This paper has been accepted for publication in the IEEE/IFIP Network Operations and Management Symposium, Honolulu, HI, USA, May 2025. This is the author's copy. Copyright resides with IEEE/IFIP.

vNetRunner: Per-VNF Slice Modeling for 5G and Beyond Networks

Muhammad Sulaiman*, Bo Sun*, Mohammad A. Salahuddin*, Raouf Boutaba*, Aladdin Saleh†
{m4sulaim, b24sun, mohammad.salahuddin, rboutaba}@uwaterloo.ca, aladdin.saleh@rci.rogers.com,
*University of Waterloo, †Rogers Communications Canada, Inc.

***Abstract*—The adoption of virtualization in 5G and beyond networks enables the creation of network slices tailored to specific application requirements. While this flexibility is transformative, it introduces new challenges in slice management and orchestration (MANO). AI-based techniques are becoming essential for automated slice MANO. However, their effectiveness relies on the accuracy of network models that map VNF configurations and resource allocations to slice performance. Previous approaches, including simulations, control theory, and machine learning, face limitations such as high computational complexity, limited visibility, large data requirements, or specific use cases, e.g., in data center networks. In this work, we present vNetRunner, a framework for slice modeling using individually trained virtual network function models. We validate our framework using datasets from an open-source 5G testbed, focusing on traffic metrics such as mean delay and throughput. Our results demonstrate that vNetRunner estimates mean packet delay and throughput with Wasserstein distances of 6 ms and 0.554 Mbps, respectively, achieving execution times that are an order of magnitude faster than the state-of-the-art modeling approaches.**

***Index Terms*—5G, Network Slicing, Network Digital Twins, Deep Learning, Performance Estimation**

I. Introduction

Network Function Virtualization (NFV) and Software-defined Networking (SDN) are key enablers of 5G and beyond networks, allowing infrastructure providers to create virtual, isolated networks—referred to as network slices—on shared physical infrastructure. These network slices are composed of Virtual Network Functions (VNFs) deployed across the Radio Access Network (RAN), transport network, and core network, where each VNF is configured to meet specific service requirements.[1] The management and orchestration (MANO) of these network slices faces several challenges, including monitoring of slice KPIs [1], slice admission control and VNF placement [2], resource allocation and scaling [3], and fault detection and mitigation [4]. Given the complexity of end-to-end 5G networks, AI-based approaches are emerging as a dominant solution for automating these tasks. However, such approaches require extensive interaction with the network during model training, and their policies must be validated prior to deployment in real-world settings.

[1]We use the terms 'network' and 'slice', as well as 'device' and 'VNF', interchangeably (e.g., 'network model' vs. 'slice model', or 'per-VNF model' vs. 'per-device model').

Network simulators such as ns-3 [5], OMNeT++ [6], and OPNET [7] offer feasible platforms for training, testing and validating solutions. However, these simulators suffer from significant computation complexity, often requiring hours or even days to generate results, even for relatively small-scale networks [8]. Although several open-source 5G testbeds, such as OAI [9], srsRAN [10], Free5GC [11], and Open5GS [12], offer in-lab deployment options for experimentation, they often prove difficult to set up, configure, and maintain effectively, presenting challenges in terms of scalability and usability.

To address these challenges, recent studies have explored the use of Machine Learning (ML)-based models to represent individual network elements (e.g., [13, 8]), which can then be composed into larger network models. Such approaches have been shown to be particularly effective in data center environments [8]. However, these approaches are often limited to specific use cases (e.g., data centers [8]) or rely on complex, packet-level models that require several minutes for inference [13]. Other studies have proposed monolithic models to represent entire networks [14, 15, 3]. Although these approaches offer significantly faster inference, they often reply on large datasets and often lack generalizability to unseen network topologies or scenarios during model training.

In this work, we tackle these challenges by developing a 5G slice modeling framework that models each VNF individually, which are then composed into an end-to-end slice model. Unlike [13], which emphasizes packet-level modeling, our approach focuses on using flow-level features as the input and output of each VNF, resulting in enhanced efficiency and scalability. This distinction is crucial, as recent algorithms [16, 3, 17] may require tens of thousands of interactions with the network model to converge. Additionally, our model outputs a distribution over flow-level features, making it particularly well-suited for 5G VNFs, where performance can fluctuate significantly due to unpredictable factors, such as VNF implementation, user mobility, and varying network conditions. We summarize our main contributions as follows.

- **Per-VNF slice modeling framework:** We introduce the first device-level modeling framework, called **v**irtual **NetRunner** (**vNetRunner**), for 5G and beyond networks. Our approach enables model composability, allowing seamless replacement, removal, and addition of VNFs within a slice. This modularity enhances the flexibility and scalability of slice MANO compared to traditional network-level models.
- **Comprehensive comparison with state-of-the-art models:**

This paper has been accepted for publication in the IEEE/IFIP Network Operations and Management Symposium, Honolulu, HI, USA, May 2025. This is the author's copy. Copyright resides with IEEE/IFIP.

We compare our device-/flow-level model against a device-/packet-level model, and a network-/flow-level model. These approaches are evaluated based on prediction accuracy for key flow-level metrics, including mean delay and throughput. Our results show that vNetRunner significantly outperforms state of the art in both accuracy and execution time.
- **Incorporation of Mixture Density Networks:** We integrate Mixture Density Networks (MDNs) into our framework, allowing for more robust modeling of the inherent variability and uncertainty in 5G VNFs.
- **Real-world 5G testbed evaluation:** Unlike most existing studies that rely on simulations or synthetic datasets, our models are trained and evaluated using data from a real-world 5G testbed. This ensures the relevance and robustness of our proposed approach in practice.
- **Publicly available containerized deployment and dataset:** To facilitate further research and reproducibility, we provide a containerized deployment of our 5G testbed along with the dataset used in this work.[2] This will allow other researchers to easily deploy the environment and extend our work.

The remainder of this paper is organized as follows. In Section II, we provide a comprehensive review of the relevant literature, and discuss the research gap we aim to address. Section III offers a detailed exposition of the vNetRunner framework. Section IV outlines the implementation details, including our testbed and evaluation setup, while Section V presents and analyzes the results. Finally, we conclude in Section VI, where we also instigate potential future directions.

