

Efficient and Accurate Network Modeling with ML

Regina Bayer, Johannes Späth*

**Chair of Network Architectures and Services*

School of Computation, Information and Technology, Technical University of Munich, Germany

Email: regina.bayer@tum.de, spaethj@net.in.tum.de

Abstract—Efficient and accurate network models are critical tools to analyze a wide range of network scenarios, enabling performance estimation, capacity planning, and network optimization. Existing network simulators primarily build on traditional modeling techniques, such as queueing theory, fluid models and Discrete Event Simulation (DES). While DES remains the most commonly used method in practice, as it is valued for its high level of accuracy, network simulators exhibit elevated computational costs, limiting their scalability in large-scale network scenarios. In recent years, machine learning has emerged as a promising alternative to DES-based simulation, offering significantly faster computation. Among various machine learning architectures, Graph Neural Networks (GNNs) have shown particular potential due to their ability to naturally represent network topologies and capture complex dependencies of network elements as graphs. This paper provides a broad literature overview of recent GNN-based network modeling approaches, focusing on RouteNet-Gauss and TwinNet, and discusses their respective strengths and limitations. Additionally, it highlights the challenges of obtaining high-quality datasets for training machine learning models, and outlines complementary techniques, such as transfer learning and machine-learning-based dataset optimization, that aim to bridge the gap between simulated and hardware-measured data.

Index Terms—network modeling, graph neural networks, discrete event simulation, transfer learning, simulation-to-hardware gap

1. Introduction

A high-quality network model is critical for solving optimization problems, conducting network research, and performing capacity planning, as it enables the prediction of key performance metrics such as latency, packet loss and jitter. Therefore, a network model should satisfy two main criteria:

- 1) It must be computationally efficient, so that many network scenarios can be evaluated within a short period of time.
- 2) It must be accurate, since effective optimization is only possible when the underlying model reflects real network behavior.

Although techniques like queueing theory offer valuable insights, they often rely on strong assumptions that limit their utility in highly complex modern networks [1]. DES-based network simulators, with commonly used simulators such as OMNeT++ [2] and ns-3 [3], produce more

accurate results, but at a high computational cost, limiting their scalability and therefore rendering them unsuitable for large-scale network scenarios [1].

In recent years, several machine-learning-based approaches have been proposed to address these challenges. In particular, GNN-based machine learning methods have demonstrated promising results due to their inherent ability to operate on graph-structured data. Notable recent advances that leverage GNNs, discussed in this paper, include RouteNet-Gauss [4] and TwinNet [1]. While the machine-learning-based network models discussed in this paper significantly improve computational efficiency, they still require large and realistic datasets for training, as their accuracy is largely dependent on a high-quality dataset. However, acquiring real network datasets is costly, poses privacy concerns, and testing scenarios directly in production networks is inherently risky. The lack of high-quality real-world datasets remains one of the key obstacles in developing robust machine-learning-based network models, potentially limiting their ability to generalize over scenarios not seen during training. Training models on simulated data offers a cheaper alternative to hardware-measured datasets. However, those models output less accurate results when applied in real-world scenarios, as DES-based simulators struggle to fully capture hardware-dependent behavior and probabilistic events.

This paper provides a broad overview of recent advances of machine-learning-based network modeling, presented in recent literature. Furthermore, the paper discusses the issues of obtaining meaningful datasets and presents recent approaches that aim to overcome the gap between measured and simulated datasets.

Section 2 introduces the functionality and limitations of DES-based simulation, as well as the fundamental principles of GNNs. Furthermore, it explores the reason why GNN-based network models are more efficient than machine learning models that rely on traditional Neural Networks (NNs). Section 3 presents several GNN-based network modeling approaches and compares their characteristics. Finally, Section 4 discusses the challenges of obtaining meaningful datasets for training machine-learning-based models and presents potential solutions.

2. Background

This section provides detailed information on DES and commonly used network simulators. It further introduces the general architecture of GNNs and highlights their advantages when modeling networks compared to traditional Neural Networks NNs.

2.1. The Functionality of DES-based Simulation

The fundamental principles of DES are described by James Gross and Mesut Güneş [5]. DES simulators are widely used for network analysis and performance evaluation, as the simulation paradigm of DES aligns well with the structural characteristics of computer networks. In DES, relevant components, such as packets, are represented as entities, each associated with a set of attributes, such as a source address, destination address and length. From these entities and their interactions, a system is then constructed. The system is then updated in discrete time steps by processing events, such as packet transmission or packet reception. After termination, the final state of the simulation can be evaluated with respect to predefined performance metrics.

