks R_D, R_J, and R_L are implemented as FCNs, composed of two hidden layers with 16 units each and a final output layer with one unit.

The learning rate γ in Algorithm 1 is set to 0.001, with parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$ following the default parameter settings used in the original Adam paper [43]. The ϵ is set to the Tensorflow backend default of $\epsilon = 10^{-7}$. The dimensions of the hidden states and the number of message passing iterations are treated as hyperparameters and set using a grid search. Specifically, the hidden states, namely $\boldsymbol{h}_{f_i}, \boldsymbol{h}_{q_i}$, and $\boldsymbol{h}_{l_i}$, are configured to have 32 units. The MPNN is set to perform $M = 8$ message passing iterations.

To train and evaluate MHNet, we employ the publicly available dataset from [27], which was generated using the BNNetSimulator [28]. This dataset was carefully designed to approximate the characteristics of real Internet traffic and includes five sub-datasets, each corresponding to a distinct traffic distribution [31]: constant bitrate, on–off, autocorrelated exponentials, modulated exponentials, and a mixed combination of all traffic types. The inclusion of autocorrelated and modulated exponential traffic distributions is particularly important, as these have been shown to closely capture the self-similar properties commonly observed in Internet traffic traces [46]. Furthermore, the mixed-traffic scenarios provide additional diversity and complexity, enhancing the realism of the dataset. It is worth noting that this dataset [27] has been widely adopted in recent works on network modeling [20], [25], [30], [31], [47], further demonstrating its relevance and acceptance as a benchmark for learning-based performance prediction. The training set is generated using the NSF network topology with 14 nodes [33] and the Geant2 network topology with 24 nodes [34], while the test set is constructed using the GBN network topology with 17 nodes [36], which remains unseen during training.

1) ROUTENet-F [25]

This trains separate MPNNs for each metric prediction. We set all training hyperparameters as mentioned in the original paper to train this model.

2) MHNet-U

In this baseline, the MHNet model is trained with equal scaling coefficients applied to the three output-specific loss functions, i.e., $\lambda_D = \lambda_J = \lambda_L = 1$. This configuration acts as an ablated variant of our full model, allowing us to assess the benefits of dynamically adjusting these weights through Algorithm 1.

3) RNN

RNNs are used in some works as a way of predicting end-to-end network performance [48]. In this baseline implementation, we only implement the function U_f using GRU layer to update the hidden states of traffic flows. The performance metrics are predicted using the outputs of U_f using the readout networks as described in (7a), (7b) and (7c).

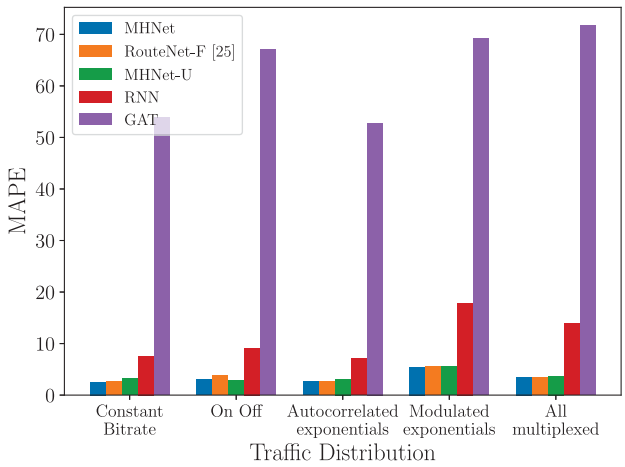


FIGURE 3. Comparison of accuracy in predicting delay of traffic flows.

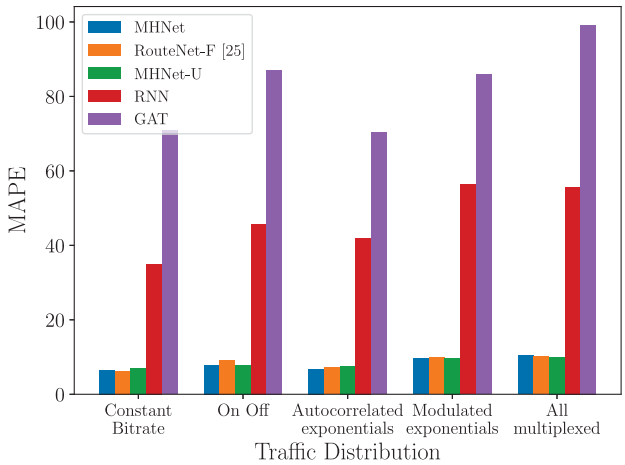


FIGURE 4. Comparison of accuracy in predicting jitter of traffic flows.

4) GAT

In this baseline, a shared backbone utilizing attention layers for message passing, inspired by the Graph Attention Network (GAT) [49] GNN architecture, is implemented to update the hidden states of the traffic flows, links, and queues. The three performance metrics are directly read from the hidden states of the traffic flows using three readout networks.

B. PREDICTION ACCURACY

In this section, we analyze the prediction accuracy of the MHNet model. The model is trained for 100 epochs for each traffic distribution, where 2000 samples are fetched to the model in each epoch. Fig. 3, Fig. 4, and Fig. 5 show the comparison of prediction errors of the predictions of the MHNet model and the baseline models. For delay and jitter, prediction error is shown as MAPE (10). The prediction error of packet losses is shown as Mean Absolute Error (MAE) value, i.e.,

$$\mathcal{L}_{MAE} = \left\| y_{\text{pred}} - y_{\text{true}} \right\|, \tag{22}$$

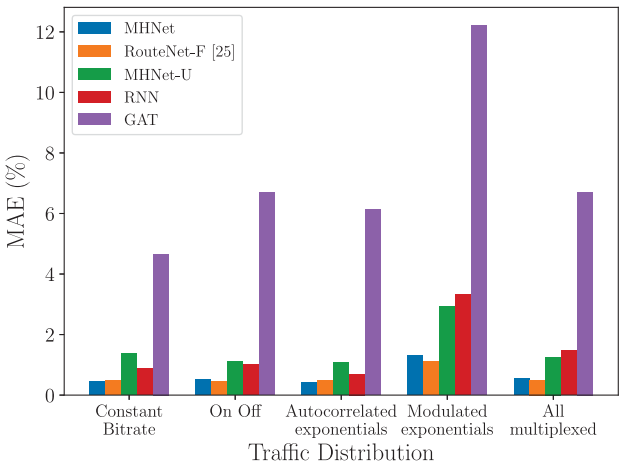


FIGURE 5. Comparison of accuracy in predicting packet loss of traffic flows.

where we scale the model predictions by multiplying by 100 to get MAE as a percentage.

