DeepQueueNet reproducibility: A Two-Phase Experimental Approach

Claudio Saponaro
ETSETB
UPC, Barcelona, Spain
c.saponaro@upc.edu

***Abstract*—The present report details a thorough reproducibility study of DeepQueueNet, a deep learning framework for predicting network queue delays at the packet level. The present study employs a two-phase experimental methodology. The first phase involves the execution of controlled experiments on synthetic M/D/1 queue data, with the objective of validating the proposed approach. The second phase comprises large-scale experiments on realistic network simulation data. The present study addresses the key challenges in scaling deep learning models to large network datasets while maintaining temporal dependencies. The experimental results, utilizing a toy dataset, yielded an R^2 score of 0.58 and an MSE of 0.098, thereby substantiating the model's capacity to capture queue dynamics. The full-scale experiment demonstrated the capacity to capture queue dynamics with an MSE of 0.0685. However, the low R^2 value of 0.28 suggests that the model may be underfit. Notwithstanding the computational and temporal constraints, insights are to be provided into the practical challenges of deploying such models in real-world scenarios. This work contributes to the understanding of deep learning applications in network performance prediction and highlights critical considerations for reproducibility in this domain.**

I. Introduction

Network performance prediction remains a fundamental challenge in modern communication systems, directly impacting quality of service (QoS) and network optimization strategies. Conventional methodologies depend on analytical models or discrete event simulation (DES), which, while precise, encounter scalability constraints as network intricacy rises. Recent advances in deep learning have opened new avenues for network performance estimation. DeepQueueNet [1] represents a significant contribution in this domain, proposing a hybrid approach that combines queuing theory with neural networks to predict packet-level delays. In contrast to approaches that are entirely data-driven, such as MimicNet, as discussed in the literature, [4], DeepQueueNet utilizes domain knowledge to prioritize learning on complex, analytically intractable components. The present paper conducts a thorough reproducibility study of the DeepQueueNet architecture, with the aim of addressing several critical research questions. Firstly, it is necessary to ascertain whether the proposed architecture is capable of effectively learning temporal patterns in queue behaviour. Secondly, it is important to consider how the model is scaled to realistic network datasets. Thirdly, it is imperative to ascertain the practical challenges encountered in the reproduction of deep learning-based network models. Our contributions are threefold. Firstly, a rigorous two-phase experimental methodology is provided, enabling controlled validation of deep learning approaches for network modeling. Secondly, the efficacy of LSTM-based architectures in capturing queue dynamics is demonstrated through extensive experimentation on both synthetic and realistic datasets. In conclusion, a critical analysis of the reproducibility challenges in this domain is presented, including computational constraints and the complexities of data preprocessing.

II. Related Work

A. Network Performance Modeling

Conventional network performance modeling has been predominantly reliant on analytical approaches rooted in queuing theory [2].While these methods provide a theoretical foundation, they frequently fail to capture the complexity of contemporary network environments, which are characterized by dynamic traffic patterns and heterogeneous topologies.

Recent research has explored machine learning approaches for network modeling. Graph Neural Networks (GNNs) have demonstrated efficacy in capturing network topology effects [3], while time series forecasting methods have been applied to traffic prediction [6]. However, most approaches focus on aggregate-level predictions rather than packet-level visibility.

B. Deep Learning for Queue Delay Prediction

The application of deep learning to queue delay prediction has gained significant attention. RouteNet [5] demonstrated the potential of GNNs for network-wide performance prediction. MimicNet [4] proposed an end-to-end learning approach but lacked interpretability and required extensive training data.

DeepQueueNet [1] addresses these limitations by integrating analytical queuing models with neural networks. This hybrid approach enables more efficient learning while maintaining packet-level granularity. Our study extends this work by providing comprehensive reproducibility analysis and identifying practical deployment challenges.

III. First Phase - Toy Dataset Experiment

This section delineates the preliminary experimental phase that was conducted using a simplified toy dataset. The objective of this phase was to validate the overall approach and test specific methods that would subsequently be reused in the full-scale experiment.