Common DES-based simulators, used in both industry and academic research, include for instance OM-NeT++ [2] and ns-3 [3]. Both simulators are valued for their packet-level visibility and fine-grained detail, as they model each individual packet throughout the simulation. However, this detailed per-packet simulation also limits the scalability of DES-based network simulators, making them computationally expensive for large-scale network scenarios.

Several approaches attempt to reduce the computational cost of DES-based simulation by introducing parallelization techniques (e. g. DONS [6]) or by incorporating machine learning models (e. g. DQN [7]). Although these approaches demonstrate promising results in reducing simulation time, DES-based simulation methods remain expensive, as shown by Carlos Güemes-Palau et al. [4] in their work.

2.2. The Simulation Gap: Idealized Models vs. Real Hardware

While DES simulation is conceptually accurate, multiple researchers have shown that performance metrics produced by DES simulators deviate from measurements obtained on real hardware testbeds [1], [4], [8], [9]. One source of inaccuracy is the limited availability of precise device specifications and implementations due to commercial interests or lack of documentation [4]. Another source stems from the inherent idealization of real network traffic, as the precise representation of real computer networks would require a highly complex simulation design. For example, simulators face difficulties when modeling the network stack or accounting for stochastic processes, such as interrupts [8].

The network simulator ns-3 mitigates some of these issues by designing the model's implementation closer to the actual software implementations they represent, for example by using C++ as a programming language, by incorporating an architecture similar to the Linux Kernel or by emphasizing emulation capabilities [3]. However, these advances do not fully eliminate the inherent challenges associated with network simulation.

2.3. The Concept of Graph Neural Networks

GNNs, as presented by Franco Scarselli et al. in their work [10], represent a general class of neural network ar-

chitectures whose defining characteristics are their ability to operate directly on data given in the form of complex graph structures. A graph is a data structure consisting of a set of objects, referred to as nodes, and the relationships or interactions between them, encoded as edges. Nodes and edges may be associated with a label expressed as a vector of reals which can encode various features or properties of their corresponding concepts. Consequently, each node is characterized by its own feature vector and by the feature vectors of its neighboring nodes, which together define the state of the node.

GNNs impose no assumptions about the specific appearance or the ordering of the input graph and treat isomorphic graphs identically. As a result, GNNs naturally support a broad range of graph types, including cyclic, acyclic, directed and undirected graphs. GNNs' ability to generalize across diverse graph topologies stems from this invariance.

The core functionality of GNNs typically consists of two phases: message-passing and readout.

A) *Message-Passing Phase*: Message passing is an iterative procedure in which the states of two neighboring nodes, optionally combined with the label of their connecting edge, are used to compute a message vector that encodes their relationship. After the computation of all messages, an aggregation function combines the incoming messages for each node into a single vector, ensuring a fixed-size representation that is independent of the number of neighbors. Finally, the node's state is updated by combining the aggregated messages from its neighbors with its previous state.

In summary, the message-passing mechanism propagates information iteratively over the entire graph for a fixed number of iterations or until a convergence criterion is met.

B) *Readout Phase*: During the readout phase, the updated node states are transformed into the final output of the GNN. A GNN can output node-level predictions by applying a readout function to each node individually, or global graph-level predictions by first aggregating the states of all nodes and then applying the readout function to the resulting global representation.

The message, update and readout functions are typically approximated using smaller neural network architectures, such as Multi-Layer Perceptrons (MLPs), Gated Recurrent Units (GRUs) or Recurrent Neural Networks (RNNs). [1]

As computer networks can be naturally represented as graphs, GNNs are frequently used to model them and to generate meaningful performance-related estimations, including metrics such as latency and packet loss.

2.4. Advantages of GNNs Compared to Traditional Neural Networks

As stated by Krzysztof Rusek et al. in their work [11], early approaches that incorporated machine learning into network modeling have primarily relied on traditional NNs, including fully-connected NNs, Convolutional Neural Networks (CNNs) [12], which operate on fixed grids

(e.g. in image processing), RNNs [12], which are designed to process sequential data (e. g. in video, text or speech recognition), as well as Variational Auto Encoders. The reason why those models exhibit lower accuracy than GNN-based models stems from the nature of the problem itself: Computer networks are naturally represented as complex graph structures with circular interdependencies. Since traditional NNs cannot operate directly graph-structured data, they fail to generalize to topologies not seen during training.