The MHNet model outperforms or matches the RouteNet-F [25] in predicting all metrics. It is important to note that RouteNet-F trains a separate end-to-end GNN model for each metric, whereas MHNet uses a single model to predict all metrics simultaneously. The RNN model performs poorly on all metrics, highlighting its inability to learn the relationships among the states of links, queues, and flows. The GAT baseline shows the highest errors in predicting all metrics among all models. This result highlights the importance of customizing the GNN architecture to capture the interdependencies within network element states. The MHNet-U model, an ablated variant of our framework with equal weights across the three loss terms for performance metric prediction, achieves accuracy comparable to our approach in predicting delay and jitter but exhibits a pronounced degradation in packet loss prediction, even performing worse than the RNN baseline under certain traffic models. This shortcoming stems from its training procedure, where the use of equal weights ($\lambda_D = \lambda_J = \lambda_L = 1$) causes gradient updates to be dominated by the loss components of delay and jitter, thereby underrepresenting packet loss. This observation underscores the necessity of balancing gradient contributions across the three prediction objectives, a challenge explicitly addressed by Algorithm 1. When trained with Algorithm 1 (depicted in blue in Fig. 3, Fig. 4, and Fig. 5), the MHNet model achieves accuracies on par with RouteNet-F [25] across all performance metrics.

We also calculate the coefficient of determination (R^2) value defined as,

$$R^2 = 1 - \frac{\sum \left(y_{\text{pred}} - y_{\text{true}}\right)^2}{\sum \left(y_{\text{true}} - \bar{y}_{\text{true}}\right)^2} \tag{23}$$

to evaluate how well the predictions of the MHNet model align with the ground truth data from the dataset. In (23), $\bar{y}_{\text{true}}$ denotes the mean value of the ground values. The value of R^2 reflects the proportion of variance in the target performance

TABLE 1. R^2 values for model predictions for performance metrics.

Traffic distribution	R^2		
	Delay	Jitter	Packet Loss
Constant bitrate	0.9959	0.9609	0.9415
On-off	0.9951	0.9722	0.9326
Autocorrelated exp.	0.9967	0.9689	0.9387
Modulated exp.	0.9863	0.9739	0.9362
All multiplexed	0.9947	0.9752	0.9826

metric that the model successfully explains [50]. Table 1 shows the R^2 values for the performance predictions on the test set. The R^2 values exceed 0.93 in all cases, indicating that the predictions of MHNet explain over 93% of the variance in ground truth value in all cases.

It is worth noting that the test topology, GBN, is entirely unseen during training, as the training set includes only the NSFNet and Geant2 topologies. Nevertheless, MHNet demonstrates strong generalization capability across topologies, indicating its robustness in learning transferable structural and functional dependencies of network behavior. However, the model should be retrained when link capacities deviate significantly from those represented in the training data, as this is a general limitation inherent to all learning-based methods that encounter out-of-distribution numerical values [51], [52]. In practice, we can design training datasets that include the expected link bandwidths of the target network scenarios, as we typically have prior knowledge of the range of link capacities and bandwidth configurations encountered in networks.

C. COMPUTATIONAL COMPLEXITY

We quantify computational complexity using inference-stage FLOPs, as this metric reflects the total number of floating-point operations required for forward propagation, independent of hardware or software implementations.

To make the process clearer, we calculate the number of FLOPs for the feature encoding stage (5), message passing stage (6), and the readout stage (7) separately. Let $\mathcal{C}_F$, $\mathcal{C}_{MPNN}$, and $\mathcal{C}_R$ denote the number of FLOPs the feature encoding stage, message passing stage, and the readout stage, respectively. Then the total number of FLOPs will be equivalent to the summation of these three terms, i.e.,

$$\mathcal{C}_{\text{total}} = \mathcal{C}_F + \mathcal{C}_{MPNN} + \mathcal{C}_R. \tag{24}$$

The feature encoder networks F_f, F_l, and F_q are executed for each flow, link, and queue in the network, respectively. As a result, the FLOP count for these feature encoder modules can be calculated as:

$$\mathcal{C}_F = \mathcal{C}(F_f)\mathcal{N}(\mathcal{F}) + \mathcal{C}(F_l)\mathcal{N}(\mathcal{L}) + \mathcal{C}(F_q)\mathcal{N}(\mathcal{Q}), \tag{25}$$

where $\mathcal{C}(\theta)$ denotes the number of FLOPs required for a module θ. The FLOPs of FCN can be calculated as,

$$\mathcal{C}(\theta_{\text{FCN}}) = \sum_{\ell=1}^{L} n[\ell](2n[\ell-1] \ +1) \tag{26}$$

where $n[\ell]$ denotes the dimension of the hidden state of the layer ℓ. The derivation of this equation is given in the Appendix A.

The FLOPs of readout networks can be calculated as:

$$\mathcal{C}_R = \mathcal{C}(R_L)\mathcal{N}(\mathcal{F}) - 2\mathcal{N}(\mathcal{F}) + [6 + \mathcal{C}(R_D) + \mathcal{C}(R_J)] \sum_{\forall f \in \mathcal{F}} \mathcal{N}(P(f)). \tag{27}$$

The readout network for packet loss is executed once per flow and is therefore its FLOPs count scaled by the number of flows, $\mathcal{N}(\mathcal{F})$. In contrast, the delay and jitter readout operations, denoted by R_D and R_J respectively, are applied to each pair $(l, q) \in P(f)$ along the path of every flow f. The delay readout operation involves two division operations and one summation, while the jitter readout operation includes one division per $(l, q) \in P(f)$. Thus, for each $(l, q) \in P(f)$, the total number of FLOPs required for delay and jitter predictions is given by $4 + \mathcal{C}(R_D) + \mathcal{C}(R_J)$. The total computation cost is then obtained by multiplying this value by the sum of all path lengths. Also, it requires $\mathcal{N}(P(f)) - 1$ summations for each flow f when predicting delay or jitter. When adding up this becomes $2\sum_{\forall f \in \mathcal{F}} \mathcal{N}(P(f)) - 2\mathcal{N}(\mathcal{F})$. We add this all up to get the final FLOPs count of $\mathcal{C}_R$.