A. Dataset Definition

This phase may be characterized as theoretical in nature. Prior to the implementation of a solution, it is imperative to comprehend the nature of the dataset required to address the problem. The toy dataset is a synthetic dataset representing a queuing theory model, specifically an M/D/1 queue. An M/D/1 queue is a classic queuing model in which arrivals follow a Poisson process, service times are constant, and there is one server. This model is designed to capture systems characterized by random packet arrivals but fixed processing times. Consequently, it is particularly useful for the analysis of delay and queue behaviour in network routers or switches under conditions of predictable service. Following the presentation of the theoretical background and underlying assumptions, the subsequent step will be the presentation of the toy dataset, which comprises 5,000 simulations. Each simulation involves a variable number of packets; on average, 900 packets have to be processed per simulation.

B. Dataset Creation

The initial step in the process was to create a CSV (Comma-Separated Values) dataset from the initial simulations. This objective was accomplished by employing a bespoke function to map each simulation packet into a designated row of the CSV file. The resulting CSV file contains the following structure:

- simulation id : placeholder of the simulation
- arrival rate : represents the rate at which the packet arrive in the network
- arrival time : represents the specific timestamp when each individual packet arrives in the system
- delay : represents the total delay
- queue delay : representing the target feature of the dataset

Initially, the service time was included, but as it remains constant across all simulations, it does not contribute to the predictive power of the model and can therefore be disregarded.

C. Preprocessing

The second step of the experiment involved the preprocessing of the dataset prior to its introduction into the model. The primary challenge encountered during this phase pertained to the process of sequence creation. The model undergoing training was a Long Short-Term Memory (LSTM) [7] architecture, which requires sequential input data. LSTM [7] is a type of recurrent neural network designed to learn patterns in sequential data by retaining relevant information over long time steps. In the context of predicting queuing delay, it is suitable because it can capture temporal dependencies in the queue's behavior, such as how past packet arrivals and service events influence future delays. The initial step in the process entailed feature augmentation, which entailed the incorporation of the inter-packet gap into the DataFrame as a relevant feature. The inter-packet gap is defined as the time interval between successive packet arrivals. When packets arrive in close succession, characterized by a minimal inter-packet gap, this signifies that the router is receiving data at an elevated rate. This, in turn, leads to an increased probability of congestion, consequently resulting in delay. Conversely, in scenarios where the inter-packet gaps are more substantial, the router is allotted a greater amount of processing time, and delays are prone to being negligible. This feature has the potential to enhance the predictive capability of the final machine learning (ML) model. Subsequently, a Min-Max scaler was applied using the Sklearn library, but only to the training indices to avoid data leakage. The Min-Max Scaler is a technique for normalization that transforms features to a fixed range, typically [0,1], by rescaling based on the minimum and maximum values of the feature. This is a common technique used to ensure faster convergence during prediction.

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{1}$$

The next crucial task was generating input sequences while preserving temporal order within each simulation. It was important to ensure that the end of simulation *n* was not mistakenly merged with the beginning of simulation *n+1*. To enforce this separation, the simulation identifier was used to delimit sequences between different simulations. This was accomplished using a sliding window approach combined with padding to manage sequences containing fewer data points. The artificial padding elements were defined using an arrival time set to zero and an inter-packet gap set to 10, two orders of magnitude higher than the average value found in the CSV, to distinguish them from real simulation data. Following the generation of the sequences, the dataset was divided according to the training indices that had been previously determined, following an 80-10-10 ratio: 80% for training, 10% for validation, and 10% for testing. Finally, the dataset was subjected to a process of shuffling prior to the training phase. This procedure was undertaken with a view to mitigating the risk of overfitting and facilitating the model's acquisition of the true underlying relationships inherent within the data.

D. Architecture

The final step in the process was to construct the model architecture. In order to address this issue, the decision was taken to implement an LSTM model using TensorFlow and Keras. The architecture of the model consists of a two-layer LSTM stack, with the first layer comprising 64 units and the second layer containing 32 units. The initial layer is configured to capture detailed sequential patterns, while the subsequent layer is designed to focus on the most significant long-term dependencies. L2 regularization was applied to both LSTM layers in order to prevent overfitting by penalizing large weights. This is an essential consideration for time series data, where models can easily memorize patterns and overfit. The primary challenge encountered pertained to the loss function. In the initial version of the implementation, it was discovered that common loss functions such as MSE (Mean Square Error) did not perform well, particularly in

predicting spikes in the delay, which are not well captured by the original architecture. In order to address this challenge, the Huber loss was selected, a loss function that has been shown to be more robust to outliers and more effective in capturing queuing delay spikes. The rationale behind this phenomenon pertains to the behaviour of Huber loss, which functions as a combination of both mean squared error (MSE) and mean absolute error (MAE). Specifically, Huber loss acts as MSE for small errors, ensuring smooth gradients, while also operating as MAE for large errors, thereby reducing sensitivity to outliers. Consequently, Huber loss provides a balanced trade-off in capturing both normal delay and rare spikes. This can be proved looking at the following formulation:

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{if } |a| \leq \delta \\ \delta\left(|a| - \frac{1}{2}\delta\right) & \text{otherwise} \end{cases} \quad (2)$$

where $a = y - \hat{y}$ and δ is a threshold parameter.
In Table III we can take a look of the hyperparameter which characterize the architecture.