II. Related Work

Network modeling can be categorized into simulation-based, ML-based, and theoretical-model-based approaches. The theoretical modeling approaches include network calculus, queue-theoretic or control-theoretic network modeling but, as discussed in [18], these approaches either lack the required granularity or are not scalable. Therefore, this section mainly focuses on simulation-based and ML-based approaches.

A. Simulation-based Approaches

AI-driven approaches often necessitate interactions with the network for either training, validation, or fine-tuning. In these scenarios, the network can be substituted with a proxy, which may be a simulation, emulation, or a machine learning (ML)-based model. However, ML-based network models are generally not readily available in practice [16, 19], necessitating the use of network simulators like ns-3 [5] or OMNeT++ [6]. Although these simulators provide a feasible proxy for real network, they come with several significant drawbacks.

Network simulators, such as ns-3 [5], operate at the packet level, which inherently makes them computationally intensive and time-consuming [15, 13]. Consequently, their use is often impractical for both online optimization and offline training of network management policies [20]. The computational requirements are further exacerbated as the complexity and scale of the simulated network grows. In some instances, network simulation can require several orders of magnitude more execution time compared to the real-world duration of the simulated scenario [8].

Moreover, simulators often fall short of capturing real-world network dynamics Liu et al. [16]. Recent works [21] have investigated Bayesian Neural Networks and Bayesian optimization to reduce the simulation to reality gap by selectively optimizing high-discrepancy states allowing more realistic network simulation. However, this process need may need to be repeated for each minor alteration in the network and solutions based on such augmented network simulators still require online fine-tuning [16] on the real network.

B. ML-based Estimators

Machine learning-based network models can be categorized based on their modularity—specifically, device-level versus network-level—and their granularity—packet-level versus flow-level. Device-level modeling approaches, as exemplified by Yang et al. [13], focus on modeling each network device individually, subsequently integrating these models to simulate the entire network. Conversely, network-level modeling approaches [14, 15, 3] employ sophisticated neural network architectures, such as Graph Neural Networks (GNNs), to model the network as a whole. Despite their sophistication, these network-level models often struggle to generalize when the network topology changes. Furthermore, they demand extensive datasets for training due to the vast input space, which includes all possible combinations of VNF configurations and resource allocations in case of end-to-end 5G slices [3].

Some methodologies, such as MimicNet [8], occupy an intermediate position between device-level and network-level modeling. MimicNet constructs cluster-level models that encompass multiple hosts and switches, which can be assembled into a larger topology comprising several clusters. However, this approach is limited to the FatTree topology and is hindered by high computational complexity.

Network modeling approaches can also be categorized based on their prediction granularity. Flow-level modeling techniques [14, 15, 3, 17] focus on predicting flow-level metrics, yet they lack the capability to provide insights into packet-level metrics, such as per-packet delay and packet drop rates. Despite this limitation, flow-level metrics are generally sufficient for most AI applications [3, 16, 19]. In contrast, packet-level modeling approaches, such as those proposed by Yang et al. [13], are capable of predicting detailed packet-level metrics, including per-packet delay. However, this increased granularity comes at the cost of higher computational complexity, resulting in inference times that can range from several seconds to several minutes. This may be too high for algorithms that require thousands of interactions with the network model [17, 16].

Our proposed vNetRunner framework, falls in the device-level and flow-level category, which has not been sufficiently explored. Also, it is potentially the ideal option (cf., Section V) in the context of AI-based 5G slice MANO due to its high accuracy and low computation complexity. One of the

[2]Deployment instructions for the testbed are available on GitHub. The dataset is publicly available on GitHub.

This paper has been accepted for publication in the IEEE/IFIP Network Operations and Management Symposium, Honolulu, HI, USA, May 2025. This is the author's copy. Copyright resides with IEEE/IFIP.

limitations of device-level/flow-level models is their inability to account for different routing configurations, unlike packet-level models, which can handle packet forwarding through tensor multiplication [13]. However, this limitation may be less significant in the context of 5G network slices, as they primarily consist of sequential chains of VNF.

III. vNetRunner Framework Design

In vNetRunner, each VNF is modeled individually, and these individual models are then connected by passing the output of an upstream VNF as the input to the downstream VNF, creating an end-to-end model of a network slice. Fig. 1 provides an overview of the vNetRunner framework.

This design enables the seamless addition, removal, or replacement of VNFs by simply swapping out the corresponding VNF model within the end-to-end slice model. Furthermore, because the input set for a single VNF (i.e., traffic distributions, VNF configurations, and resource allocations) is considerably smaller than that of a monolithic slice model, which must account for all possible combinations across multiple VNFs, the dataset required for training is significantly reduced. This modularity enhances both the flexibility and scalability of the vNetRunner framework.

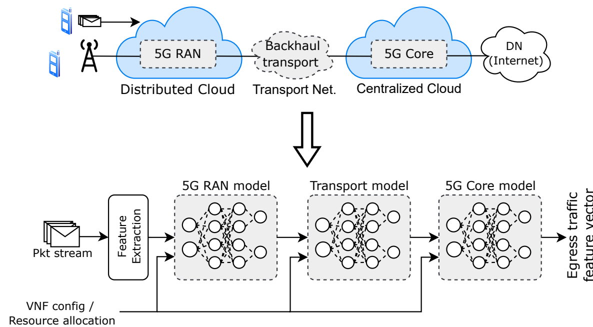


Fig. 1: vNetRunner overview

A. VNF Modeling

The VNF models are trained using datasets generated by injecting traffic into 5G VNFs under various resource allocations and configurations, while capturing both the input and output packet streams (i.e., PCAPs). Following data collection, feature extraction is conducted to convert the raw packet captures into feature vectors suitable for ML models. These feature vectors encompass flow-level metrics, such as packet size, packet rate, throughput, and inter-arrival time, among others. In this work, we propose employing MDNs for VNF modeling, leveraging their capability to represent complex, arbitrary distributions [22].

Mixture Density Networks. Let $\boldsymbol{x} = [x_1, \cdots, x_D] \in \mathbb{R}^D$ represent the traffic feature vector comprising D features, and let $\boldsymbol{r} = [r_1, \cdots, r_R] \in \mathbb{R}^R_{\geq 0}$ denote the resource allocation vector consisting of R resources. Note that $\boldsymbol{r}$ can be extended to include VNF configurations as well. The objective of VNF modeling is to learn the function $f_\theta(\boldsymbol{x}, \boldsymbol{r})$, parameterized by weights θ, which captures the relationship between the VNF's resource allocation $\boldsymbol{r}$, the ingress traffic feature vector $\boldsymbol{x} = \boldsymbol{x}_{in}$, and the egress traffic feature distribution parameters. The output traffic feature vector $\boldsymbol{x}_{out}$ can then be sampled from the distribution specified by $f_\theta(\boldsymbol{x}_{in}, \boldsymbol{r})$.