We can calculate the FLOPs for the MPNN

$$\mathcal{C}_{MPNN} = M\Bigg(\mathcal{C}(U_f) \sum_{\forall f \in \mathcal{F}} \mathcal{N}(P(f)) + \mathcal{C}(U_q)\mathcal{N}(\mathcal{Q}) + n[h_q] \sum_{\forall q \in \mathcal{Q}} \left[\mathcal{N}(Q_f(q)) - 1\right] + \mathcal{C}(U_l) \sum_{\forall l \in \mathcal{L}} \mathcal{N}(L_q(l))\Bigg), \tag{28}$$

where the overall multiplication factor M corresponds to the number of message passing iterations. $n[h_q]$ represents the dimensionality of the hidden states associated with the queues. Recall that U_f, U_l, and U_f are implemented as GRU layers. Their FLOPs count can be calculated as,

$$\mathcal{C}(\theta_{\text{GRU}}) = 6n[h]n[x] + 6n[h]^2 + 10n[h], \tag{29}$$

where $n[h]$ denotes the dimension of the hidden state, $n[x]$ denotes the dimension of the input layer of the GRU. The derivation of this equation is given in Appendix B. The FLOP count for the flow update GRU layer U_f is scaled by the sum of $\mathcal{N}(P(f))$ over all flows f, since it executes $\mathcal{N}(P(f))$ steps for each flow. The GRU layer that updates the queue states is executed once per queue, as in (6b). To compute the summation in (6b), an additional $n[h_q][\mathcal{N}(Q_f(q)) - 1]$ FLOPs are required for each queue q. For link updates, the corresponding GRU layer runs for $\mathcal{N}(L_q(l))$ steps for each link l in the network. Note that the number of FLOPs grows

TABLE 2. FLOPs calculation for MHNet model.

# of FLOPs	Topology		
	NSF	Geant2	GBN
$\mathcal{C}_F$	7.87×10^5	2.12×10^6	1.12×10^6
$\mathcal{C}_R$	1.54×10^6	6.05×10^6	2.79×10^6
$\mathcal{C}_{MPNN}$	6.71×10^7	2.57×10^8	1.21×10^8
$\mathcal{C}_{\text{total}}$	6.94×10^7	2.65×10^8	1.25×10^8

linearly with M. $\mathcal{C}_F$ and $\mathcal{C}_R$ are independent of M as the feature encoding and readout operations are executed once regardless of the number of message passing iterations.

Table 2 presents the FLOP counts $\mathcal{C}_F$, $\mathcal{C}_R$, $\mathcal{C}_{MPNN}$, and $\mathcal{C}_{\text{total}}$ required to predict the performance metrics of a data sample corresponding to three different topologies. These FLOPs counts vary across topologies due to differences in the number of flows, links, and queues. In all cases, the MPNN component accounts for approximately 96% of the total FLOPs. This highlights that we can achieve higher computational efficiency by using a shared backbone, eliminating redundant heavy computations in the MPNN. Fig. 6 compares the FLOPs between the MHNet model and the RouteNet-F [25] model. MHNet achieves a reduction of approximately 67% in FLOPs compared to RouteNet-F by leveraging a common backbone MPNN to learn representations for network elements. Furthermore, we evaluate the inference time of the model on a cloud computing environment equipped with an NVIDIA Tesla V100 GPU. The reported inference time values are obtained by averaging the results over 1000 test samples. Fig. 7 illustrates the inference time comparison; MHNet achieves an inference time of 23–30ms, whereas RouteNet requires 65–90ms. This corresponds to approximately 66% reduction in inference time relative to RouteNet-F. This further highlights the efficiency of our design, as the proposed model requires fewer FLOPs. Since FLOPs directly determine the number of arithmetic computations executed during a forward pass, reducing them decreases the total workload on the hardware, thereby lowering computational complexity and leading to shorter inference time.

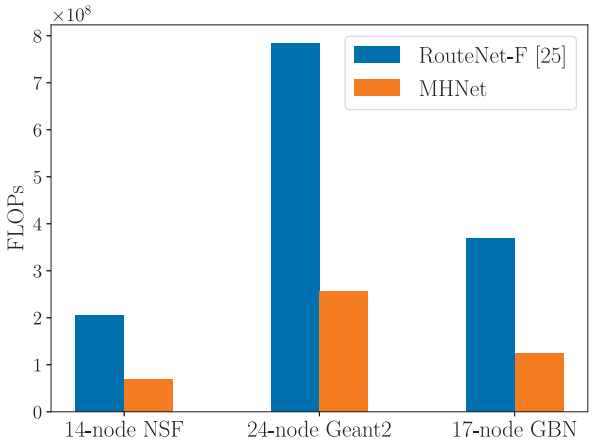


FIGURE 6. Comparison of FLOPs required to predict performance metrics of one data sample. The proposed MHNet model architecture requires approximately 67% less number of FLOPs compared to the state of the art RouteNet [25] model.

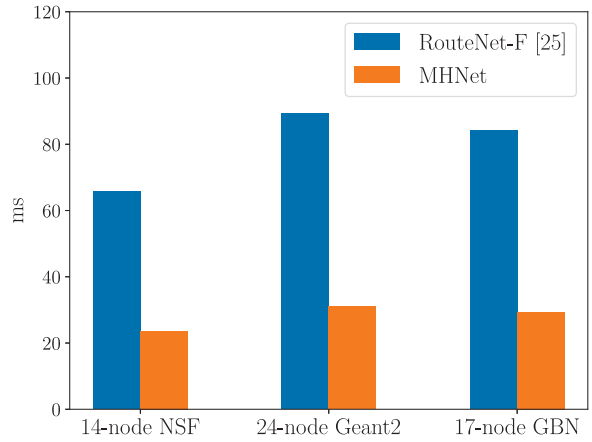


FIGURE 7. Comparison of average inference time to predict performance metrics of one data sample. The proposed MHNet model architecture requires approximately 66% less inference time compared to the state-of-the-art RouteNet [25] model.

D. MEMORY COMPLEXITY

In this section, we analyze the memory requirements of the MHNet model during training. Our analysis focuses on both the model parameters and the intermediate activations, which must be stored to facilitate gradient computation during backpropagation.