TABLE I
MODEL HYPERPARAMETERS

Parameter	**Value**
LSTM Layer 1	64 units
LSTM Layer 2	32 units
Learning Rate	0.001
Dense Layer	16 units
Output Layer	1 unit
Epochs	100
Delta (Huber Loss)	1
L2 regularization	0.001
batch size	32

E. Results

The final part of the first experiment focuses on the collection and evaluation of results. The model was evaluated on a 10% of the data, designated as the test set. As illustrated in Table 2, the best combination of hyperparameters was determined through a process of experimentation. The results of this process are summarized in TableII which presents the performance metrics achieved on the toy dataset.

TABLE II
PERFORMANCE RESULTS ON TOY DATASET

Two-layers LSTM model	MSE	R^2	MAE	MAPE
Optimized Configuration	0.0985	0.5838	0.1470	47.66

Here a brief illustration of the metrics:

- MSE : Measures the average of the squared differences between predicted and actual values, penalizing the errors more
- R^2 (Coefficient of Determination): Measures the proportion of variance in the target explained by the model, in other terms how the predictions are correlated with the true values.
- MAE (Mean Absolute Error) : Measures the average of absolute differences between predicted and actual values.
- MAPE (Mean Absolute Percentage Error) : Measures the average of absolute percentage errors, useful for relative accuracy. In particular to avoid the division by zero, since the target could have been zero sometimes, we decided to refine the formula as the following, where ε represents a smoothing factor to avoid the division by zero.

$$\frac{y_i - \hat{y}_i}{y_i + \varepsilon} \quad (3)$$

As shown in Table II, the model demonstrates a high degree of precision in capturing the underlying queue dynamics. This assertion is further substantiated by the low mean square error (MSE) value, which serves as a metric for evaluating the accuracy of the model's predictions. Additionally, the R^2 score value is noteworthy, particularly in the context of complex network systems, where delays are influenced by multiple stochastic factors and achieving a high variance is challenging. It is imperative to note that a key observation regarding the MAPE value must be made, as this may appear less favourable than expected. This metric imposes a significant penalty on predictions during periods of low traffic volume, even in circumstances where absolute errors are negligible. The MAPE of 47.66% indicates that the model performs optimally in predicting delays during high-delay network conditions, but exhibits reduced precision under low-delay scenarios, where small absolute errors translate to disproportionately large percentage errors. Fig. 1. is intended to demonstrate the performance of a predictive model by means of a comparison between its predictions and the actual observed values over time or across a sequence of samples. The horizontal axis denotes the sample index, signifying the progression of predictions, whilst the vertical axis corresponds to the target variable, i.e. queue delays. The observed queue delays are represented by the blue data points, while the model's predictions are shown as the orange data points. From a visual perspective, the two curves frequently follow a comparable trajectory. This finding suggests that the model is generally effective in capturing the underlying trends present in the data. However, discrepancies become more noticeable in regions characterized by sharp spikes. In such contexts, the model has a tendency to either underestimate or overestimate delays, suggesting difficulties in managing more volatile or extreme values.

IV. EXPERIMENT REPRODUCIBILITY - FULL DATASET

Building upon the insights obtained from the toy dataset experiment, this section presents our comprehensive experimentation using the full dataset, with the objective of reproducing the DeepQueueNet experiment.