In our previous work [3], we proposed approximating the function $f_\theta(\mathbf{x}, \mathbf{r})$ using a normal distribution, where the distribution parameters were learned through a neural network. However, a single normal distribution may be insufficient to capture the complexity of real-world data distributions. To address this limitation, we extend our approach by incorporating MDNs. MDNs allow the model to learn the parameters for a mixture of multiple normal distributions, i.e., $\mathcal{N}_k(\mu_k, \sigma_k)$ for $k = 1, \ldots, K$, along with their corresponding mixture weights (π_k), representing the probability that the output is drawn from the kth normal distribution. The resulting distribution can then be expressed as a weighted sum of these distributions:

$$p(\boldsymbol{x}_{out}|\boldsymbol{x}_{in}, \boldsymbol{r}) = \sum_{k=1}^{K} \pi_k \cdot \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left(-\frac{(\boldsymbol{x}_{out} - \mu_k)^2}{2\sigma_k^2}\right), \quad (1)$$

where μ_k and σ_k are functions of $\boldsymbol{x}_{in}$ and $\boldsymbol{r}$.

For a sufficiently large value of K, an MDN is capable of approximating an arbitrary distribution [22]. Let B denote the size of a training batch, and $\{(\boldsymbol{x}^j_{in}, \boldsymbol{r}^j), \boldsymbol{x}^j_{out}\}$ denote the jth data sample. The loss for the model is then computed as:

$$L = -\frac{1}{B} \sum_{j=1}^{B} \log p(\boldsymbol{x}^j_{out}|\boldsymbol{x}^j_{in}, \boldsymbol{r}^j). \quad (2)$$

Differentiable Sampling. The differentiability of VNF model is highly advantageous for approaches that rely on gradient-based optimization, as it allows for seamless integration with optimization techniques that leverage model gradients [3]. If the model is analytically differentiable, existing automatic differentiation libraries, such as [23], can be employed to efficiently compute model gradients with respect to various objectives (e.g., minimizing resource allocation or optimizing VNF configuration for specific traffic distribution). After the model predicts the output traffic feature distribution, the output traffic vector can be obtained via random sampling from this distribution. However, with MDNs, two key operations in the random sampling process are non-differentiable. The first non-differentiable operation is sampling from a categorical or discrete distribution, i.e., the random selection of the kth distribution $\mathcal{N}_k(\mu_k, \sigma_k)$ from which to sample. The second is the random sampling from the selected distribution.

To address the first challenge, we propose the method introduced by Graves [24]. This method leverages a multivariate quantile transform, allowing gradients to be propagated through the mixture weights (π_k) in a differentiable manner. The second challenge can be addressed by employing the reparameterization trick [25], which reformulates the random sampling process in a differentiable manner. This technique

This paper has been accepted for publication in the IEEE/IFIP Network Operations and Management Symposium, Honolulu, HI, USA, May 2025. This is the author's copy. Copyright resides with IEEE/IFIP.

expresses the sampling operation as a linear transformation of the model outputs (i.e., $\boldsymbol{\mu} = \{\mu_k\}_{k=1}^{K}$ and $\boldsymbol{\sigma} = \{\sigma_k\}_{k=1}^{K}$), and a noise term $\boldsymbol{\epsilon} \in \mathbb{R}^D$ drawn from a standard normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{1})$. The reparameterized sample is then computed as $\boldsymbol{x}_{out} = \boldsymbol{\mu} + \boldsymbol{\sigma}\boldsymbol{\epsilon}$.

IV. Implementation

In this section, we describe the implementation of our network slicing testbed, shown in Fig. 2. More detailed description of our testbed is available in [3].

A. Testbed Infrastructure

The testbed is implemented on a three-node Kubernetes cluster. The physical machine hosting the RAN is the most powerful, featuring 32 CPU cores and 32 GB of RAM. The transport network and core are hosted on Intel NUC PCs, each equipped with 8 CPU cores and 16 GB of RAM. The three nodes are interconnected through a 1 Gbps NETGEAR switch.

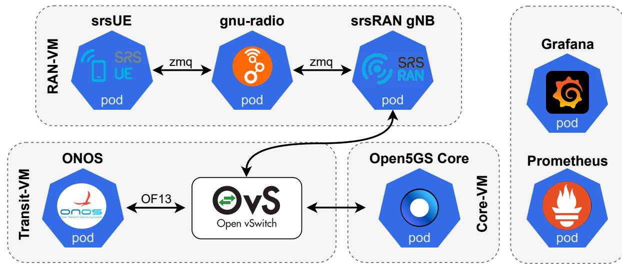


Fig. 2: Overview of our 5G testbed

B. 5G Network Implementation

RAN. The 5G RAN is implemented using the srsRAN project [10], an open-source software that provides a 3GPP Release 17 (R17) compliant gNB. User Equipments (UEs) are emulated with srsUE [26]. Instead of physical radios, we use virtual radios from srsRAN to facilitate communication between the gNB and UEs. Additionally, GNU Radio Companion is utilized to manage uplink and downlink signals.

Core. The 5G core network is based on Open5GS [12], an open-source implementation of 3GPP R17. Network functions, including AMF, SMF, UPF, and NRF, are containerized and deployed on a Kubernetes cluster. In our network slicing scenario, each slice has dedicated UPF and SMF, while other functions such as AMF and NRF are shared.

Transport. The transport network employs a software-defined VXLAN overlay using Open vSwitch (OvS) [27] on the underlying physical network. Integration of the 5G network functions with this overlay is achieved via the OvS CNI plugin for Kubernetes.

C. Management and Control

MANO. Kubernetes v1.29 is used for MANO of the 5G network, encapsulating the various 5G functions, including RAN, in lightweight containers. These containers are dynamically deployed, scaled, and managed across the distributed cluster, offering flexibility and scalability. The Kubernetes API facilitates the placement of network functions and the creation of network slices with desired topologies. Linux cgroups are used to dynamically manage the CPU resources for these network functions.