We first begin with calculating the number of intermediate activations. For clarity, we calculate the number of intermediate activations for the feature encoding stage (5), message passing stage (6), and the readout stage (7) separately. Let $\mathcal{M}_F$, $\mathcal{M}_{MPNN}$, and $\mathcal{M}_R$ denote the number of intermediate activations for the feature encoding stage, message passing stage, and the readout stage, respectively. Then the total number of intermediate activations will be equivalent to the summation of these three terms, i.e.,

$$\mathcal{M}_{\text{total}} = \mathcal{M}_F + \mathcal{M}_{MPNN} + \mathcal{M}_R. \tag{30}$$

For the feature encoding stage, we can get

$$\begin{aligned}\mathcal{M}_F = \mathcal{M}(F_f)\mathcal{N}(\mathcal{F}) &+ \mathcal{M}(F_l)\mathcal{N}(\mathcal{L}) \\ &+ \mathcal{M}(F_q)\mathcal{N}(\mathcal{Q}),\end{aligned} \tag{31}$$

where $\mathcal{M}(\theta)$ denotes the number of intermediate activations for a module θ. The number of intermediate activations of a FCN can be easily calculated as

$$\mathcal{M}(\theta_{\text{FCN}}) = \sum_{\ell=0}^{L} n[\ell], \tag{32}$$

where we start the indexing of ℓ from 0 to include the input layer. The activation counts are scaled by the number of flows, queues, and links as the feature encoder networks are run for each of them separately on their input features.

TABLE 3. Number of parameters.

GRUs in MPNN		Input Encoding FCNs		FCNs in Readout	
U_f	9.41×10^3	F_f	1.63×10^3	R_D	0.82×10^3
U_l	6.34×10^3	F_l	1.25×10^3	R_J	0.82×10^3
U_q	6.34×10^3	F_q	1.25×10^3	R_L	0.82×10^3
Σ	22.09×10^3	Σ	4.13×10^3	Σ	2.46×10^3

For the readout stage, we can get

$$\mathcal{M}_R = [\mathcal{M}(R_D) + \mathcal{M}(R_J)] \sum_{\forall f \in \mathcal{F}} \mathcal{N}(P(f)) + \mathcal{M}(R_L)\mathcal{N}(\mathcal{F}) \tag{33}$$

The readout network for packet loss R_L is executed once per flow and is therefore its intermediate activation count scaled by the number of flows, $\mathcal{N}(\mathcal{F})$. In contrast, the delay and jitter readout networks, denoted by R_D and R_J respectively, are applied to each pair $(l, q) \in P(f)$ along the path of every flow f. Hence, their intermediate activation counts are scaled with the sum of path lengths, $\mathcal{N}(P(f))$. Note that both $\mathcal{M}_F$ and $\mathcal{M}_R$ are independent of the number of message passing iterations M.

We can calculate the number of intermediate activations required for the message passing stage as

$$\mathcal{M}_{MPNN} = M\Big(\mathcal{M}(U_f) \sum_{\forall f \in \mathcal{F}} \mathcal{N}(P(f)) + \mathcal{M}(U_q)\mathcal{N}(\mathcal{Q}) + \mathcal{M}(U_l) \sum_{\forall l \in \mathcal{L}} \mathcal{N}(L_q(l))\Big), \tag{34}$$

where the overall multiplication factor M came from the number of message passing iterations, as the model needs to keep the copies of intermediate activations of each message passing iteration. Specifically, the term inside the bracket corresponds to the number of intermediate activations of one message passing iteration. $\mathcal{M}(U_f)$, $\mathcal{M}(U_l)$ and $\mathcal{M}(U_q)$ denote the number of intermediate activations required for one timestep of the GRU cells U_f, U_l, and U_q respectively. The intermediate activation count for the flow update GRU layer U_f is scaled by the sum of $\mathcal{N}(P(f))$ over all flows f, since it executes $\mathcal{N}(P(f))$ steps for each flow f. Similarly, the intermediate activation count of U_l is scaled by summation of $\mathcal{N}(L_q(l))$ over all links l. The intermediate activation count of U_q is scaled by $\mathcal{N}(\mathcal{Q})$ as it executes once for each queue after taking the summation of GRU time step outputs as in (6b). The number of intermediate activations required for one time step of a GRU cell with a hidden state dimension of $n[h]$ can be calculated as,

$$\mathcal{M}(\theta_{\text{GRU}}) = 4n[h] \tag{35}$$

where the factor of 4 comes as we have 4 intermediate outputs in a GRU cell as described in (38).

The number of intermediate activations in our experiments is summarized in Table 4. The parameter counts for each of the NN modules are summarized in Table 3. The equations for calculating the parameter counts for each NN module are given in the Appendix C. It is important to note that the number of model parameters depends solely on the dimensions of the model layers and does not scale with the network's size or the number of message-passing iterations. These results indicate that the number of intermediate activations is significantly higher than the model's parameter count, even for relatively small topologies. The amount of intermediate activations naturally increases with the number of links, queues, flows, message-passing iterations, and the summation of path lengths; yet, the proposed MHNet design requires considerably less amount of intermediate activations due to the use of a shared backbone MPNN. In particular, the MPNN accounts for nearly 86% of the intermediate output activations, which highlights the effectiveness of our approach in minimizing memory usage during the training stage. Our empirical results using the 32-bit floating point precision indicate that one iteration of MPNN requires approximately 2.1 MB for the 14-node NSF topology, 8 MB for the 24-node Geant2 topology, and 3.8 MB for the 17-node GBN topology. In our experiments with $M = 8$, the estimated memory requirements for the MPNN are approximately 17 MB for the NSF topology, 64 MB for the Geant2 topology, and 30 MB for the GBN topology during the gradient backpropagation process, accounting for over 75% of the total memory requirement, including the TensorFlow overheads. Fig. 8 compares the

TABLE 4. Number of intermediate activations.

# of intermediate activations	Topology		
	NSF	Geant2	GBN
$\mathcal{M}_F$	2.05×10^4	5.49×10^4	2.92×10^4
$\mathcal{M}_R$	6.25×10^4	2.45×10^5	1.13×10^5
$\mathcal{M}_{MPNN}$	4.85×10^5	1.80×10^6	8.58×10^5
$\mathcal{M}_{\text{total}}$	5.68×10^5	2.10×10^6	1.00×10^6

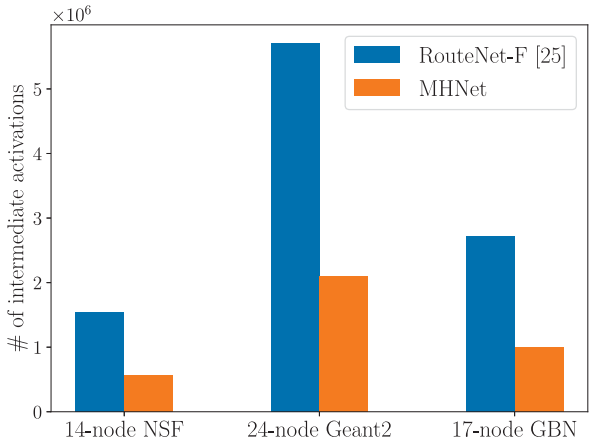


FIGURE 8. Comparison of the number of intermediate activations when predicting performance metrics of one data sample. The proposed MHNet model architecture requires approximately 64% less intermediate activations compared to the state-of-the-art RouteNet [25] model.

TABLE 5. Interpreting learned relationships: Metrics calculated from (18), (19), (20) and (21).

Traffic distribution	Delay				Jitter				Packet loss			
	IoU_f	IoU_l	Γ_f	Γ_l	IoU_f	IoU_l	Γ_f	Γ_l	IoU_f	IoU_l	Γ_f	Γ_l
Constant bitrate	0.84	0.77	0.91	0.99	0.80	0.75	0.88	0.99	0.79	0.77	0.92	0.99
On off	0.80	0.78	0.90	0.99	0.77	0.77	0.87	0.99	0.76	0.77	0.89	0.99
Autocorrelated exponentials	0.83	0.80	0.90	0.99	0.82	0.79	0.89	0.99	0.82	0.79	0.94	0.99
Modulated exponentials	0.81	0.81	0.90	0.99	0.78	0.81	0.90	0.99	0.79	0.81	0.92	0.99
All multiplexed	0.80	0.79	0.90	0.99	0.78	0.78	0.89	0.99	0.73	0.77	0.88	0.99

intermediate activations between MHNet and RouteNet-F [25], showing that MHNet requires approximately 64% fewer activations. This reduction is significant, as it directly improves scalability to larger topologies where memory growth becomes a critical bottleneck.

E. INTERPRETING LEARNED RELATIONSHIPS

We analyze and interpret the relationships that the MHNet model has learned using the methodology constructed in Section V. We randomly selected 10 samples from the test dataset that uses the GBN topology. For each prediction output, we calculate gradients with respect to link and path input features using the built-in tf.GradientTape API of the TensorFlow framework. We calculate the metrics defined in (18), (19), (20), and (21) for each flow prediction and take the average over all flows.

The resulting metric values are summarized in Table 5. All IoU_ℓ scores fall within the range of 0.75 to 0.81, indicating that the proposed MHNet model predominantly focuses on links traversed by the target flow and its neighboring flows when estimating performance metrics. Similarly, the IoU_f values range from 0.73 to 0.84, suggesting that the model's predictions for a given flow are largely influenced by its input features as well as those of neighboring flows. The Γ_f values range from 0.87 to 0.94, demonstrating that the gradient norms corresponding to the features of a flow f and its neighboring flows are significantly higher than those of unrelated flows. Meanwhile, the Γ_ℓ values consistently lie around 0.99, showing that the gradient magnitudes of links traversed by a flow and their neighboring flows dominate over those of other links.

It is worth mentioning that we have 8 message-passing iterations in our GNN model. The metrics in Table 5 show that the MHNet model avoids the oversmoothing effect typically encountered in GNNs with multiple message passing iterations. The high Γ_ℓ values (around 0.99) and Γ_f values (0.87–0.94) show that the gradient norms are significantly larger for features of the target flow f, its neighboring flows, and the links traversed by them, compared to other links and flows in the network graph. This suggests that the model maintains a strong focus on relevant substructures when predicting performance metrics of traffic flows. Moreover, the IoU_f and IoU_ℓ scores further confirm that the predictions are primarily influenced by the input features of the flow, its neighboring flows, and the links they traverse, rather than being diluted by distant or irrelevant features. Together, these results provide strong evidence that MHNet learns discriminative representations and effectively limits the propagation of noisy or non-informative features, thereby mitigating the oversmoothing phenomenon.

VII. CONCLUSION

We introduced MHNet, a GNN architecture developed to predict delay, jitter, and packet loss in computer networks using a single model. MHNet employs a shared MPNN network to extract representations from flows, queues, and links, and applies separate readout branches to produce the three performance metrics. To enable effective multi-task training, we proposed a gradient-based optimization method that balances the influence of the individual loss functions. Furthermore, we developed an interpretation framework based on gradient analysis to provide insights into the operation of the model. This framework quantifies the contribution of input features to the predictions and highlights the network substructures that most strongly influence the outcomes. Experimental results demonstrate that MHNet achieves accuracy comparable to or better than existing methods, while requiring 67% fewer FLOPs during inference. The interpretation analysis further shows that the model consistently attends to the most relevant substructures of the network feature graph even after multiple rounds of message passing. Although the simulated dataset used in this study has been carefully constructed to approximate the statistical characteristics of real Internet traffic and is widely recognized as a benchmark in network modeling, an important direction for future work is to validate MHNet on real-world traffic traces.

APPENDIX A

NUMBER OF FLOATING POINT OPERATIONS OF FCNs

A FCN is composed of multiple sequential layers, each applying a linear transformation followed by a non-linear activation function. The forward propagation through layer ℓ can be expressed as:

$$\boldsymbol{y}[\ell] = \psi(\boldsymbol{W}[\ell]\boldsymbol{y}[\ell-1] \ + \boldsymbol{b}[\ell]), \tag{36}$$

where $\boldsymbol{y}[\ell-1]$ represents the input to layer ℓ (typically the output of layer $\ell-1$ unless it is the network input), $\boldsymbol{W}[\ell]$ is the weight matrix, $\boldsymbol{b}[\ell]$ is the bias, and $\boldsymbol{y}[\ell]$ denotes the output of layer ℓ (usually passed to the next layer unless it is the output layer). ψ is an activation function to introduce

non-linear behavior. The number of FLOPs required for the forward pass of one input can be calculated as,

$$\mathcal{C}(\theta_{\text{FCN}}) = \sum_{\ell=1}^{L} n[\ell](2n[\ell-1] \ + 1) \tag{37}$$

where $n[\ell]$ denotes the dimension of the hidden state of the layer ℓ and L is the number of layers. The term $2n[\ell-1]$ coming from taking the dot product between one row of $\boldsymbol{W}[\ell]$ with $\boldsymbol{y}[\ell-1]$. Specifically, it requires $n[\ell-1]$ multiplications and $n[\ell-1]$ additions. This is multiplied by $n[\ell]$ because we need to obtain $n[\ell]$ dot products to complete the matrix multiplication. The final $n[\ell]$ term is coming from the addition of the bias vector.

APPENDIX B
NUMBER OF FLOATING POINT OPERATIONS OF GRUs

A GRU processes sequential input data using gating mechanisms to control the flow of information across time steps. At each time step t, a GRU cell computes the following operations [53]:

$$\boldsymbol{z}_t = \sigma(\boldsymbol{W}_z\boldsymbol{x}_t + \boldsymbol{U}_z\boldsymbol{h}_{t-1} + \boldsymbol{b}_z), \tag{38a}$$

$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}_r\boldsymbol{x}_t + \boldsymbol{U}_r\boldsymbol{h}_{t-1} + \boldsymbol{b}_r), \tag{38b}$$

$$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{W}_h\boldsymbol{x}_t + \boldsymbol{U}_h(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}) + \boldsymbol{b}_h), \tag{38c}$$

$$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}}_t, \tag{38d}$$

where $\boldsymbol{x}_t \in \mathbb{R}^{n[x]}$ is the input at time step t, $\boldsymbol{h}_{t-1} \in \mathbb{R}^{n[h]}$ is the previous hidden state, and $\boldsymbol{z}_t \in \mathbb{R}^{n[h]}$, $\boldsymbol{r}_t \in \mathbb{R}^{n[h]}$, $\tilde{\boldsymbol{h}}_t \in \mathbb{R}^{n[h]}$, and $\boldsymbol{h}_t$ are the update gate, reset gate, candidate hidden state, and final hidden state respectively. The matrices $\boldsymbol{W}_*$ and $\boldsymbol{U}_*$ are weight matrices for input and recurrent connections, and $\boldsymbol{b}_*$ are bias vectors.

The number of FLOPs required for the forward pass of a single GRU cell at one time step can be calculated by considering the cost of matrix multiplications, element-wise operations, and activation functions. Each gate requires:

- $2n[x]n[h]$ FLOPs for matrix multiplications $\boldsymbol{W}_z\boldsymbol{x}_t$, $\boldsymbol{W}_r\boldsymbol{x}_t$ and $\boldsymbol{W}_h\boldsymbol{x}_t$.
- $2n[h]^2$ FLOPs for matrix multiplications $\boldsymbol{U}_z\boldsymbol{h}_{t-1}$, $\boldsymbol{U}_r\boldsymbol{h}_{t-1}$ and $\boldsymbol{U}_h(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1})$.
- $2n[h]$ FLOPs for addition operations.

There are three such gates (*update*, *reset*, *forget*), plus an additional $4n[h]$ FLOPs for the final update step (38d) and element-wise multiplication of $\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}$ in (38c).

Thus, the total number of floating point operations per time step of the GRU forward pass is:

$$\mathcal{C}(\theta_{\text{GRU}}) = 6n[h]n[x] + 6n[h]^2 + 10n[h]. \tag{39}$$

APPENDIX C
NUMBER OF WEIGHTS PARAMETERS OF NN MODULES

Let $\eta(\theta)$ denote the number of parameters required for a NN layer or model θ. A fully connected layer ℓ with n_{in} input neurons and n_{out} output neurons contains

$$\eta(\ell) = (n_{\text{in}} \times n_{\text{out}}) + n_{\text{out}} \tag{40}$$

trainable parameters, where the first term corresponds to the weight matrix and the second term accounts for the bias vector. For a feed-forward network with L layers, where layer ℓ has $n[\ell-1]$ inputs and $n[\ell]$ outputs, the total parameter count is given by

$$\eta(\theta_{\text{FCN}}) = \sum_{\ell=1}^{L} (n[\ell-1] \times n[\ell] + n[\ell]), \tag{41}$$

with $n[0]$ denoting the input dimension of the model and $n[L]$ the output dimension.

A GRU cell with input dimension $n[x]$ and hidden state dimension $n[h]$ has

$$\eta(\theta_{\text{GRU}}) = 3 \times \Big(n[x] \times n[h] + n[h]^2 + n[h]\Big), \tag{42}$$

parameters. The three terms inside the brackets correspond to the weights and biases corresponding to each of the three gates (reset, update, and forget).

REFERENCES

[1] L. Kleinrock, "Analytic and simulation methods in computer network design," in *Proc. Joint Comput. Conf.*, New York, NY, USA, 1970, pp. 569–579, doi: 10.1145/1476936.1477022.

[2] J. Zerwas, I. Poese, S. Schmid, and A. Blenk, "On the benefits of joint optimization of reconfigurable CDN-ISP infrastructure," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 1, pp. 158–173, Mar. 2022, doi: 10.1109/TNSM.2021.3119134.

[3] A. Sacco, M. Flocco, F. Esposito, and G. Marchetto, "Partially oblivious congestion control for the Internet via reinforcement learning," *IEEE Trans. Netw. Service Manag.*, vol. 20, no. 2, pp. 1644–1659, Jun. 2023, doi: 10.1109/TNSM.2022.3215669.

[4] A. Mestres et al., "Knowledge-defined networking," *SIGCOMM Comput. Commun. Rev.*, vol. 47, no. 3, pp. 2–10, 2017, doi: 10.1145/3138808.3138810.

[5] H. Wang, Y. Wu, G. Min, and W. Miao, "A graph neural network-based digital twin for network slicing management," *IEEE Trans. Ind. Informat.*, vol. 18, no. 2, pp. 1367–1376, Feb. 2022, doi: 10.1109/TII.2020.3047843.

[6] T. G. Robertazzi, *Computer Networks and Systems: Queueing Theory and Performance Evaluation*. Heidelberg, Germany: Springer-Verlag, 2000.

[7] R. M. Fujimoto, K. Perumalla, A. Park, H. Wu, M. H. Ammar, and G. F. Riley, "Large-scale network simulation: How big? How fast?" in *Proc. 11th IEEE/ACM Int. Symp. Model. Anal. Simulation Comput. Telecommun. Syst.*, 2003, pp. 116–123, doi: 10.1109/MASCOT.2003.1240649.

[8] Z. Xu et al., "Experience-driven networking: A deep reinforcement learning based approach," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 1871–1879, doi: 10.1109/INFOCOM.2018.8485853.

[9] T. Karagiannis, M. Molle, M. Faloutsos, and A. Broido, "A nonstationary Poisson view of Internet traffic," in *Proc. IEEE INFOCOM*, vol. 3, 2004, pp. 1558–1569, doi: 10.1109/INFCOM.2004.1354569.

[10] T. Karagiannis, M. Molle, and M. Faloutsos, "Long-range dependence ten years of Internet traffic modeling," *IEEE Internet Comput.*, vol. 8, no. 5, pp. 57–64, Sep./Oct. 2004, doi: 10.1109/MIC.2004.46.

[11] E. Kresch and S. Kulkarni, "A Poisson based Bursty model of Internet traffic," in *Proc. IEEE 11th Int. Conf. Comput. Inf. Technol.*, 2011, pp. 255–260, doi: 10.1109/CIT.2011.95.

[12] V. Paxson and S. Floyd, "Wide area traffic: The failure of Poisson modeling," *IEEE/ACM Trans. Netw.*, vol. 3, no. 3, pp. 226–244, Jun. 1995, doi: 10.1109/90.392383.

[13] K. Xie, X. Li, X. Wang, G. Xie, J. Wen, and D. Zhang, "Graph based tensor recovery for accurate Internet anomaly detection," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 1502–1510, doi: 10.1109/INFOCOM.2018.8486332.

[14] D. Andreoletti, S. Troia, F. Musumeci, S. Giordano, G. Maier, and M. Tornatore, "Network Traffic prediction based on diffusion convolutional recurrent neural networks," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 2019, pp. 246–251, doi: 10.1109/INFCOMW.2019.8845132.

[15] L. Deng, X.-Y. Liu, H. Zheng, X. Feng, and Z. Chen, "Graph-tensor neural networks for network traffic data imputation," *IEEE/ACM Trans. Netw.*, vol. 31, no. 6, pp. 3010–3024, Dec. 2023, doi: 10.1109/TNET.2023.3268982.

[16] A. Mestres, E. Alarcón, Y. Ji, and A. Cabellos-Aparicio, "Understanding the modeling of computer network delays using neural networks," in *Proc. Workshop Big Data Anal. Mach. Learn. Data Commun. Netw.*, 2018, pp. 46–52.

[17] A. G. Parlos, "Identification of the Internet end-to-end delay dynamics using multi-step neuro-predictors," in *Proc. Int. Joint Conf. Neural Netw.*, vol. 3, 2002, pp. 2460–2465, doi: 10.1109/IJCNN.2002.1007528.

[18] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, Jan. 2009, doi: 10.1109/TNN.2008.2005605.

[19] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2021, doi: 10.1109/TNNLS.2020.2978386.

[20] K. Rusek, J. Suárez-Varela, P. Almasan, P. Barlet-Ros, and A. Cabellos-Aparicio, "RouteNet: Leveraging graph neural networks for network modeling and optimization in SDN," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 10, pp. 2260–2270, Oct. 2020, doi: 10.1109/JSAC.2020.3000405.

[21] W. Zheng et al., "End-to-end delay modeling via leveraging competitive interaction among network flows," *IEEE Trans. Netw. Service Manag.*, vol. 21, no. 2, pp. 1634–1647, Apr. 2024, doi: 10.1109/TNSM.2023.3340848.

[22] S. Messaoudi, A. Ksentini, F. Messaoudi, and C. Bonnet, "GNN-based SDN admission control in beyond 5G networks," in *Proc. IEEE Global Commun. Conf.*, 2023, pp. 6103–6108, doi: 10.1109/GLOBECOM54140.2023.10437301.

[23] Q. He et al., "Routing optimization with deep reinforcement learning in knowledge defined networking," *IEEE Trans. Mobile Comput.*, vol. 23, no. 2, pp. 1444–1455, Feb. 2024, doi: 10.1109/TMC.2023.3235446.

[24] C. Liu et al., "FERN: Leveraging graph attention networks for failure evaluation and robust network design," *IEEE/ACM Trans. Netw.*, vol. 32, no. 2, pp. 1003–1018, Apr. 2024, doi: 10.1109/TNET.2023.3311678.

[25] M. Ferriol-Galmés et al., "RouteNet-Fermi: Network modeling with graph neural networks," *IEEE/ACM Trans. Netw.*, vol. 31, no. 6, pp. 3080–3095, Dec. 2023, doi: 10.1109/TNET.2023.3269983.

[26] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green AI," *Commun. ACM*, vol. 63, no. 12, pp. 54–63, 2020, doi: 10.1145/3381831.

[27] M. Ferriol-Galmés et al., "Network modelling dataset: Version 6," 2025. [Online]. Available: https://github.com/BNN-UPC/NetworkModelingDatasets/tree/master/datasets_v6

[28] M. Ferriol-Galmés et al., "BNNetSimulator," 2025. [Online]. Available: https://github.com/BNNUPC/BNNetSimulator

[29] A. S. Langari, L. Yeganeh, and K. K. Nguyen, "Grothendieck graph neural network (GGNN): A path-based framework for network modelling," in *Proc. IEEE Global Commun. Conf.*, 2023, pp. 6548–6553, doi: 10.1109/GLOBECOM54140.2023.10437032.

[30] J. Liu et al., "Practical network modeling using weak supervision signals for human-centric networking in metaverse," *IEEE J. Sel. Areas Commun.*, vol. 42, no. 3, pp. 680–693, Mar. 2024, doi: 10.1109/JSAC.2023.3345391.

[31] M. Ferriol-Galmés et al., "RouteNet-Erlang: A graph neural network for network performance evaluation," in *Proc. Conf. Comput. Commun.*, 2022, pp. 2018–2027, doi: 10.1109/INFOCOM48880.2022.9796944.

[32] A. Varga, "Discrete event simulation system," in *Proc. Eur. Simulat. Multiconf. (ESM)*, vol. 17, 2001, pp. 1–11.

[33] X. Hei, J. Zhang, B. Bensaou, and C.-C. Cheung, "Wavelength converter placement in least-load-routing-based optical networks usinggenetic algorithms," *J. Opt. Netw.*, vol. 3, no. 5, pp. 363–378, 2004, doi: 10.1364/JON.3.000363.

[34] F. Barreto, "Fast emergency paths schema to overcome transient link failures in OSPF routing," *Int. J. Comput. Netw. Commun.*, vol. 4, no. 2, pp. 17–34, 2012, doi: 10.5121/ijcnc.2012.4202.

[35] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski, "SNDlib 1.0—Survivable network design library," *Networks*, vol. 55, no. 3, pp. 276–286, 2010.

[36] J. Pedro, J. Santos, and J. Pires, "Performance evaluation of integrated OTN/DWDM networks with single-stage multiplexing of optical channel data units," in *Proc. 13th Int. Conf. Transparent Optical Netw.*, 2011, pp. 1–4, doi: 10.1109/ICTON.2011.5970940.

[37] K. Hashimoto, C. Xiong, Y. Tsuruoka, and R. Socher, "A joint many-task model: Growing a neural network for multiple NLP tasks," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, Copenhagen, Denmark, 2017, pp. 1923–1933, doi: 10.18653/v1/D17-1206.

[38] H. D. Fernando, H. Shen, M. Liu, S. Chaudhury, K. Murugesan, and T. Chen, "Mitigating gradient bias in multi-objective learning: A provably convergent approach," in *Proc. 11th Int. Conf. Learn. Represent.*, 2023, pp. 1–14.

[39] D. Chanda and N. Y. Soltani, "A heterogeneous graph-based multi-task learning for fault event diagnosis in smart grid," *IEEE Trans. Power Syst.*, vol. 40, no. 2, pp. 1427–1438, Mar. 2025, doi: 10.1109/TPWRS.2024.3447533.

[40] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn, "Gradient surgery for multi-task learning," in *Proc. 34th Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 1–21.

[41] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 6th ed. London, U.K.: Pearson, 2012.

[42] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[43] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Represent.*, 2015, pp. 1–11.

[44] J. Pizarroso, J. Portela, and A. Muñoz, "NeuralSens: Sensitivity analysis of neural networks," *J. Stat. Softw.*, vol. 102, no. 7, pp. 1–36, 2022, doi: 10.18637/jss.v102.i07.

[45] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," in *Proc. Workshop Int. Conf. Learning Represent.*, 2014, pp. 1–9.

[46] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the self-similar nature of Ethernet traffic (extended version)," *IEEE/ACM Trans. Netw.*, vol. 2, no. 1, pp. 1–15, Feb. 1994, doi: 10.1109/90.282603.

[47] S. Huang et al., "xNet: Modeling network performance with graph neural networks," *IEEE/ACM Trans. Netw.*, vol. 32, no. 2, pp. 1753–1767, Apr. 2024, doi: 10.1109/TNET.2023.3329357.

[48] S. Belhaj and M. Tagina, "Modeling and prediction of the Internet end-to-end delay using recurrent neural networks," *J. Netw.*, vol. 4, no. 6, pp. 528–535, 2009.

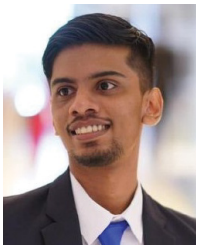
[49] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proc. 6th Int. Conf. Learn. Represent.*, 2017, pp. 1–21.

[50] N. Draper and H. Smith, *Applied Regression Analysis*(Wiley Series in Probability and Statistics). Hoboken, NJ, USA: Wiley, 1998.

[51] L. Engstrom, B. Tran, D. Tsipras, L. Schmidt, and A. Madry, "Exploring the landscape of spatial robustness," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 1802–1811.

[52] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Trans. Evol. Comput.*, vol. 23, no. 5, pp. 828–841, Oct. 2019, doi: 10.1109/TEVC.2019.2890858.

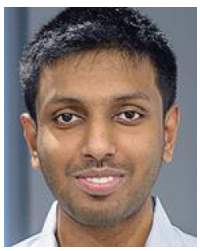
[53] M. Ławryńczuk and K. Zarzycki, "LSTM and GRU type recurrent neural networks in model predictive control: A Review," *Neurocomputing*, vol. 632, Jun. 2025, Art. no. 129712, doi: 10.1016/j.neucom.2025.129712.



SANDUSHAN RANAWEERA (Student Member, IEEE) received the B.Sc. engineering Honours (first class) degree in electronic and telecommunication engineering from the University of Moratuwa, Sri Lanka, in 2022. He is currently pursuing the Ph.D. degree with the School of Electrical and Data Engineering, University of Technology Sydney, Australia. His research interests are machine learning for network modeling, and physical layer algorithms with machine learning.

REN PING LIU (Senior Member, IEEE) received the B.E. degree in telecommunication engineering and the M.E. degree in computer engineering from the Beijing University of Posts and Telecommunications, Beijing, China, in 1985 and 1988, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Newcastle, Callaghan, NSW, Australia, in 1996. He is a Professor and the Head of Discipline of Network and Cybersecurity, University of Technology Sydney, Ultimo, NSW. As a Research Leader, a Certified Network Professional, and a Full Stack Web Developer, he has delivered networking and cybersecurity solutions to government agencies and industry customers. He has supervised over 30 Ph.D. students and has over 200 research publications. His research interests include wireless networking, 5G, the IoT, vehicular networks, 6G, cybersecurity, and blockchain. He was the Winner of NSW iAwards 2020 for leading the Blockchain Enabled Fish Provenance and Quality Tracking Project. He received the Australian Engineering Innovation Award 2012 and the CSIRO Chairman's Medal for his contribution in the Wireless Backhaul Project. He was the Founding Chair of IEEE NSW VTS Chapter.

YING HE (Senior Member, IEEE) received the B.Eng. degree in telecommunications engineering from the Beijing University of Posts and Telecommunications, China, in 2009, and the Ph.D. degree in telecommunications engineering from the University of Technology Sydney, Australia, in 2017, where she is currently a Senior Lecturer with the School of Electrical and Data Engineering. Her research interests are physical layer algorithms in wireless communication with machine learning, vehicular communication, spectrum sharing, and satellite communication.



BEESHANGA JAYAWICKRAMA (Senior Member, IEEE) received the B.E. degree (Hons.) in telecommunications engineering and the Ph.D. degree in electronic engineering from Macquarie University, Australia, in 2011 and 2015, respectively. He is currently a Visiting Fellow with the University of Technology Sydney, Australia. He was extensively involved in spectrum sensing and interference mitigation research for spectrum access systems. His research interests include nonterrestrial networks, 5G/6G, cognitive radio, and signal processing.