A. Dataset Definition

The dataset under consideration constitutes a representation of Poisson traffic entering and exiting a six-node router. It is composed of 1,600 simulations, with each simulation requiring up to 1 GB of disk space, thereby resulting in a total dataset

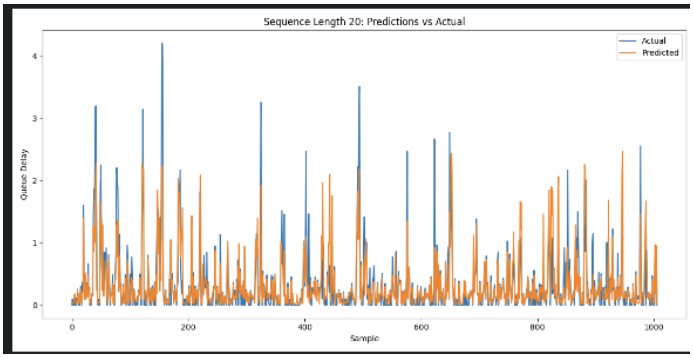

Fig. 1. Prediction vs ground truth using the toy dataset

size of 1.6 TB. This substantial size poses a considerable challenge to the steps of preprocessing and training.

B. Dataset Creation

The objective of this section is to analyze the provided dataset and to elucidate the methodology employed in its processing prior to preprocessing and model training. The dataset is structured as a 6x6 matrix, with each row representing a source port and each column corresponding to a destination port. In order to extract the relevant data, the values in each row of the matrix were iterated over in order to collect packet information, which was then stored in CSV format. The following features were extracted for each packet:

- simulation id : placeholder of the simulation
- source port : represents the port from where the packet is coming
- arrival timestamp : represents the specific timestamp when each individual packet arrives in the system
- packet size : represents the length of the packet
- queue delay : representing the target feature of the dataset

It is imperative to acknowledge the inherent characteristics of the simulation process, as it may potentially result in the loss of packets, consequently leading to the absence of delay values. In such cases, any row containing a 'none' value for delay will be excluded during the preprocessing stage, prior to the commencement of training. The initial phase of the reproducibility experiment resulted in the mapping of each simulation to a specific CSV file, with each file being saved in a designated folder. The folder thus serves as the data source for subsequent processing steps

C. Preprocessing

This section provides a detailed description of the preprocessing steps applied to the dataset. The primary challenge pertained to the inefficiency and slowness of the conventional approach, which was due to the dataset requiring approximately 1.6 TB of disk space. Consequently, a decision was taken to divide all simulations into batches of 10, with a view to enhancing parallelization and accelerating the process without compromising the integrity of any simulation data. With the exception of this key modification to the implementation, the majority of the steps are analogous to those performed in the toy dataset experiment: feature augmentation, scaling, sequence creation, and splitting into training, validation, and test sets. The augmentation of features was initiated, with the inter-packet gap and workload features being computed. In the context of network traffic modeling, workload is defined as a smoothed estimate of the traffic intensity experienced by a router over time. In lieu of reliance upon instantaneous packet sizes, which have been observed to fluctuate significantly and thus introduce noise, the workload was computed using an exponential weighted moving average (EWMA).

$$\text{workload}_i = \alpha \cdot \text{workload}_{i-1} + (1-\alpha) \cdot \text{pk\_size}_i \quad (4)$$

where α represents a smoothing factor between [0,1], if α = 0 the workload relies only the packet size while α = 1 means that the workload is solely depending on the workload of the previous packet. According to the DeepQueueNet paper [1], the value is set to α = 0.95. Once the dataset was enriched with the required features, all variables were scaled using StandardScaler (using the Sklearn library). In network delay prediction tasks, input features such as packet sizes, inter-arrival times, or instantaneous workloads often exhibit heterogeneous scales and statistical properties, using this strategy we ensure stable learning because each feature is scaled to have zero mean and unit variance. Compare to the Min-Max scaling we have used in the toy experiment dataset, using the StandardScaler approach is more suitable for this application domain since network traffic data is often non-stationary and heavy-tailed, with outliers and bursts that skew the minimum and maximum.

$$x' = \frac{x - \mu}{\sigma} \quad (5)$$

where μ represents the mean and σ the variance. Finally, the preprocessed dataset was split into training, validation, and test sets. This was done by computing a training index and partitioning the data accordingly: 80% for training, 10% for validation, and the remaining 10% for testing, mirroring the approach used in the toy dataset experiment.