For this reason, we focus exclusively on network modeling approaches that leverage GNNs.

3. Different Machine Learning Approaches

After introducing the fundamental concepts of DES-based simulation and GNNs, two novel machine learning models for efficient network modelling will be presented. The focus lies on outlining and comparing the different approaches.

3.1. TwinNet: A Digital Twin for Network Optimization

TwinNet is a GNN-based network model introduced by Miquel Ferriol-Galmés et al. [1]. As the authors focus on network optimization, TwinNet is designed to capture complex network interrelationships, including different queueing policies, network topologies and routing configurations. TwinNet can be used to predict relevant network performance metrics, including jitter, packet loss and end-to-end path delays, meaning that TwinNet produces flow-level predictions rather than packet-level prediction. However, the authors mainly evaluated the model's ability to estimate path delays in their work, not focusing on the other performance metrics.

TwinNet incorporates the GNN architecture by representing links, queues and source-destination paths as nodes and by assigning feature vectors to each of these elements. The different components exhibit a circular dependency, as shown in Figure 1.

- 1) The state of each link depends on the states of all queues injecting traffic into the link.
- 2) The state of each queue depends on the states of all paths injecting traffic into the queue.
- 3) The state of each path depends on the states of all queues and links it traverses.

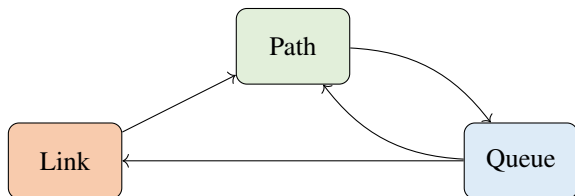


Figure 1: Network components modeled by TwinNet and their dependencies.

To then resolve the interdependencies among links, queues and paths, TwinNet extends the standard GNN framework by implementing a more complex message passing algorithm that updates each node type (link, queue, path) using messages received from the node types that it depends on.

Since the order in which a path traverses links and queues is essential for determining the path's state, RNNs are employed to aggregate link and queue messages, allowing the model to capture the sequential dependencies inherent to these elements. The queues' states are aggregated in a similar way. RNNs are a simpler class of neural network architectures that are well suited for modeling sequential or temporal domains [12].

TwinNet is trained on simulated data produced by a DES-based network simulator (OMNeT++ v5.5). The dataset consists of multiple network scenarios with randomly assigned topology, traffic, routing and queueing configurations and its corresponding per-packet delay measured on each source-destination path.

Limitations of TwinNet lie in both the simulated dataset and the lack of evaluation of jitter and packet loss.

3.2. RouteNet-Gauss: Hardware Enhanced Flow-Level Modeling

RouteNet-Gauss is a machine-learning-based network modeling approach introduced by Carlos Güemes-Palau et al. in [13]. It constitutes the most recent iteration of the RouteNet model family [13]–[15]. To improve computational efficiency, RouteNet-Gauss operates at flow-level granularity rather than at packet-level like DES-based simulators. However, the authors argue that the reduced granularity does not diminish the model's ability to facilitate network operation tasks such as capacity planning, Quality of Service (QoS) assurance, topology design, traffic engineering, and performance estimation.

To further address the absence of packet-level visibility, RouteNet-Gauss introduces a mechanism called Temporal Aggregated Performance Estimation (TAPE), designed to capture the temporal dependencies of flows. TAPE works by dividing the network scenario into temporal windows of configurable size, aggregating traffic information of each window and processing every window individually. RouteNet-Gauss takes a network scenario consisting of packet-level traffic information, the network topology and routing configurations as input.

The model represents the complex behavior of computer networks by modeling devices, queues, flows and links as individual entities. The components share a circular dependency, as shown in Figure 2.

- 1) The devices interact with the queues they are home to.
- 2) The queues depend on the resources provided by their networking device, the flow they process and the links they serve.
- 3) The flows interact with a sequence of queues and links on their path.
- 4) The links depend on the queues that inject traffic into the links.

Each of these elements is implemented using a dedicated NN module as a building block, enabling RouteNet-Gauss to learn the general principles regarding the interactions among network components.

RouteNet-Gauss encodes the initial state of each component using an MLP. During the message passing phase, each component type (device, queue, link, flow) is updated based on the relevant components that they depend on,

TABLE 1: Comparison of TwinNet and RouteNet-Gauss

Work	Ref	Year	Data Source	Training Topologies	Unseen Test Topologies	Granularity	Target
TwinNet	[1]	2022	SIM	2	106	flow-level	jitter, packet loss, delay
RouteNet-Gauss	[4]	2025	HW	11	> 11	flow-level, temporal window	jitter, packet loss, delay

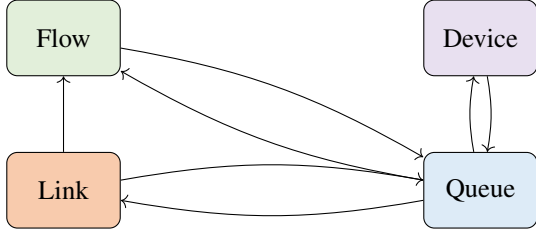


Figure 2: Network components modeled by RouteNet-Gauss and their dependencies.

using GRU cells to model their respective update functions. To incorporate the temporal aspect of RouteNet-Gauss, the message passing algorithm is executed for each time window. The final readout function, which outputs the estimated flow-level performance metrics of the network scenario per temporal window, is approximated using MLPs. The output of the model includes critical performance metrics, such as end-to-end delay, packet loss, and queue occupancy.

RouteNet-Gauss is trained on a dataset captured from a physical testbed consisting of 11 distinct topologies, ranging from five to eight nodes, in order to address the issue of inaccuracies presented in simulated data. However, as the dataset is relatively small, it could hinder RouteNet-Gauss’s ability to generalize to modern large-scale networks.

A comparison of the two network models can be seen in Table 1.

4. Dealing with Datasets

Machine learning approaches fundamentally rely on meaningful datasets to produce accurate estimations. Accordingly, the training dataset must satisfy three essential criteria, as described in [9]:

- 1) *Abundance*: Training datasets must contain a sufficiently large number of samples to ensure the model’s ability to generalize over scenarios not seen during training.
- 2) *Diversity*: Training data must be diverse, comprising a broad spectrum of network conditions including rare edge cases that may nonetheless play a significant role in real-world computer networks.
- 3) *Completeness*: Training data must cover all relevant factors and variables influencing network behavior.

Collecting real-world datasets is often challenging, as measured data tends to be scarce, incomplete and insufficiently diverse, since many edge cases cannot be safely captured on production networks, and acquiring data on hardware testbeds is prohibitively expensive, particularly for large-scale networks. Relying exclusively on simulated data mitigates the issue of data scarcity and can generate a large, diverse and complete dataset. However, for the reasons discussed in Section 2.2, simulated data diverges from real-world conditions, leading to inaccurate results.

This section introduces transfer learning [9] and Sim2HW [8] as potential solutions for these challenges, by benefiting from the advantages of both simulated and hardware-measured datasets.

4.1. Transfer Learning: From Simulation to Reality

The concept of transfer learning, as introduced by Carlos Güemes-Palau et al. in [9], is to reuse the weights of a pre-trained donor model to provide a strong starting point for initializing and training a receiver model, thereby mitigating the challenges associated with acquiring meaningful datasets. For network modeling, the authors propose a hybrid approach that combines both simulated and real-world data.

First, they trained RouteNet-Fermi [15], a predecessor of RouteNet-Gauss, solely on a large, diverse and complete simulated dataset acquired by the OMNeT++ network simulator, covering network topologies ranging from five to eight nodes. Afterwards, a significantly smaller real-world dataset was collected by extracting samples from a physical testbed.

The example model, as well as most GNN-based architectures can be divided into three main components: An encoding block, a message passing block and a readout block. This type of fragmentation is essential for the effective use of transfer learning, as it enables the specific configuration of a weight-transfer strategy. According to the authors, the most effective manual transfer learning configuration is to freeze the encoding block, fine-tune the message passing block and to retrain the readout block. Following this approach, only the encoding and message passing blocks of the donor model are transferred to the receiver model.

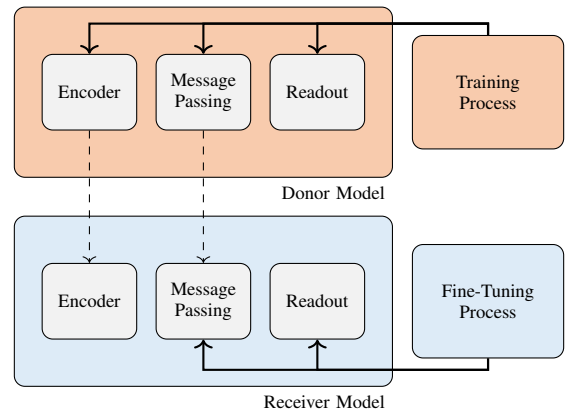


Figure 3: Illustration of the best manual transfer learning configuration for network modeling. The weights of the encoding block and message passing block are transferred from donor model to receiver model. Subsequently, the message passing and readout block of the receiver model are fine-tuned /retrained based on the received initial weights.

The receiver model must then acquire its own readout block through the standard training procedure. During

fine-tuning the receiver's encoding block remains frozen to preserve the transferred initial weights, as modifying the encoding block would invalidate the benefits of transfer learning. Fine-tuning the message passing block proceeds in a similar way to training a model from scratch, but requires substantially less data and training time. Automated transfer strategies, such as Autofreeze [16], L2-SP [17] and GTOT-Tuning [18] were also evaluated and generally outperform the best manual configuration.

In conclusion, transfer learning techniques appear to be a promising approach for improving the accuracy of machine learning based network models while keeping the costs of acquiring real-world datasets low, as transfer learning allow models to benefit from both the abundance and diversity of simulated and the fidelity of real-world datasets simultaneously.

4.2. Sim2HW: Correcting Simulation with GNNs

Sim2HW is a machine learning dataset optimization approach introduced by Johannes Späth et al. [8]. It addresses the problem of inaccuracies in simulated datasets by correcting simulation outputs using a GNN-based model. The primary objective of the Sim2HW model is to learn the discrepancy between simulated and hardware-measured datasets. The authors rely on a dataset obtained from a physical hardware testbed by Wiedner et al. [19]. In addition, they replicate the network scenarios present in the physical dataset using the OMNeT++ simulator. Both datasets therefore contain latency values for the same topologies and flow configurations. The flow latency values are then aggregated into percentile metrics, ranging from the minimum percentile to the percentile of 99.999. A percentile p indicates that p percent of the observed latency values are below the given value. For example, the 50th percentile, also referred to as the median, denotes the latency value below which 50% of the flows fall. Using percentiles as target prediction metric allows Sim2HW to capture the distribution of latency values rather than estimating just the mean latency. After converting the network topology, flow properties and latency percentiles from both datasets into a graph representation, Sim2HW's GNN model is trained on this graph. Once trained, the model can be used to infer the latency distribution that would have been obtained on a hardware testbed when provided with simulated latency samples as input.

Sim2HW is particularly effective at estimating performance values for lower percentiles, meaning that the model performs best when predicting flow latencies on lightly used links, where the latency values are small. Its accuracy decreases for high latency values, as such extreme outliers are sparsely represented in both training sets, limiting the model's ability to learn their behavior. Therefore, future iterations of Sim2HW should utilize a more diverse hardware dataset.

5. Conclusion

The reviewed GNN-based network models demonstrate significant potential to improve the efficiency of network modeling compared to traditional DES-based approaches. However, the effectiveness of these models remains heavily dependent on the availability of high-quality

datasets. While simulated data provides abundance and diversity, it often diverges from real-world measurements, potentially limiting the model's accuracy. Techniques such as transfer learning and Sim2HW offer promising strategies to mitigate these challenges by combining the advantages of simulated datasets with the fidelity of real-world measurements.

References

- [1] M. Ferriol-Galmés, J. Suárez-Varela, J. Paillissé, X. Shi, S. Xiao, X. Cheng, P. Barlet-Ros, and A. Cabellos-Aparicio, "Building a digital twin for network optimization using graph neural networks," *Computer Networks*, vol. 217, p. 109329, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1389128622003681>
- [2] A. Varga, *A Practical Introduction to the OM-NeT++ Simulation Framework*. Cham: Springer International Publishing, 2019, pp. 3–51. [Online]. Available: https://doi.org/10.1007/978-3-030-12842-5_1
- [3] G. F. Riley and T. R. Henderson, *The ns-3 Network Simulator*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 15–34. [Online]. Available: https://doi.org/10.1007/978-3-642-12331-3_2
- [4] C. Güemes-Palau, M. Ferriol-Galmés, J. Paillisse-Vilanova, A. López-Brescó, P. Barlet-Ros, and A. Cabellos-Aparicio, "Routenet-gauss: Hardware-enhanced network modeling with machine learning," 2025. [Online]. Available: <https://arxiv.org/abs/2501.08848>
- [5] J. Gross and M. Güneş, *Introduction*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 1–11. [Online]. Available: https://doi.org/10.1007/978-3-642-12331-3_1
- [6] K. Gao, L. Chen, D. Li, V. Liu, X. Wang, R. Zhang, and L. Lu, "Dons: Fast and affordable discrete event network simulation with automatic parallelization," in *Proceedings of the ACM SIGCOMM 2023 Conference*, ser. ACM SIGCOMM '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 167–181. [Online]. Available: <https://doi.org/10.1145/3603269.3604844>
- [7] Q. Yang, X. Peng, L. Chen, L. Liu, J. Zhang, H. Xu, B. Li, and G. Zhang, "Deepqueue: towards scalable and generalized network performance estimation with packet-level visibility," in *Proceedings of the ACM SIGCOMM 2022 Conference*, ser. SIGCOMM '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 441–457. [Online]. Available: <https://doi.org/10.1145/3544216.3544248>
- [8] J. Späth, M. Helm, B. Jaeger, and G. Carle, "Sim2hw: Modeling latency offset between network simulations and hardware measurements," in *Proceedings of the 3rd GNet Workshop on Graph Neural Networking Workshop*, ser. GNet '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 20–26. [Online]. Available: <https://doi.org/10.1145/3694811.3697820>
- [9] C. Güemes-Palau, M. Ferriol-Galmés, J. Paillisse-Vilanova, A. López-Brescó, P. Barlet-Ros, and A. Cabellos-Aparicio, "Bridging the gap between simulated and real network data using transfer learning," 2025. [Online]. Available: <https://arxiv.org/abs/2510.00956>
- [10] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.
- [11] K. Rusek, J. Suárez-Varela, A. Mestres, P. Barlet-Ros, and A. Cabellos-Aparicio, "Unveiling the potential of graph neural networks for network modeling and optimization in sdn," in *Proceedings of the 2019 ACM Symposium on SDN Research*, ser. SOSR '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 140–151. [Online]. Available: <https://doi.org/10.1145/3314148.3314357>
- [12] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković, "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges," 2021. [Online]. Available: <https://arxiv.org/abs/2104.13478>

- [13] 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 Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2260–2270, 2020.
- [14] M. Ferriol-Galmés, K. Rusek, J. Suárez-Varela, S. Xiao, X. Shi, X. Cheng, B. Wu, P. Barlet-Ros, and A. Cabellos-Aparicio, "Routenet-erlang: A graph neural network for network performance evaluation," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, 2022, pp. 2018–2027.
- [15] M. Ferriol-Galmés, J. Paillisse, J. Suárez-Varela, K. Rusek, S. Xiao, X. Shi, X. Cheng, P. Barlet-Ros, and A. Cabellos-Aparicio, "Routenet-fermi: Network modeling with graph neural networks," *IEEE/ACM Transactions on Networking*, vol. 31, no. 6, p. 3080–3095, Dec. 2023. [Online]. Available: <http://dx.doi.org/10.1109/TNET.2023.3269983>
- [16] Y. Liu, S. Agarwal, and S. Venkataraman, "Autofreeze: Automatically freezing model blocks to accelerate fine-tuning," 2021. [Online]. Available: <https://arxiv.org/abs/2102.01386>
- [17] X. LI, Y. Grandvalet, and F. Davoine, "Explicit inductive bias for transfer learning with convolutional networks," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 2825–2834. [Online]. Available: <https://proceedings.mlr.press/v80/li18a.html>
- [18] J. Zhang, X. Xiao, L.-K. Huang, Y. Rong, and Y. Bian, "Fine-tuning graph neural networks via graph topology induced optimal transport," 2022. [Online]. Available: <https://arxiv.org/abs/2203.10453>
- [19] F. Wiedner, M. Helm, S. Gallenmüller, and G. Carle, "Hvnet: Hardware-assisted virtual networking on a single physical host," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2022, pp. 1–6.