SDN Controller. The ONOS SDN controller [28] is employed to manage the routing of network flows within network slices. By interfacing with OvS switches in the VXLAN overlay, ONOS directs slice traffic through OvS queues at specific rates, enabling bandwidth slicing capabilities.

D. Dataset Collection

To generate the dataset for training VNF and slice models, we inject Poisson-distributed traffic into the UPF, Transport (OvS), and RAN VNFs. Since open-source per-VNF implementations of the RAN are not yet available, we treat entire RAN as a single VNF. However, vNetRunner can be extended to include per-VNF modeling for the RAN once such implementations become available. Additionally, since the UPF requires significantly lower CPU resources compared to the RAN, we do not perform CPU resource allocation for the UPF. The Poisson traffic is generated with rates ranging from 1 Mbps to 40 Mbps, in 5 Mbps increments. For synchronizing the time between different machines, we use NTP [29]. For each traffic profile, VNF resources are varied, and the resulting output traffic is captured as PCAP files. Data is collected for 60 seconds for each combination of traffic and resource allocation. The captured PCAP files are then pre-processed to extract feature vectors representing flow-level and packet-level metrics, which serve as input and output for the VNF and slice models.

V. Performance Evaluation

A. Comparison Approaches

Network-level Modeling with Flow-Level Metrics: To represent approaches that utilize a single ML model for the entire network, we train a model that predicts output traffic features based on input traffic features and resource allocation. Similar to [14, 15, 3, 17], this model predicts flow-level metrics for the slice's egress traffic given the ingress traffic feature vector and VNF resource allocations. As discussed in Section II, this model requires a large amount of data for training. Therefore, for a fair comparison, we ensure that the size of the dataset used for training this model matches the size of that used to train vNetRunner. We refer to this approach as **Net-Flow**.

Device-level Modeling with Packet-level Metrics: To capture approaches that model each network element individually, we train a separate model for each VNF based on the architecture proposed in [13]. This architecture includes a Bidirectional LSTM for encoding, followed by an attention mechanism, and another Bidirectional LSTM for decoding. Unlike vNetRunner, each device model takes a per-packet feature vector as input and predicts per-packet delay and packet drop. The overall slice model is constructed by sequentially modifying the packet streams using the output of each VNF model, with the

This paper has been accepted for publication in the IEEE/IFIP Network Operations and Management Symposium, Honolulu, HI, USA, May 2025. This is the author's copy. Copyright resides with IEEE/IFIP.

TABLE I: Hyperparameters for vNetRunner and Net-Flow Models

Approach	VNF	Model	Hidden Layers	Num. Gaussians
vNetRunner	RAN	Feature model	[128, 64, 32]	2
vNetRunner	UPF	Feature model	[64, 32]	2
vNetRunner	OvS	Feature model	[128, 64, 32]	2
vNetRunner	RAN	Delay model	[128, 64, 32]	3
vNetRunner	UPF	Delay model	[64]	2
vNetRunner	OvS	Delay model	[128, 64, 32]	3
Net-Flow	N/A	Feature model	[64, 64]	1
Net-Flow	N/A	Delay model	[128, 64, 32]	3

TABLE II: Hyperparam. for Dev-Pkt Models

VNF	LSTM Width	Num. Attn. Heads	Attn. Dim	Window Size
UPF	[128, 64, 32]	3	32	10
RAN	[128, 64, 32]	3	32	10
OvS	[128, 64, 32]	3	32	10

modified packet stream serving as the input for the next VNF in the sequence. The same dataset used to train vNetRunner is employed to train this model. We optimize the model using a combination of mean squared error and binary cross-entropy loss. This method is referred to as **Dev-Pkt**.

B. Hyperparameters

For the flow-level approaches (i.e., Net-Flow, vNetRunner), we train two distinct models for each VNF using the collected dataset (cf., Section IV-D). The first model, referred to as the feature model, follows the design outlined in Section III. The second model focuses on predicting packet delay. Unlike the feature model, the output of the delay model is not used as input to subsequent VNFs. Instead, the delay predicted by each VNF model is accumulated to calculate the total end-to-end delay for the slice. The hyperparameters used for both models are listed in Table I, with their values determined through experimentation and fine-tuning.

For the packet-level models (i.e., Dev-Pkt), we only train a single model per VNF to predict the per-packet delay and drop. The hyperparameters for these VNF models are given in Table II. All the models are trained with a learning rate of 0.0001 for 30K–50K epochs until the loss converges.

To train and validate the VNF and slice models, the collected data is segmented into 1-second windows for feature extraction and prediction. For the VNF models, the test dataset is constructed using combinations of resource allocations and traffic intensities that are not present in the training set. This approach is employed to assess the generalization capability of the models across unseen conditions, ensuring robust performance beyond the training distribution.

C. Performance Evaluation of Per VNF Model

Model Training. In this subsection, we compare the VNF models for vNetRunner (i.e., Dev-Flow) against those for Dev-Pkt. Figures Fig. 3a and Fig. 3b illustrate the training and validation losses for both the vNetRunner and Dev-Pkt approaches, specifically for the RAN VNF model. Note that vNetRunner is trained using negative log probability loss (Eq. (2)), while Dev-Pkt is trained using a combination of mean square error for delay prediction and binary cross entropy for packet drop prediction. From these plots, we observe that the training and validation losses decrease consistently as training progresses and eventually converge to similar levels. This indicates that the models do not suffer from overfitting or underfitting. The VNF models' losses for UPF and OvS exhibit similar trends. Therefore, we omit their discussions for the sake of brevity.

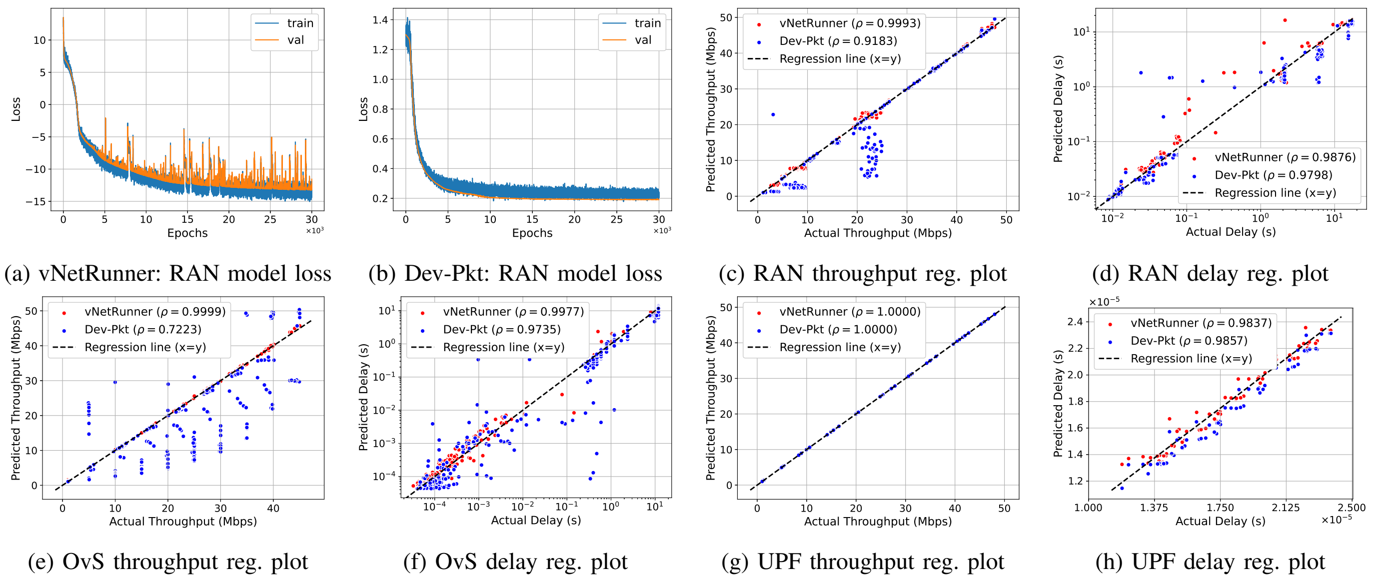
Regression Plots. We validate the models by plotting regression plots in Fig. 3c–3h. In these plots, the x-axis represents the ground truth, while the y-axis shows the predicted average values. Predictions that fall closer to the x = y line are considered more accurate. Since packet delay ranges from milliseconds to tens of seconds, we use logarithmic scales for both axes in the delay plots. The plots also display the Pearson correlation coefficient (ρ), which quantifies the linear relationship between the ground truth and the predicted values.

Analyzing the regression plots for the RAN and OvS VNF models (Fig. 3c–3f), we can observe that Dev-Pkt produces a higher number of inaccurate predictions compared to vNetRunner. For both of the VNFs, we can see that Dev-Pkt tends to under-predict the throughput. However, the correlation between the ground truth and the predictions for Dev-Pkt is higher for delay models, indicating that while Dev-Pkt demonstrates reasonable accuracy in learning packet delay, it struggles with modeling packet drops, which impacts throughput estimation. In contrast to RAN and OvS, the regression plots for the UPF (Fig. 3g and Fig. 3h) do not follow the same trend. Since no resource allocation is performed for the UPF (cf., Section IV-D), it results in almost zero packet loss (i.e., input throughput equals output throughput). Consequently, both vNetRunner and Dev-Pkt achieve near-perfect throughput and delay estimation for the UPF. Notably, the Dev-Pkt Pearson correlation for the UPF delay (Fig. 3h) marginally exceeds that of vNetRunner, suggesting that packet-level approaches might perform better when the learning task is relatively simple. This implies that such approaches could benefit from larger datasets and more complex models when dealing with more challenging VNF models.

Wasserstein Distance. To quantify the distributional performance, we compute the Wasserstein distance (W-distance) between the predicted and ground truth distributions, as shown in Fig. 4. Given the wide range of delay values, from milliseconds to tens of seconds, we calculate W-distances for specific delay and throughput thresholds, i.e., we limit the ground truth delay and throughput under specific values and compute the W-distance for the corresponding predictions.

For the RAN delay (Fig. 4b), both vNetRunner and Dev-Pkt perform quite well. For delay values under 10 ms (10^{-2} s), vNetRunner achieves a W-distance of approximately 0.0385 ms, while Dev-Pkt has a slightly higher W-distance of 0.112 ms. Although vNetRunner performs better than Dev-

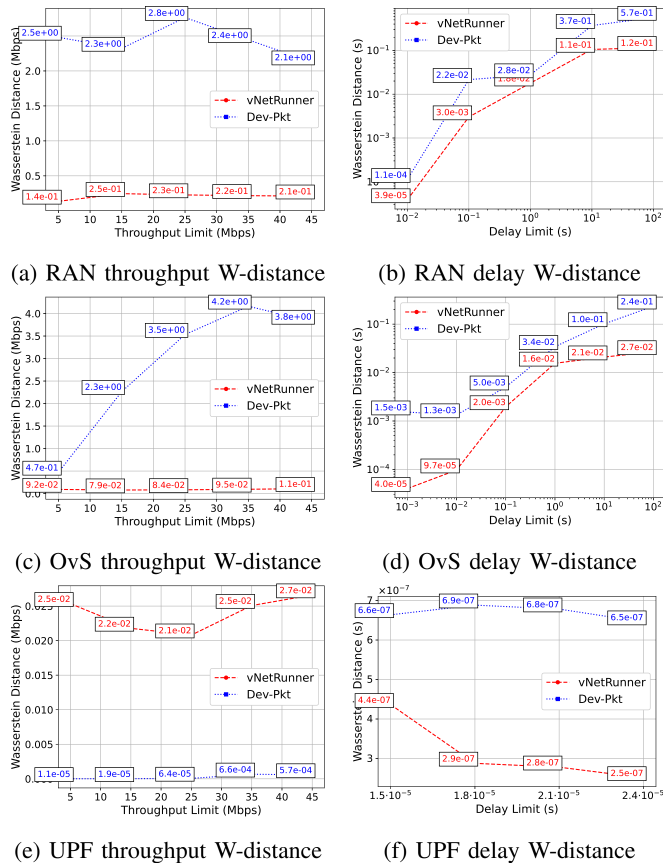
This paper has been accepted for publication in the IEEE/IFIP Network Operations and Management Symposium, Honolulu, HI, USA, May 2025. This is the author's copy. Copyright resides with IEEE/IFIP.



(a) vNetRunner: RAN model loss (b) Dev-Pkt: RAN model loss (c) RAN throughput reg. plot (d) RAN delay reg. plot

(e) OvS throughput reg. plot (f) OvS delay reg. plot (g) UPF throughput reg. plot (h) UPF delay reg. plot

Fig. 3: Loss and Regression plots for VNF models' throughput and delay



(a) RAN throughput W-distance (b) RAN delay W-distance

(c) OvS throughput W-distance (d) OvS delay W-distance

(e) UPF throughput W-distance (f) UPF delay W-distance

Fig. 4: Wasserstein distance for VNF models

Pkt, both approaches achieve a small W-distance relative to the ground truth. This trend is maintained across larger delay thresholds, such as 10 s where vNetRunner and Dev-Pkt show a W-distance of 0.105 s and 0.371 s, respectively.

For the OvS delay metrics (Fig. 4d), vNetRunner also consistently achieves lower W-distances, while Dev-Pkt struggles at lower delay thresholds. Specifically, for a delay threshold of 1 ms, vNetRunner records a W-distance of 0.0402 ms, while Dev-Pkt's W-distance is 1.53 ms. This means that vNetRunner's W-distance is approximately 38 times smaller than that of Dev-Pkt, showcasing its significant edge in this context. Even for higher thresholds like 100 s, vNetRunner maintains a W-distance of 0.0268 s, compared to 0.235 s for Dev-Pkt, representing an 88.6% reduction. However, at higher thresholds, the relative scale of the error (W-distance) is quite small compared to the ground truth for both approaches.

When considering throughput metrics, the performance gap between the two approaches consistently remains high across most throughput limits. For example, in Fig. 4a, for RAN throughput under the threshold of 5 Mbps, vNetRunner achieves a W-distance of 0.135 Mbps, while for Dev-Pkt, it reaches 2.48 Mbps. This suggests that vNetRunner reduces the deviation by approximately 94.5% compared to Dev-Pkt. Similar trends are observed for OvS throughput thresholds, where at 35 Mbps, vNetRunner's W-distance is 0.0954 Mbps versus Dev-Pkt's 4.16 Mbps, representing a significant improvement of roughly 97.7%.

For the UPF, it is noteworthy that the ground truth delay values are significantly lower compared to RAN and OvS. Nevertheless, from Fig. 4f and Fig. 4e we can see that both vNetRunner and Dev-Pkt demonstrate good performance. At a delay threshold of 1.48e-05 s, vNetRunner achieves a W-distance of 4.36e-07 s, compared to 6.64e-07 s for Dev-Pkt. On the other hand, for UPF throughput, vNetRunner performs

This paper has been accepted for publication in the IEEE/IFIP Network Operations and Management Symposium, Honolulu, HI, USA, May 2025. This is the author's copy. Copyright resides with IEEE/IFIP.

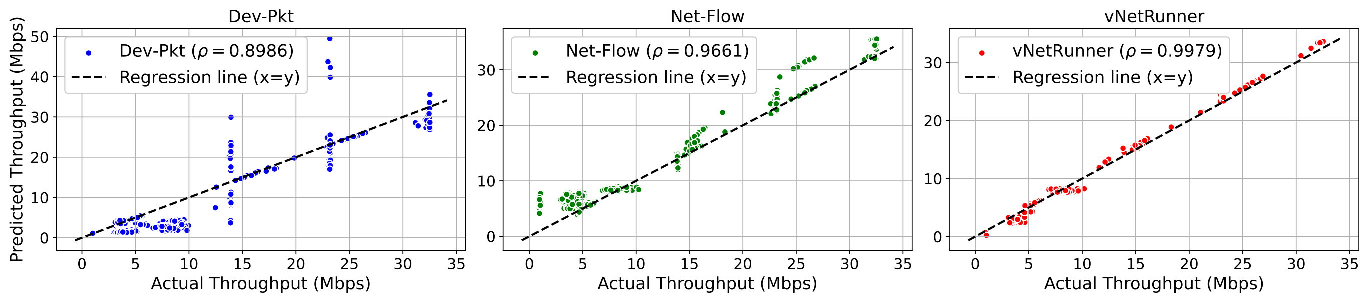


Fig. 5: Slice model throughput regression plots

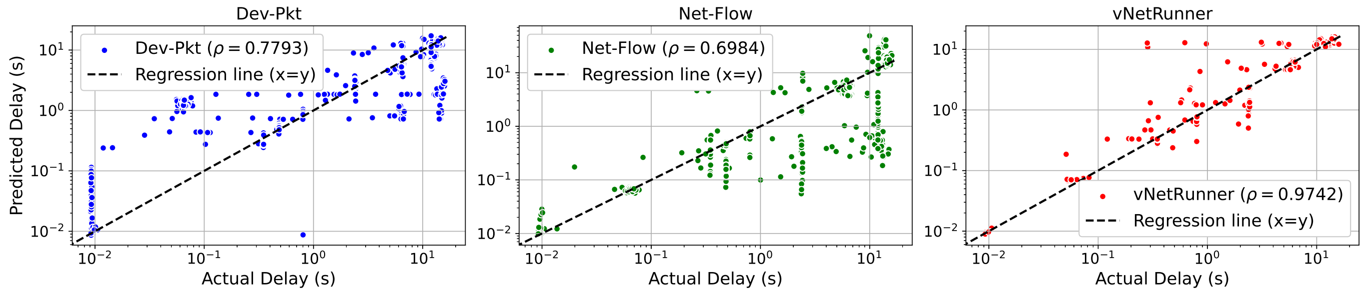


Fig. 6: Slice model delay regression plots

worse (i.e., higher W-distance) compared to Dev-Pkt. For example, for a throughput threshold of 5 Mbps, vNetRunner records 0.0253 Mbps, while Dev-Pkt achieves 1.13e-05 Mbps, highlighting a significant gap. However, these errors are negligible compared to the ground truth value. For UPF delay, vNetRunner exhibits slightly lower Pearson correlation (Fig. 3h), but also a lower W-distance (Fig. 4f) compared to Dev-Pkt. This suggests vNetRunner has more variability in capturing the linear trend, but its overall distribution of predictions aligns more closely with the true delay values.

D. Performance Evaluation of End-to-End Slice Model

Model Training. For slice modeling, we composed the individually trained VNFs for device-level approaches to form a complete slice-level (or network-level) model. Additionally, we trained a single network-level model (Net-Flow), as discussed in Section V-A. Fig. 7 shows the training and validation loss for the Net-Flow model, where, similar to the VNF models, we observe a steady decrease in both losses as training progresses, converging after 30K epochs.

Regression Plots. We validated the slice models by plotting their regression plots, shown in Fig. 5 and Fig. 6. Comparing these to the VNF regression plots, we notice a higher occurrence of outliers across all approaches, indicated by the scattered points deviating from the $x = y$ line. For device-level approaches (Dev-Pkt, vNetRunner), this behavior is attributed to error compounding as the models are connected sequentially. Additionally, the delay regression plots exhibit more inaccurate predictions compared to the throughput plots. For the Net-Flow model, the outliers are a result of the model's

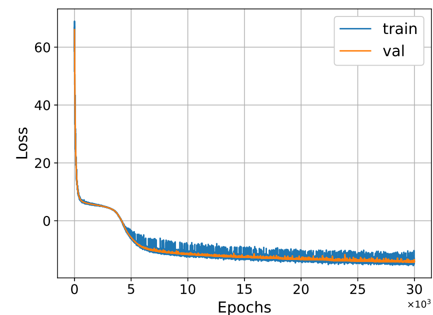


Fig. 7: Net-Flow model loss (negative log probability Eq. (2))

requirement for a significantly larger dataset to generalize effectively to unseen inputs.

In the throughput regression plots (Fig. 5), we can see that vNetRunner delivers the most accurate predictions, followed by Net-Flow and Dev-Pkt. This is further supported by the Pearson correlation (ρ) between the predicted and actual distributions. For the delay plots, despite the higher frequency of inaccurate predictions, vNetRunner maintains a strong relationship between the input and output, reflected in its high correlation value. In contrast, both Dev-Flow and Net-Pkt exhibit a larger number of inaccurate predictions and a notable drop in correlation values.

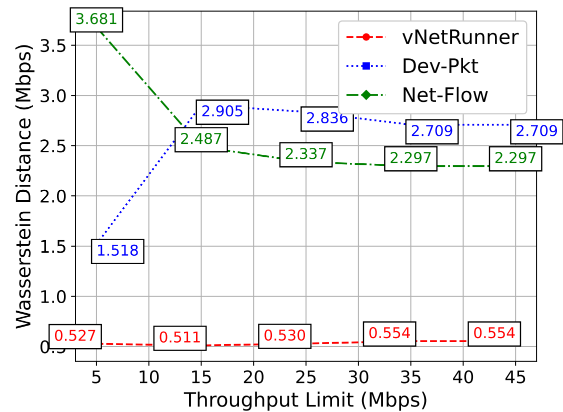
Wasserstein Distance. To numerically quantify the performance of the different slice modeling approaches, we plot their Wasserstein distance (W-distance) results, shown in Fig. 8, using the same methodology discussed in the

This paper has been accepted for publication in the IEEE/IFIP Network Operations and Management Symposium, Honolulu, HI, USA, May 2025. This is the author's copy. Copyright resides with IEEE/IFIP.

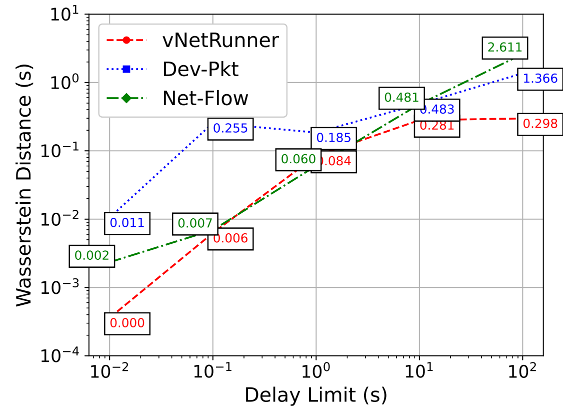
previous subsection. As expected, vNetRunner maintains the lowest W-distance for both throughput and delay metrics. In terms of throughput (Fig. 8a) vNetRunner maintains a W-distance of less than 0.554 Mbps across the entire range of throughput thresholds. Specifically, at 45 Mbps, vNetRunner achieves 0.554 Mbps, compared to 2.709 Mbps for Dev-Pkt and 2.297 Mbps for Net-Flow. This represents an improvement of approximately 79.5% and 75.9%, respectively, over Dev-Pkt and Net-Flow.

For delay metrics, vNetRunner also outperforms the other approaches significantly, especially at lower thresholds. From Fig. 8b, we can see that when delay is limited to 10 ms (10^{-2} s), vNetRunner achieves a W-distance of 0.0827 ms, while Net-Flow and Dev-Pkt record 2.34 ms and 10.58 ms. This means that vNetRunner's W-distance is 96.5% lower than Dev-Pkt and 96.5% lower than Net-Flow. These differences are especially critical for usecases that require ultra-low latency *e.g.*, URLLC (Ultra-Reliable Low Latency Communication) slices. At higher delay limits, the predictions of all models become relatively more accurate. For example, when the delay is limited to 1 second, vNetRunner records a W-distance of 80.88 ms, Net-Flow achieves 60.16 ms, and Dev-Pkt records 175.85 ms. In this scenario, vNetRunner still maintains an edge, but Net-Flow also provides relatively competitive performance compared to Dev-Pkt, whose W-distance remains 54.0% higher than vNetRunner.

Overall, from Fig. 8, we can see that vNetRunner demonstrates consistently lower W-distances compared to both Dev-Pkt and Net-Flow across delay and throughput metrics. This highlights vNetRunner's effectiveness in accurately modeling the slice behavior, particularly under stringent thresholds.



(a) Slice model throughput W-distance



(b) Slice model delay W-distance

Fig. 8: Slice models' Wasserstein distance plots

Execution Time. As discussed in Section I, most AI-based network management approaches can require tens of thousands interactions with the network model for convergence. This makes the speed up the different approaches provide over the real network critical. Therefore, in this subsection, we compare the execution time of vNetRunner with Net-Flow and Dev-Pkt. For this, we tested each method over a real-world scenario lasting 100 minutes. For device-level approaches, we ensured that the upstream VNF model process the entire input stream before passing it to the downstream VNF. In this setup, Net-Flow, vNetRunner, and Dev-Pkt achieved execution times of 0.774s, 5.90s, and 340s, respectively. These results translate to speedups of more than 7500-fold, 1000-fold, and 17-fold. The long execution time of Dev-Pkt is due to several time-intensive operations on the entire packet stream, including adding delays to packets, removing dropped packets, sorting the output stream, and performing feature extraction using a sliding window. This indicates that Dev-Pkt's execution time scales poorly as the data rate (i.e., the number of packets) increases. While parallelization can significantly reduce execution time for packet-level approaches, it can still range from several seconds to minutes [13].

VI. Conclusion

This paper presents vNetRunner, a novel per-VNF slice modeling framework tailored for 5G and beyond networks. By focusing on flow-level features for individual VNFs, vNetRunner significantly reduces the computational complexity and improves scalability compared to packet-level models, and reduces the dataset requirement compared to device-level models. Additionally, unlike network-level models, vNetRunner enables model composability, allowing seamless replacement, removal, and addition of VNFs within a slice. Our evaluation demonstrates that vNetRunner not only achieves high accuracy in estimating key performance metrics like delay and throughput but also offers significant speedup in execution time, making it well-suited for real-time applications. Future work could explore modeling VNFs from different open-source projects, hybrid packet-level and flow-level models, and using vNetRunner for network 5G slice management tasks such as slice admission control and resource allocation.

Acknowledgement

This work was supported in part by Rogers Communications Canada Inc. and in part by a Mitacs Accelerate Grant.

This paper has been accepted for publication in the IEEE/IFIP Network Operations and Management Symposium, Honolulu, HI, USA, May 2025. This is the author's copy. Copyright resides with IEEE/IFIP.

References

[1] N. Saha, N. Shahriar *et al.*, "MonArch: network slice monitoring architecture for cloud native 5G deployments," in *Proceedings of IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2023, pp. 1–7.

[2] M. Sulaiman, A. Moayyedi *et al.*, "Coordinated slicing and admission control using multi-agent deep reinforcement learning," *IEEE Transactions on Network and Service Management*, 2022.

[3] M. Sulaiman, M. Ahmadi *et al.*, "Microopt: Model-driven slice resource optimization in 5g and beyond networks," 2024. [Online]. Available: https://arxiv.org/abs/2407.18342

[4] S. S. Johari, N. Shahriar *et al.*, "Anomaly detection and localization in nfv systems: an unsupervised learning approach," in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, 2022, pp. 1–9.

[5] G. F. Riley and T. R. Henderson, *The ns-3 Network Simulator*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. [Online]. Available: https://doi.org/10.1007/978-3-642-12331-3_2

[6] A. Varga, *A Practical Introduction to the OMNeT++ Simulation Framework*. Cham: Springer International Publishing, 2019, pp. 3–51. [Online]. Available: https://doi.org/10.1007/978-3-030-12842-5_1

[7] Z. Lu and H. Yang, *Unlocking the Power of OPNET Modeler*. Cambridge University Press, 2012.

[8] Q. Zhang, K. K. W. Ng *et al.*, "Mimicnet: fast performance estimates for data center networks with machine learning," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, ser. SIGCOMM '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 287–304. [Online]. Available: https://doi.org/10.1145/3452296.3472926

[9] "5G RAN &x2013; OpenAirInterface — openairinterface.org," https://openairinterface.org/oai-5g-ran-project/, [Accessed 19-10-2024].

[10] "5g - srsRAN Project." [Online]. Available: https://www.srsran.com/

[11] "GitHub - free5gc/free5gc: Open source 5G core network based on 3GPP R15 — github.com," https://github.com/free5gc/free5gc, [Accessed 19-10-2024].

[12] S. Lee, "Documentation — open5gs.org," https://open5gs.org/open5gs/docs/, [Accessed 19-10-2024].

[13] Q. Yang, X. Peng *et al.*, "DeepQueueNet: towards scalable and generalized network performance estimation with packet-level visibility," in *Proceedings of the ACM SIGCOMM*. ACM, 2022, pp. 441–457.

[14] K. Rusek, J. Suarez-Varela *et al.*, "Routenet: Leveraging graph neural networks for network modeling and optimization in sdn," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, p. 2260–2270, Oct. 2020. [Online]. Available: http://dx.doi.org/10.1109/JSAC.2020.3000405

[15] M. Ferriol-Galmés, J. Paillisse *et al.*, "RouteNet-Fermi: Network modeling with graph neural networks," *IEEE/ACM Transactions on Networking*, 2023.

[16] Q. Liu, N. Choi, and T. Han, "Atlas: automate online service configuration in network slicing," in *Proceedings of International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. ACM, 2022, pp. 140–155.

[17] M. Sulaiman, M. Ahmadi *et al.*, "Generalizable resource scaling of 5G slices using constrained reinforcement learning," in *Proceedings of IEEE/IFIP Network Operations and Management Symposium (NOMS)*. IEEE, 2023, pp. 1–9.

[18] D. Bega, M. Gramaglia *et al.*, "DeepCog: Cognitive network management in sliced 5G networks with deep learning," in *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2019, pp. 280–288.

[19] Q. Liu, N. Choi, and T. Han, "OnSlicing: online end-to-end network slicing with reinforcement learning," in *Proceedings of ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. ACM, 2021, pp. 141–153.

[20] O. Iacoboaiea, J. Krolikowski *et al.*, "From design to deployment of zero-touch deep reinforcement learning WLANs," *IEEE Communications Magazine*, vol. 61, no. 2, pp. 104–109, 2023.

[21] Y. Zhang, M. Zhao, and Q. Liu, "Learn to augment network simulators towards digital network twins," 2023. [Online]. Available: https://arxiv.org/abs/2311.12745

[22] C. Bishop, "Mixture density networks," Aston University, Tech. Rep., 1994.

[23] A. Paszke, S. Gross *et al.*, "Automatic differentiation in pytorch," 2017.

[24] A. Graves, "Stochastic backpropagation through mixture density distributions," *arXiv preprint arXiv:1607.05690*, 2016.

[25] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick," 2015. [Online]. Available: https://arxiv.org/abs/1506.02557

[26] "Introduction — srsRAN 4G 23.11 documentation." [Online]. Available: https://docs.srsran.com/projects/4g/en/latest/usermanuals/source/srsue/source/1_ue_intro.html

[27] The Linux Foundation, "OpenVSwitch," https://www.openvswitch.org/, 2023, version 2.9.8.

[28] The Open Networking Foundation, "ONOS," https://github.com/opennetworkinglab/onos, 2023, version 2.5.7-rc1.

[29] Canonical, "Ubuntu Manpage: ntpdate - set the date and time via NTP — manpages.ubuntu.com," https://manpages.ubuntu.com/manpages/bionic/man8/ntpdate.8.html, [Accessed 15-10-2024].