D. Architecture

In this section, we present the final version of the training and evaluation steps, which follows the architectural pipeline proposed by the authors of DeepQueueNet [1]. As described in the original paper, the architecture is a neural network designed for sequence-to-sequence (seq2seq) tasks. Each network device is modeled as a transformer of packet streams, with the goal of predicting the queuing delay experienced by each packet as it traverses the device. To effectively capture both short and long-term dependencies in the data, DeepQueueNet integrates two key components: Bidirectional Long Short-Term Memory (BiLSTM) networks and a multi-head attention mechanism. [8] In the context of router queuing delay prediction, BiLSTM is useful because queuing behavior often depends not only on past traffic patterns but also on future or concurrent traffic trends. A BiLSTM [7] can learn temporal dependencies in both directions, capturing both causes and effects. Multi-head attention further improves prediction by

allowing the model to dynamically focus on relevant time steps or traffic features when estimating delay using the attention mechanism explained in detail in the original paper. [8] The training process begins by loading the datasets training, validation (initially), and testing (subsequently), from their respective directories. Given the large size of the dataset, reading all data at once would be inefficient.

1) Sequence Creation: The sequence creation logic employed in this instance is analogous to that utilized for the toy dataset. However, it has been encapsulated within the generator with a view to optimizing the process, given the considerable number of packets requiring sequencing. In order to elucidate the matter, it is imperative to state that the generator is a framework that is provided by Python with the purpose of optimizing the process of loading and streaming data into a Tensorflow model, without the risk of the device's memory being exceeded. The generator produces a data object, which is then converted into a TensorFlow Dataset format to ensure compatibility with the training pipeline.

2) Hyperparameters choice: The subsequent stage in this process is model instantiation. In order to accomplish this objective, the configuration parameters provided in the original DeepQueueNet source code [1] were used, including the learning rate and regularization settings. Due to the substantial volume of data, the model was trained for a relatively small number of epochs and a low number of steps per epoch to maintain computational feasibility. The implementation of the model architecture was accomplished using the most recent versions of TensorFlow and Keras, which were used for the construction of the BiLSTM and multi-head attention layers. The final model architecture consists of three main components: an encoder module composed of a bidirectional long-short-term memory (LSTM) layer, a multi-head attention mechanism, and a decoder module implemented as another LSTM layer. Here we can see the in Table III with the hyperparameters used in the architecture and training process:

TABLE III
MODEL HYPERPARAMETERS

Parameter	**Value**
BiLSTM Layer 1	200 units
BiLSTM Layer 2	100 units
Attention dim	64
Learning Rate	0.001
Epochs	3
time steps	21
L2 regularization	1e-5
batch size	256

E. Results - Full Dataset

This subsection presents a comprehensive set of results obtained from a full experimental investigation using the entire dataset. It is imperative to emphasize that this result does not accurately reflect the model's actual capabilities. This is due to the use of a negligible proportion of the model, owing to the substantial time requirements for training and evaluation. The estimation suggests that use the entirety of the samples per epoch in the training phase will require approximately 120 hours per epoch. This calculation, when applied to the current configuration of three epochs, yields a total of 360 hours allocated for the training phase. Due to time constraint we decide to training the model using a total of 10,000 steps per epoch for a total of 3 epochs, which results in 6 hours per epoch.

Table IV presents the overall performance metrics achieved on the full dataset.

TABLE IV
PERFORMANCE RESULTS ON TOY DATASET

Two-layers LSTM model	MSE	R^2	MAE	MAPE
DeepQueueNet Configuration	0.0685	0.2838	0.1170	67.66

In Table IV, we observe that the model demonstrates reasonable capability in capturing the underlying queue dynamics, with an MSE of 0.0685 indicating relatively good prediction accuracy. However, the R^2 score of 0.2838 suggests that the model explains only about 28% of the variance in the actual data, which is notably lower compared to the toy dataset performance. This reduced R^2 score can be attributed to several factors. First, the real-world dataset presents more complex patterns and higher variability compared to the synthetic toy dataset. Second, the model may require additional training epochs to fully capture the intricate queue dynamics present in this more challenging scenario. The time constraints mentioned prevented optimal hyperparameter tuning and extended training periods. The MAE of 0.1170 and MAPE of 67.66% provide additional insights into the model's performance. While the MAE indicates that predictions deviate from actual values by approximately 0.12 units on average, the relatively high MAPE suggests significant percentage errors, particularly problematic when actual queue lengths are small (near zero), which can inflate this metric substantially. Examining the Figure 2 reveals that as in the toy dataset, the model faces greater challenges with the delay spikes. While the model successfully learns the general underlying patterns and baseline queue behavior (as evidenced by the orange predicted values following the general trend), it consistently struggles to predict sudden spikes and extreme values in queue length. The model tends to underestimate peak queue lengths, particularly evident around samples 600-650 where actual values reach above 4 units while predictions remain below 2 units. This conservative prediction behavior suggests the model has learned to predict closer to the mean queue length, which helps minimize MSE but fails to capture the full dynamic range of queue variations. This limitation in spike prediction is critical for practical queue management applications, both present in the toy and full dataset, where accurately forecasting peak congestion periods is often more important than predicting steady-state conditions.

F. Reproducibility Challenges

Our study reveals several critical challenges in reproducing deep learning-based network models:

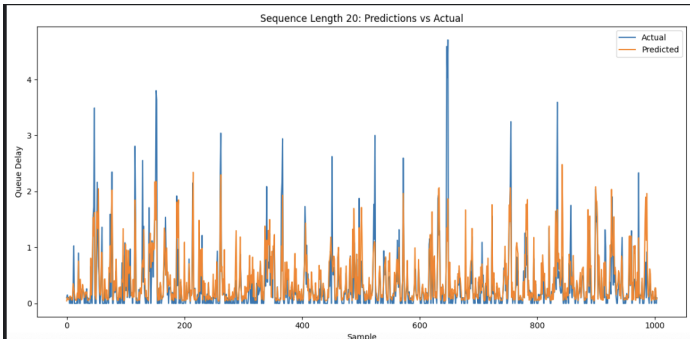

Fig. 2. Prediction vs ground truth using the full dataset

Computational Requirements: The scale of realistic network datasets demands significant computational resources and time, often limiting the depth of experimental validation possible in academic settings.

Data Processing Complexity: The preprocessing pipeline for temporal network data requires careful consideration of sequence boundaries and temporal consistency.

Hyperparameter Sensitivity: Network delay prediction models exhibit high sensitivity to architectural choices and hyperparameters, requiring extensive validation across different scenarios.

V. Limitations

This study has several limitations that should be acknowledged:

Limited Training: Time constraints prevented full convergence on the realistic dataset, limiting our ability to assess ultimate model performance.

VI. Conclusions

In conclusion the reproducibility study of DeepQueueNet provides valuable insights into both the potential and challenges of deep learning-based network performance prediction. Our two-phase experimental approach successfully demonstrates the model's capability to learn queue dynamics while highlighting critical reproducibility challenges.

The toy dataset experiments confirm that properly configured LSTM architectures can capture temporal patterns in queue behavior, achieving meaningful predictive performance, the main challenge was to create the sequence ensuring independence between simulations. However, the transition to realistic, large-scale datasets reveals significant computational and methodological challenges that has been encountered trough the reproducibility process. In particular the scalability of using a deep learning model applied to a real-case dataset scenario need more time resources than expected. The main challenge was to change the way of implementing the pipeline to be executable using this 6 port router simulation massive dataset. Lastly our findings emphasize the importance of computational considerations in reproducibility studies and highlight the need for more efficient training methods for large-scale network modeling applications.

VII. Future Work

Several directions emerge from this study:

Computational Optimization: Developing more efficient architectures and training procedures to handle large-scale network datasets within reasonable computational budgets.

Online Learning: Investigating online learning approaches that can adapt to changing network conditions without requiring complete retraining.

Hyperparameters tuning: Trying a different set of hyperparameter for the final model in order to achieve a final better result.

References

[1] Q. Yang et al., "DeepQueueNet: Towards Scalable and Generalized Network Performance Estimation with Packet-level Visibility," in *Proc. IEEE INFOCOM*, 2021.

[2] L. Kleinrock, *Queueing Systems Volume 1: Theory*. New York: Wiley, 1975.

[3] K. Rusek et al., "RouteNet: Leveraging Graph Neural Networks for Network Modeling and Optimization in SDN," in *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2260-2270, 2020.

[4] N. Zilberman et al., "MimicNet: Fast Performance Estimates for Data Center Networks with Machine Learning," in *Proc. ACM SIGCOMM*, 2021.

[5] K. Rusek et al., "Unveiling the potential of Graph Neural Networks for network modeling and optimization in SDN," in *Proc. ACM SOSR*, 2019.

[6] S. Smith et al., "Deep Learning for Network Traffic Prediction: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2852-2901, 2021.

[7] S. Hochreiter et al., "Long-Short Term Memory,Neural Computation 9(8):17351780, 1997

[8] A.Vaswani, et al., "Attention is all you Need", 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA