MicroOpt: Model-Driven Slice Resource Optimization in 5G and Beyond Networks

Muhammad Sulaiman, *Graduate Student Member, IEEE*, Mahdieh Ahmadi, Bo Sun, Mohammad A. Salahuddin, *Member, IEEE*, Raouf Boutaba, *Fellow, IEEE*, and Aladdin Saleh

***Abstract*—A pivotal attribute of 5G networks is their capability to cater to diverse application requirements. This is achieved by creating logically isolated virtual networks, or slices, with distinct service level agreements (SLAs) tailored to specific use cases. However, efficiently allocating resources to maintain slice SLA is challenging due to varying traffic and quality-of-service (QoS) requirements. Traditional peak traffic-based resource allocation leads to over-provisioning, as actual traffic rarely peaks. Additionally, the complex relationship between resource allocation and QoS in end-to-end slices spanning different network segments makes conventional optimization techniques impractical. Existing approaches in this domain use mathematical network models (e.g., queueing models) or simulations, and various optimization methods but struggle with optimality, tractability, and generalizability across different slice types. In this paper, we propose MicroOpt, a novel framework that leverages a differentiable neural network-based slice model with gradient descent for resource optimization and Lagrangian decomposition for QoS constraint satisfaction. We evaluate MicroOpt against two state-of-the-art approaches using an open-source 5G testbed with real-world traffic traces. Our results demonstrate up to 21.9% improvement in resource allocation compared to these approaches across various scenarios, including different QoS thresholds and dynamic slice traffic.**

***Index Terms*—5G, network slicing, dynamic resource scaling, machine learning, quality of service.**

I. Introduction

NETWORK slicing (*cf.*, Section II-A) empowers 5G networks to accommodate applications and services with diverse Quality of Service (QoS) requirements [1]. However, network slicing also brings forth the challenge of effectively managing resources in a complex and dynamic environment. Each slice is associated with a service level agreement (SLA) specifying the peak traffic and minimum QoS requirements of slice users (*i.e.*, slice tenants). To ensure QoS, the InP can allocate isolated resources to each slice based on its peak traffic. However, this approach often leads to over-provisioning as the actual slice traffic may exhibit fluctuations over time and rarely reach its peak [2], resulting in the under-utilization of resources. Moreover, SLAs can be dynamic and subject to change based on various factors, including the number of users, slice location, and time of day. For example, a smart healthcare application requires a URLLC slice with full isolation and extremely low latency during surgery [3], [4] which can be adjusted post-surgery to align with updated service expectations. This highlights the need for a more adaptable approach to resource management that is capable of accommodating changes in traffic patterns and SLA requirements. To achieve this, the InP needs to maintain QoS degradation under a specific threshold by predicting future traffic patterns and dynamically allocating resources. This problem is known as predictive resource allocation or dynamic resource scaling (DRS) of network slices.

To achieve efficient resource utilization while meeting SLA commitments, it is crucial to carefully examine the relationship between resource allocations and QoS metrics. However, QoS metrics for an end-to-end slice, such as throughput, latency, and reliability, rely on various resource types across multiple network segments, including the Radio Access Network (RAN), transport network, and core network (*cf.*, Section II-A). Traditional network models such as queues, often fail to capture these complexities accurately, especially considering the highly dynamic and mobile nature of 5G networks [5]. This adds to the complexity of DRS.

Existing approaches for resource scaling typically involve traffic forecasting [6] for a given slice, followed by utilizing simulations [5], [7] or Machine Learning (ML) [8] to learn the network model and determine the optimal resource allocation. Simulation-based methods, such as using packet-level simulators (*e.g.*, ns-3) [7] or queue-based simulators [5], are computationally expensive [9], and may not accurately represent the network. Additionally, some of these works assume the slice traffic to be constant [7].

ML-based approaches model the entire network as a single entity using a neural network and have shown promising performance [9]. However, existing works that follow this approach [8], [10] for dynamic resource scaling suffer from several issues. Reference [8] adopts a simple regression-based model, which fails to capture the complexities of an end-to-end network. Furthermore, due to the lack of a differentiable

Received 15 August 2024; revised 20 February 2025 and 21 May 2025; accepted 17 June 2025. Date of publication 30 June 2025; date of current version 7 October 2025. This work was supported in part by Rogers Communications Canada Inc. and in part by a Mitacs Accelerate Grant. The associate editor coordinating the review of this article and approving it for publication was D. Pezaros. *(Corresponding author: Mohammad Salahuddin.)*

Muhammad Sulaiman, Bo Sun, Mohammad A. Salahuddin, and Raouf Boutaba are with the David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada (e-mail: m4sulaim@uwaterloo.ca; b24sun@uwaterloo.ca; mohammad.salahuddin@uwaterloo.ca; rboutaba@uwaterloo.ca).

Mahdieh Ahmadi is with Ericsson R&D, Ottawa, ON L4W 5K4, Canada (e-mail: mahdieh.ahmadi@ericsson.com).

Aladdin Saleh is with the Technology Partnerships and Innovations, Rogers Communications Canada Inc., Toronto, ON M4W 1G9, Canada (e-mail: aladdin.saleh@rci.rogers.com).

Digital Object Identifier 10.1109/TNSM.2025.3584257

network model in these works, they cannot take advantage of efficient gradient-based optimization techniques. Instead, they are constrained to employ an inefficient optimization method such as constrained Reinforcement Learning (RL), or Bayesian Optimization which yield sub-optimal results. On the other hand, complete model-free approaches that aim to learn resource allocation directly in an online fashion can impact SLAs and require longer training times [11].

In this paper, we introduce MicroOpt, a novel framework that combines the power of ML with continuous optimization for dynamic resource scaling of network slices. Our proposed approach leverages a neural network to estimate QoS metrics and incorporates optimization techniques for efficiently scaling slice resources. By employing the reparameterization trick [12], which is a commonly used technique in probabilistic Deep Learning (DL) models, we can iteratively refine slice resource allocation to minimize resource usage while meeting SLA requirements. The reparameterization trick ensures the differentiability of the slice model, enabling continuous optimization through gradient descent. The major contributions of this work are as follows:

- We introduce MicroOpt, a novel framework for dynamic resource scaling of 5G slices that can accommodate time-varying traffic, dynamic QoS and QoS degradation thresholds, without the need for retraining. In MicroOpt, we first propose a deep neural network (DNN)-based model of 5G slices, utilizing the reparameterization trick to compute the gradients of QoS degradation constraints with respect to resource allocations. With these gradients, we further design a primal-dual optimization algorithm that leverages the differentiability of the slice model to enable efficient resource allocation optimization using gradient descent. This algorithm employs an inner loop that leverages a relaxed differentiable Lagrangian function for optimizing resource allocation, whereas, in the outer loop, strict definitions are utilized to enforce QoS degradation constraints.
- We compare MicroOpt against two state-of-the-art baselines and their variants: **Atlas** [7] and **Altas**$^{+}$ that are based on Bayesian Optimization (BO); and **CaDRL** [13] and **CaDRL**$^{+}$ that are constrained RL algorithms. By extensive simulations using real-world data traces, we demonstrate that MicroOpt can significantly reduce the mean resource allocation (up to 21.9%) while providing QoS guarantees through dynamic resource scaling. Furthermore, we evaluate the convergence properties of the different approaches, and perform an ablation study to demonstrate the necessity of modeling QoS as a distribution rather than scalar values to satisfy QoS requirements.
- We validate the MicroOpt's performance using a 5G testbed, with real-world traffic traces for multiple QoS, and QoS degradation thresholds. We also evaluate MicroOpt's generalization to different QoS and traffic types. Additionally, we make the generated datasets publicly available.

The structure of the paper is as follows: We start by briefly introducing network slicing and its enablers in Section II-A. In Section II, we provide a comprehensive review of related works encompassing both network modeling and resource allocation. Section III formally defines the problem, while Section IV presents our proposed solution. Section V details our testbed and evaluation setup, and Section VI presents the evaluation results. We conclude in Section VII and investigate future research directions.

II. Background and Related Works

A. Network Slicing

Network slicing refers to the creation of isolated virtual networks over a shared physical infrastructure, tailored to the specific needs of different services [1]. For example, enhanced mobile broadband (eMBB) slices cater to high-throughput applications like 4K video streaming, while ultra-reliable low-latency communication (URLLC) slices support latency-sensitive applications such as remote surgery [14], [15]. Key enablers of network slicing include Network Function Virtualization (NFV) and Software-defined Networking (SDN) [16], which decouple network functions from hardware and enable flexible, on-demand service provisioning.

In a 5G network, slicing spans each network segment *i.e.*, the Radio Access Network (RAN), Transport Network (TN), and Core Network (CN). RAN slicing utilizes slice-aware resource scheduling and dynamic functional splits between Centralized (CU) and Distributed Units (DU) to meet latency and throughput requirements. Core network slicing [16], as standardized by 3GPP, introduces the network functions such as the Network Slice Selection Function (NSSF) to allocate UEs to appropriate slices based on their service requirements. Transport network slicing [17], as defined by the IETF, involves fine-grained bandwidth allocation at the edge, and coarse-grained resource control, using mechanisms such as DiffServ, MPLS, and Segment Routing within the transit nodes to ensure QoS compliance across end-to-end network slices.

B. Dynamic Resource Allocation

Predictive slice resource allocation is the process of allocating resources in anticipation of future user demand. This approach can assist in maintaining SLAs with minimum overprovisioning, even when demand spikes. The quintessential components of predictive resource allocation include: (i) Traffic prediction for proactive resource management, (ii) A slice model for estimating QoS based on allocated resources, and (iii) An optimization approach that integrates traffic prediction and the slice model to determine the optimal resource allocation.

A naive approach in any of these can lead to a sub-optimal solution, and in the following sub-sections we discuss the current literature in the context of these modules. It should be noted that these modules may not be separately identifiable in all related works. For example, numerous works in the literature have assumed that resource demand can be readily derived based on traffic [18], [19], which precludes the necessity of II. Similarly, works that use RL for end-to-end resource allocation implicitly assume that RL agent learns

to predict the traffic pattern in addition to the corresponding resource requirements [5], [20].

1) Traffic Prediction: Network resource management approaches can be categorized based on the timing of reconfiguration relative to demand fluctuations [21]. Reactive approaches adjust resources only after a mismatch between demand and provisioning occurs, responding to instantaneous traffic conditions. In contrast, proactive approaches anticipate future demand using predictive models, enabling preemptive reconfiguration. Proactive approaches are better able to meet SLA requirements, but require accurate traffic prediction in advance.

Traditional model-driven approaches [22], [23] assume that traffic demand follows a predefined distribution. However, without explicit prediction, these methods rely on overly conservative estimates, leading to inefficient resource allocation to prevent QoS degradation. Conversely, approaches that predict future demand based on currently observed data often focus on single-point predictions rather than modeling a full distribution. For example, [24] employs recurrent neural networks (RNNs) to predict future channel states (e.g., SNR). However, relying solely on point predictions can lead to making resource allocation decisions that are less robust to demand fluctuations. To address this, uncertainty-aware forecasting methods can be leveraged. For example, in [21] the authors propose a Prediction Interval-based Predictor (PIP), which utilizes Gated Recurrent Units (GRU) and a bootstrap method to generate confidence intervals for future traffic demand.

In this work, we assume traffic prediction is represented as a distribution rather than a single-point estimate. However, we do not propose a specific prediction method, as it falls outside the scope of the proposed algorithm. Instead, we assume that traffic predictions are available through existing approaches, such as those in [21], [25].

2) Slice Modeling: The effectiveness of resource allocation algorithms is highly dependent on slice models, which correlate allocated resources with QoS distribution based on predicted demand and slice configuration. However, such models may not exist for end-to-end slices under various traffic distributions or network configurations. Therefore, network simulators [5], [7] and ML-based estimators [8] are commonly employed to address this challenge.

Conventional network simulators, such as ns-3, are packet-level and time-intensive [9], [26], which limits their application for online resource allocation and even for offline training of RL policies [27]. Moreover, simulators may not accurately mimic real-world scenarios, especially in wireless domains [7]. Bayesian optimization (BO) has been utilized by Liu et al. [7] to identify the optimal ns-3 parameters and reduce the disparity between simulated and real-world conditions. However, this process needs to be repeated for each minor alteration in the network and SLA metrics.

ML-driven approaches model the entire network as a neural network, which can be trained to estimate end-to-end QoS metrics using traffic traces. The constraints of this approach include limited visibility at the packet-level and lack of generalizability across diverse network settings. Recently, graph neural networks [9] and a combination of simulation and DNN models [26] have been used to alleviate these concerns in the context of transport networks. However, the transport network corresponds to only a single portion of an end-to-end slice, and usually handles aggregated traffic. Our previous work [8] and [10], have proposed the use of ML-based models for network modeling. However, since the QoS degradation derivation required non-deterministic, and non-differentiable operations on the network model's output, these works can not leverage gradient backpropagation through the network model for optimization.

3) Resource Allocation Algorithm:

Machine Learning. ML-based optimization approaches have been successfully applied in various resource allocation [28] and scheduling problems [29]. In the context of predictive resource allocation, Bega et al. [30] use an encoder-decoder model to predict the amount of resources needed to minimize the aggregated cost of resource over-provisioning and SLA violations due to customer churn. However, the aggregate cost function does not guarantee SLAs.

Recently, a number of works have used constrained deep reinforcement learning (CDRL) techniques to learn the optimal resource allocation under average SLA constraints [5], [8], [11]. These approaches require retraining for each minor change in SLA and may not generalize to real traffic patterns that are unseen during training. Liu et al. [5] utilize queue-based simulated environment to train the resource allocation policy offline. This approach may encounter issues stemming from the discrepancy between the simulated environment and the real-world network. Additionally, Liu et al. [11] adopt a fully model-free approach by training the policy online. However, this method seems impractical for implementation in a production network due to long convergence time.

Sulaiman et al. [8] address some of these drawbacks by using a simple regression-based model for QoS estimation and a risk-aware CaDRL agent trained offline over randomized traffic to satisfy the SLA constraint. However, since RL algorithms do not leverage the slice model's gradient for policy optimization, they have to rely on inefficient random exploration. In soft actor-critic algorithms adopted in this work, this is achieved by learning a critic network to estimate the state-action value of different actions and a policy network that maximizes the expected cumulative reward. However, as shown in [8], even though the offline learned policy generalizes to different online scenarios, it achieves sub-optimal results compared to approaches trained with advanced knowledge of the online scenarios.

Optimization. Many works in the literature assume that the amount of required resources can be readily determined based on the SLA requirement, and utilize optimization techniques to improve performance in scenarios with multiple slices and resource contention [18], [19]. Although this may hold for Physical Resource Block (PRB) allocation in a single base station, it cannot be generalized to an end-to-end slice that requires the allocation of various resource types across different network segments, such as the bandwidth on the transport network, and compute resources for running virtualized network functions and edge applications.

Reference [24] employs MAC scheduler-level simulation and proposes a binary search algorithm that exploits the monotonic relationship between QoS and PRB allocation in the RAN. However, this approach does not generalize to end-to-end networks, where resources are interdependent *i.e.*, the optimal allocation of one resource depends on the allocation of other resources. In contrast, [21] models traffic demand uncertainty and bandwidth fluctuations using box and ellipsoid uncertainty sets, formulating a Robust MILP that can be efficiently solved using commercial optimization solvers.

Liu et al. [7] employ Bayesian optimization (BO) to minimize the resource consumption of a single slice while satisfying its SLA. The authors relax the constraint using the Lagrangian method and approximate QoS using a Bayesian neural network. However, the Bayesian neural network approximates the probability of violating the QoS rather than approximating the QoS itself. Moreover, for each slice and traffic value, QoS is approximated for a limited set of resource allocations that have a higher chance of being the solution. Therefore, if there are changes in traffic or SLA, the neural network needs to query new points in the domain. Additionally, the model assumes a constant level of traffic throughout the entire configuration interval (*i.e.*, 1-2 hours), which is not the case in practice. This assumption compels to consider the worst-case traffic value within each interval to satisfy the SLA, which leads to a sub-optimal solution. Moreover, even with an assumption of constant traffic, the convergence time of this method is in the order of hours, which can span a significant portion of each reconfiguration interval.

In this work, we address two main drawbacks identified in the current literature. First, large QoS querying times—network simulators, such as ns-3 [7], and queue-based network models [5], [11], are computationally intensive and require significant time to compute the QoS for a given resource allocation. Second, ineffective exploration—optimization methods like RL, which use ϵ-greedy algorithms for solution space exploration, require numerous interactions with the network and often leading to sub-optimal solutions. While modern network virtualization solutions, such as Kubernetes and ONOS, enable rapid reconfiguration of resource allocation, these current approaches fail to fully leverage this capability due to their inherent inefficiencies.

We address the first challenge by employing a DNN-based slice model to predict QoS, enabling fast queries on the order of milliseconds once trained. As shown in [26], ML-based network modeling only require several minutes to estimate the network behavior, whereas a discrete-event simulator can take tens of hours. To tackle the second challenge, we utilize the reparameterization trick and gradient-based optimization, which leverages gradient information to adjust the search direction. This approach efficiently explores the parameter space, resulting in a better solution, and faster convergence.

III. Problem Statement

The process of establishing an end-to-end network slice involves the allocation of resources across various network segments. Reconfiguration of these resources occurs at specific time intervals referred to as reconfiguration intervals. We use i to denote the i^{th} reconfiguration interval. The duration of these intervals, represented by τ_i, depends on various factors, including data collection delays [11], constraints imposed by the substrate network (*e.g.*, the use of legacy virtual infrastructure manager) [31], or the time necessary for accurate traffic forecasting [25], to name a few. Typically, the duration of a reconfiguration interval can range from several minutes [11] to hours [7]. Additionally, τ_i can be dynamically adjusted to account for unexpected situations in the real world. We denote the set of operational network slices as S, where each slice $s \in S$ is associated with a resource allocation vector $\boldsymbol{r}_i^s = [r_i^{s,1}, \ldots, r_i^{s,K}] \in \mathbb{R}^K_{\geq 0}$, comprising K resources.

The slice traffic, *i.e.*, the number of users connected to a slice, may exhibit variations during reconfiguration intervals [2]. To represent the traffic time series within interval i, we employ the notation $\boldsymbol{x}_i^s = [x_i^s(1), \ldots, x_i^s(\tau_i)] \in \mathbb{R}^{\tau_i}_{\geq 0}$. The traffic is measured in users/s and the average QoS experienced by these users is determined by the resources allocated to the slice. We represent the average QoS for slice s in interval i as $\boldsymbol{q}_i^s = [q_i^s(1), \ldots, q_i^s(\tau_i)] \in \mathbb{R}^{\tau_i}_{\geq 0}$. For each time slot, if the QoS fails to surpass the predetermined QoS threshold q^s_{thresh}, it results in QoS degradation. Under the assumption of fair resource allocation and equal average QoS experienced by each user, the average QoS degradation for slice s during the reconfiguration interval i is defined as:

$$\beta_i^s = \frac{\sum_{t=1}^{\tau_i} x_i^s(t) \cdot \mathbb{1}_{[q_i^s(t) \leq q^s_{thresh}]}}{\sum_{t=1}^{\tau_i} x_i^s(t)}. \tag{1}$$

The objective is to minimize the normalized resource allocation to network slices within each reconfiguration interval i, while respecting the QoS degradation constraints [7], [8], [25]. This can be formulated as:

$$\begin{aligned} \min_{\mathbf{r}_i^s \geq 0} \quad & \sum_{s \in \mathbf{S}} \boldsymbol{\eta}^\intercal \mathbf{r}_i^s, \\ \text{s.t.} \quad & \mathbb{E}(\beta_i^s) \leq \beta^s_{thresh}, \quad \forall s \in S \\ & \sum_{s \in S} \mathbf{r}_i^s \leq \mathbf{R}, \end{aligned} \tag{2}$$

where $\boldsymbol{\eta} \in \mathbb{R}^K_{>0}$ is the normalization vector, and can also be used to represent the priority or the relative cost associated with different resources, $\mathbb{E}$ represents the expectation operator with respect to the QoS distribution, and vector $\mathbf{R} = [R^1, \ldots, R^K] \in \mathbb{R}^K_{\geq 0}$ represents the capacities of different resources. The aforementioned problem pertains to a black-box continuous optimization problem due to the unknown distribution of QoS, $q_i^s(t)$. In the following section, we will introduce the MicroOpt framework designed to tackle this problem.

IV. MicroOpt Framework

We propose MicroOpt, shown in Fig. 1, a framework for continuous optimization of network slice resources, that solves the constrained optimization problem in (2) by addressing three key aspects: (i) predicting slice traffic, (ii) obtaining the slice model function, and (iii) solving the constrained optimization problem. We assume that slice traffic exhibits

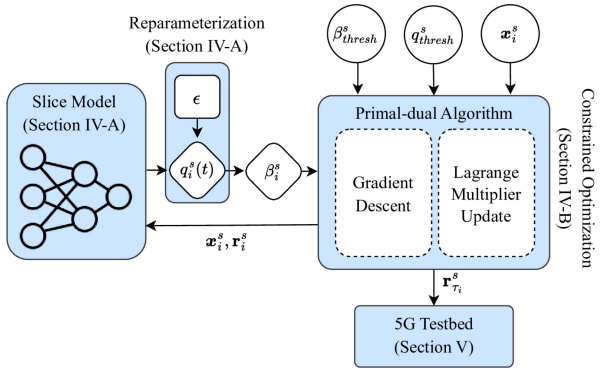


Fig. 1. MicroOpt framework overview.

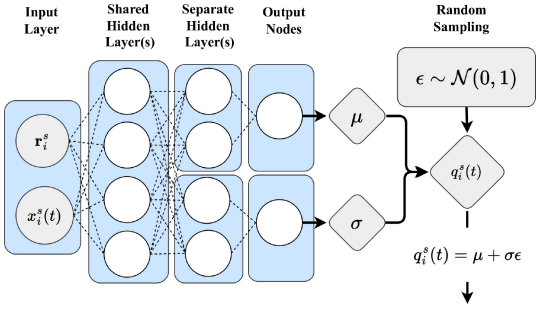


Fig. 2. Slice model.

regular and predictable patterns [2], [32] and can be predicted using existing time-series forecasting techniques (*e.g.*, [21]). To model the slice, we employ a DNN-based approach, which has been successfully employed in the network digital twin space [9], [26]. This approach leverages the expressive power of neural networks to capture the complex relationships and dependencies within the slice, and allows for analytical gradient calculation through the reparameterization trick. Finally, for tackling the constrained optimization problem, we propose a primal-dual optimization algorithm that capitalizes on the differentiability of the neural network for efficient online optimization.

A. Slice Model

Slice modeling encompasses the acquisition of the function $f^s_{QoS}(x^s_i(t), \mathbf{r}^s_i)$, that captures the relationship among resource allocation $\mathbf{r}^s_i$, slice traffic $x^s_i(t)$, and QoS distribution. The QoS sampled from this distribution, *i.e.*, $q^s_i(t) \sim f^s_{QoS}(x^s_i(t), \mathbf{r}^s_i)$, can be used to calculate the QoS degradation β^s_i using (1). Finally, the estimated β^s_i is used for solving the constrained optimization problem in (2). In this work, we propose using a DNN model to learn $f^s_{QoS}(x^s_i(t), \mathbf{r}^s_i)$ using a QoS dataset encompassing various resource allocations $(\mathbf{r}^s_i)$ and traffic $(\boldsymbol{x}^s_i)$. Unlike mathematical models or queues, a DNN-based model can easily handle heterogeneous types of resources and predict different QoS metrics. Additionally, the complexity of this approach does not depend on the traffic volume, which is the case with packet-level simulators.

We assume the QoS normally distributed for the remainder of this paper, with the network model designed to predict the parameters of the QoS distribution. We choose the normal distribution as it proves to be sufficient for effectively modeling the data in our case. However, it is important to note that the proposed slice model can be extended to incorporate mixture density networks (MDNs), which have the ability to represent arbitrarily complex distributions [33]. The architecture of the slice model is shown in Fig. 2. The inputs to the model consist of the slice traffic $x^s_i(t)$ and resource allocation $\mathbf{r}^s_i$. These inputs are connected to a set of shared hidden layers, followed by separate hidden layers dedicated to each Gaussian distribution parameter. As a result, the model outputs the Gaussian distribution parameters μ and σ associated with the predicted QoS distribution. The probability density function of $q^s_i(t)$ under this distribution can be written as:

$$p(q^s_i(t) \mid x^s_i(t), \mathbf{r}^s_i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\left(q^s_i(t)-\mu\right)^2}{2\sigma^2}\right). \quad (3)$$

Finally, the loss for the model is computed as:

$$L_{QoS} = -\frac{1}{B}\sum_{j=1}^{B} \log p\Big(q^s_{i,j}(t) | \mathbf{r}^s_{i,j}, x^s_{i,j}(t)\Big), \quad (4)$$

where B is the batch size and the subscript j represents the j^{th} sample in the batch. This loss function calculates the negative log-likelihood of the QoS $q^s_{i,j}(t)$ under the normal distribution $\mathcal{N}(\mu, \sigma)$ generated by the model for the input $(x^s_{i,j}(t), \mathbf{r}^s_{i,j})$. Once trained, this model can be used to sample the QoS from the predicted distribution. The detailed parameter settings and slice model results are given in Section VI.

The drawback of naïvely sampling QoS from this distribution is that any subsequent optimization algorithm that relies on the sampled QoS cannot leverage its gradients. This is because random sampling from a distribution involves non-deterministic operations on the model outputs, which happen to be non-differentiable. To address this, we propose using the reparameterization trick,[1] which has been employed in the ML literature [12], and can also be used with other probability distributions including MDNs [35]. For this purpose, the QoS sampling is reformulated as follows:

$$q^s_i(t) = \mu + \sigma\epsilon, \quad (5)$$

where ϵ is a random sample from a standard normal distribution $\mathcal{N}(0, 1)$ that does not depend on the input $(x^s_i(t), \mathbf{r}^s_i)$. Finally, after computing gradients through this operation and use them in subsequent optimization described in the following subsection.

B. Constrained Optimization

We start by converting the constrained problem in (2) into an unconstrained one by using dual Lagrangian relaxation *i.e.*, introducing the constraints into the objective by associating

[1]This trick allows for the use of existing automatic differentiation frameworks (e.g., PyTorch [34]) for easy implementation, and advanced optimizers (e.g., Adaptive moment optimizer) to avoid local minimas and achieve faster convergence.

a penalty (Lagrangian variables) for constraint violation. The Lagrangian is defined as follows:

$$\mathcal{L}(\mathbf{r}_i^s, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \sum_{s\in S} \boldsymbol{\eta}^\intercal \mathbf{r}_i^s + \sum_{s\in S} \lambda_s(\mathbb{E}(\beta_i^s) - \beta_{thresh}^s) + \sum_{k=1}^{K} \mu_k \left(\sum_{s\in S} r_i^{s,k} - R^k \right), \tag{6}$$

where $\boldsymbol{\lambda} = [\lambda_1, \lambda_2, \ldots, \lambda_{|S|}]$ denotes the vector of Lagrange multipliers for the QoS degradation constraints, and $\boldsymbol{\mu} = [\mu_1, \mu_2, \ldots, \mu_K]$ denotes the vector of Lagrange multipliers for the resource constraint. We can see that given a resource allocation $\mathbf{r}_i^s$, we can easily compute $\mathcal{L}(\mathbf{r}_i^s, \boldsymbol{\lambda}, \boldsymbol{\mu})$ by substituting the value of β_i^s obtained using the slice model. Based on this Lagrangian formulation, the dual problem can be written as:

$$\max_{\boldsymbol{\lambda},\boldsymbol{\mu}} g(\boldsymbol{\lambda}, \boldsymbol{\mu}) \text{ subject to } \boldsymbol{\lambda} \geq 0, \quad \boldsymbol{\mu} \geq 0, \tag{7}$$

where the dual function $g(\boldsymbol{\lambda}, \boldsymbol{\mu})$ is defined as:

$$g(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \inf_{\mathbf{r}_i^s} \mathcal{L}(\mathbf{r}_i^s, \boldsymbol{\lambda}, \boldsymbol{\mu}). \tag{8}$$

The min–max representation in (7) expresses how, for a given set of dual variables, the dual function (8) minimizes the Lagrangian by selecting an optimal resource allocation $\mathbf{r}_i^s$, while the dual variables $(\boldsymbol{\lambda}, \boldsymbol{\mu})$ themselves adjust to drive that minimum to its highest possible value by penalizing constraint violations. Assuming the objective in (2) is convex, the optimal solution to (7) will minimize resource allocation while satisfying all constraints [36]. The above dual problem could be solved iteratively using primal-dual updates with gradient-based methods [36], if it were differentiable with respect to both primal and dual variables. This is because gradient-based methods rely on the ability to compute the gradients of the objective function and the constraints with respect to relevant variables. However, the computation of the QoS degradation β_i^s using (1) involves an indicator function $\mathbb{1}_{[q_i^s(t) \leq q_{thresh}^s]}$, which is piece-wise constant and has a gradient of zero almost everywhere. This poses a challenge for gradient-based optimization algorithms that rely on gradient calculations for parameter updates [37].

To address this challenge, we introduce a surrogate QoS degradation function $\hat{\beta}_i^s$ that replaces the indicator function in (1) with a Sigmoid function $\phi(\rho * (q_i^s(t) - q_{thresh}^s))$, where ρ is a hyperparameter that controls the sharpness of the curve. The Sigmoid function is a smooth differentiable function and allows the use of gradient-based optimization methods, while still approximating the behavior of the indicator function. It is worth noting that we only utilize the surrogate formulation to approximate the gradient for iteratively updating the resource allocation $\boldsymbol{r}$ within the inner-loop in Algorithm 1. Whereas, the optimization objective and the constraints defined in (2) remain unchanged.

Remark: As the hyperparameter ρ increases, the Sigmoid approximation more closely matches the threshold function. However, an excessively large ρ can slow convergence due to diminishing gradients. Based on our numerical tests, even for moderate values of ρ, the approximation error is negligible relative to typical QoS threshold values. For example, an error of ± 0.001 is insignificant compared to QoS thresholds on the order of 10 Mbps. We recommend selecting ρ experimentally; in our setup, we set $\rho = 10^4$.

Algorithm 1 MicroOpt Algorithm

Input: Traffic $\boldsymbol{x}_i^s$, Slice Model $f_{QoS}^s(\boldsymbol{x}_i^s, \boldsymbol{r}_i^s)$, QoS threshold q_{thresh}^s, QoS degradation threshold β_{thresh}^s, $\tau_{1,max}, \tau_{2,max}, \alpha_1, \alpha_2, \alpha_3, \epsilon_1, \epsilon_2$
Output: Optimal resource allocation vector $\boldsymbol{r}_i^s$

1: Initialize $\boldsymbol{\lambda}, \boldsymbol{\mu}, \text{LB} = 0, \text{UB} = \infty, \tau_1 = 0, \tau_2 = 0$
2: **while** $\frac{\text{UB}-\text{LB}}{\text{UB}} > \epsilon_1$ **or** $\tau_1 < \tau_{1,max}$ **do**
3: $\quad \mathbf{r} \leftarrow \texttt{Initialization}(\boldsymbol{x}_i^s, f_{QoS}(\boldsymbol{x}_i^s, \mathbf{r}))$
4: $\quad$ **while** $|\nabla_r \hat{\mathcal{L}}| > \epsilon_2$ **or** $\tau_2 < \tau_{2,max}$ **do**
5: $\quad\quad \mathbf{r} \leftarrow [\mathbf{r} - \alpha_1 \nabla_r \hat{\mathcal{L}}]^+$
6: $\quad\quad \tau_2 \leftarrow \tau_2 + 1$
7: $\quad$ **end while**
8: $\quad \lambda_s \leftarrow [\lambda_s + \alpha_2(\beta_i^s - \beta_{thresh}^s)]^+, \forall s$
9: $\quad \mu_k \leftarrow [\mu_k + \alpha_3(\sum_{s\in \mathbf{S}} r^{s,k} - R^k)]^+, \forall k$
10: $\quad \text{LB} = \max(\text{LB}, \hat{\mathcal{L}}(\mathbf{r}, \boldsymbol{\mu}, \boldsymbol{\lambda}))$
11: $\quad \text{UB} = \min(\text{UB}, \sum_{s\in S} \boldsymbol{\eta}^\intercal \mathbf{r}^s)$
12: $\quad \tau_1 \leftarrow \tau_1 + 1$
13: **end while**
14: **return** $\mathbf{r}$

We denote the surrogate Lagrangian function, which incorporates the surrogate QoS degradation function, as $\hat{\mathcal{L}}(\mathbf{r}_i^s, \boldsymbol{\lambda}, \boldsymbol{\mu})$. With this formulation, we can apply analytical gradient optimization techniques to optimize the resource allocation, while the solution's feasibility is ensured using the strict definition of QoS degradation. It is worth noting that if the optimization problem is non-convex, gradient-based approaches may not always converge to the global minima, resulting in sub-optimal solutions. However, when initialized near the optimum, they are highly effective [38], [39], [40]. Therefore, in this paper, we propose randomly sampling a small number of points in the solution space to initialize the optimization process.

Algorithm 1 outlines the steps of our proposed solution. The algorithm takes as input the traffic $\boldsymbol{x}_i^s$ and the model $f_{QoS}^s(\boldsymbol{x}_i^s, \mathbf{r}_i^s)$ for each slice for the duration of reconfiguration interval. It also requires the QoS threshold q_{thresh}^s, the QoS degradation threshold β_{thresh}^s specific to each slice, and several hyper-parameters to control the algorithm's behaviour. These include parameters related to the stopping condition, such as $\tau_{1,max}$ and $\tau_{2,max}$, which define the maximum number of iterations for the outer and inner loops, respectively, and ϵ_1 and ϵ_2 that determine the desired level of convergence for the upper and lower bounds of the objective function. We also have learning rates, α_1, α_2, and α_3, for updating resource allocations and Lagrangian multipliers. Finally, the output of the algorithm is the optimal resource allocation for each slice.

The algorithm is comprised of outer and inner loops. Within the inner, the resource allocation variables are first initialized using through random sampling, shown in line 3. Subsequently, these variables are updated using the gradient of the surrogate Lagrangian function $(\nabla_r \hat{\mathcal{L}})$ in line 5. After updating the resource allocation variables in line 8 and 9, the algorithm updates the Lagrange multipliers inside the outer

loop. QoS constraint multipliers, λ_s, are updated based on the QoS degradation values β_i^s and threshold β_{thresh}^s for each slice. Notably, we adopt the strict definition of QoS degradation, *i.e.*, eq. (1), when updating the Lagrange multipliers in line 8 of Algorithm 1. This ensures that the final solution produced by the algorithm is strictly feasible. Resource constraints multipliers, μ_k, are updated for each resource k by considering the difference between the allocated resources $\sum_{s\in S} r_i^{s,k}$ and the resource capacity R^k. These updated variables are then projected into the non-negative domain, denoted by the notation $[.]^+$. At each point, the upper bound UB is equal to the best feasible solution found so far, while the lower bound LB is equal to the value of the Lagrangian function (line 10 and 11). Once the termination condition is met, the algorithm returns the resource allocation corresponding to the best LB.

1) Computational Complexity and Scalability: The computational complexity of Algorithm 1 is primarily determined by (i) the outer-loop that updates the Lagrangian penalty variables and runs for at most $\tau_{1,\max}$ iterations, and (ii) the inner-loop for resource allocation that runs for at most $\tau_{2,\max}$ iterations per outer iteration. Each inner-loop iteration requires a forward and backward pass through the slice model. Assuming a feed-forward neural network with $L+1$ layers, this results in a complexity of $O(\sum_{i=1}^{L} n_{i-1}\cdot n_i)$, where n_i denotes the number of neurons in the i^{th} layer. When each layer has approximately the same number of neurons n, the total parameter count grows as $O(Ln^2)$. Combining these factors, the worst-case complexity of MicroOpt is $O(\tau_{1,\max}\tau_{2,\max}Ln^2)$. This shows that adding a VNF, which primarily increases the input layer size n_0, results in a minor increase in computational complexity as parameters in the hidden layers dominate the overall computation. In contrast, increasing the number of hidden layers L or the number of neurons per layer can have a more significant impact on the convergence time.

However, the empirical convergence time of MicroOpt depends on several factors, including the neural network architecture, hyperparameter initialization (*e.g.*, Lagrangian multipliers and learning rates), the quality of the initial solution, the choice of gradient optimizer, the complexity of the learned functions, and the stopping criteria for both loops. In Section VI-D, we evaluate MicroOpt's convergence time against the state of the art.

V. Implementation

In this section, we describe the implementation of our network slicing testbed, shown in Fig. 3.

A. Testbed Infrastructure

Our testbed[2] consists of a substrate network deployed on a three-node Azure cluster. The virtual machines hosting the RAN, core, and transit networks are allocated 32 vCPUs, 16 vCPUs, and 8 vCPUs, respectively. Additionally, these virtual machines have RAM allocations of 64 GB, 32 GB, and 32 GB, respectively. This physical topology forms the foundation for our testbed, providing necessary computational and networking resources to support various network functions.

[2]The instructions for deploying the testbed can be found in [41].

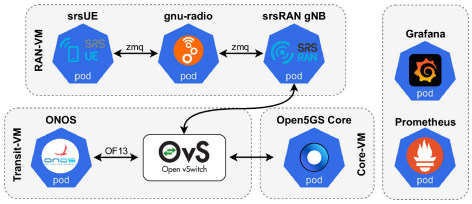


Fig. 3. Overview of our 5G testbed.

B. 5G Network Implementation

RAN. We implement the 5G RAN using the srsRAN project [42], an open-source software designed to create a 3GPP Release 17 (R17) compliant gNB. The User Equipments (UEs) are implemented using srsUE [43], a software implementation of a UE. Instead of physical radios for over-the-air transmissions between the gNB and UEs, we use virtual radios, also provided by srsRAN. Additionally, we utilize GNU Radio Companion to manage uplink and downlink signal between the UE and the gNB. GNU Radio offers a variety of signal processing blocks, enabling the emulation of complex functions such as path loss and user mobility.

Core. We implement the 5G mobile core based on Open5GS [44], an open-source implementation of 3GPP R17. The network functions, *e.g.*, AMF, SMF, UPF and NRF are containerized and deployed on our Kubernetes cluster. We consider a network slicing scenario where each slice has dedicated UPF and SMF network functions, while sharing other common 5G core functions such as AMF and NRF.

Transport. To establish the transport network (TN) infrastructure, we employ a software-defined VXLAN overlay utilizing Open vSwitch (OvS) [45] on the underlying physical substrate network. The OvS switch functions as the TN edge, where fine-grained bandwidth allocation and traffic policing are enforced [17]. However, as highlighted in [17], transit nodes within the TN do not perform fine-grained resource control. Instead, they rely on simplified traffic management mechanisms such as priority queuing and pre-emptive capacity planning to ensure QoS. Since our focus is on resource control in the transport network, we do not consider the transit nodes in our current testbed.

C. Control and Management

MANO. We use Kubernetes v1.29 for the management and orchestration (MANO) of our 5G network. This allows us to encapsulate different 5G network functions, including the RAN, into lightweight and portable containers. These containers can be dynamically deployed, scaled, and managed across our distributed cluster of nodes, providing flexibility and scalability. The Kubernetes API allows us to control the placement of these functions, and create network slices with desired topologies. To dynamically scale the CPU resources allocated to the network functions, we use Linux cgroups.

SDN Controller. We use the ONOS SDN controller [46] to enable precise control over the routing of network flows within our network slices. By communicating with the OvS switches in the VXLAN transport overlay, ONOS facilitates the routing of network slice traffic through OvS queues with specific rates, thus providing bandwidth slicing capabilities.

D. Data Collection

Edge Application. To test the proposed solution, we focus on generating enhanced Mobile Broadband (eMBB) user traffic. Note that we previously defined *slice traffic* as the number of users connected to a slice (users/s), whereas *user traffic* defined here is the traffic generated by the UEs (Mbps). To generate the user traffic, we begin by collecting a packet capture (pcap) dataset for 4K video streaming from YouTube. This dataset (available in [47]) is then replayed whenever a UE connects to the eMBB slice. Streaming 4K video can consume significant network resources, potentially degrading the QoS for other slices within the network. Therefore, it is crucial to dynamically scale the bandwidth allocation in the transport network and adjust the gNB CPU allocation for the eMBB slice.

Monitoring. To comprehensively monitor our testbed and gather datasets for our slice model, we implement a robust architecture using Prometheus and Grafana, as suggested by existing literature in [48]. Prometheus collects and stores time-series data on network traffic, resource utilization, and network function performance. Grafana provides a user-friendly platform for visualizing and analyzing this data.

Dataset Collection. We define QoS as the end-to-end throughput received by the UE, with varying QoS thresholds (q_{thresh}) of 3-5 Mbps and acceptable QoS degradation thresholds (β_{thresh}) of 0.1-0.3. Note that, for brevity, we omit the index s identifying the slice in the previously introduced notation for the rest of the paper. The targeted resources for scaling in this scenario are the transport network bandwidth and the gNB CPU with equal priority, *i.e.*, $\boldsymbol{\eta} = [\frac{1}{50}; \frac{1}{4500}]$, where 50 Mbps and 4500 millicores are the max resource allocations for the eMBB slice. Note that for the evaluation, the results show the normalized resource allocations, *i.e.*, $\boldsymbol{\eta}^\intercal \mathbf{r}_i$.

To train the slice model, we gather a QoS dataset (available in [49]) with the slice traffic varying from 1 to 5 users/s, the transport bandwidth from 5 to 40 Mbps, and the gNB CPU resource from 500 to 4000 millicores, in intervals of 1 users/s, 5 Mbps, and 500 millicores, respectively.

VI. Performance Evaluation

A. Experiment Setup and Comparison Approaches

1) Slice Model: We presented a high-level overview of the slice model in Fig. 2. However, the specific details of the model, including the number of layers, activation functions, and nodes per layer, may vary for different datasets collected from other testbeds. In our model, after the input layer, we add a batch normalization layer to normalize the inputs which leads to improved model performance by reducing the internal covariate shift. Subsequently, we use three shared hidden layers with 16 nodes each and Rectified Linear Unit (ReLU) activation. The mean and standard deviation branches have one hidden layer with 16 nodes and ReLU activation.

The mean output uses a Linear activation function, while the standard deviation output employs the Softplus activation function to ensure non-negativity. It should be noted that the slice model's specific details, such as its structure, the number of layers, activation functions, and nodes per layer, can vary for different testbeds.

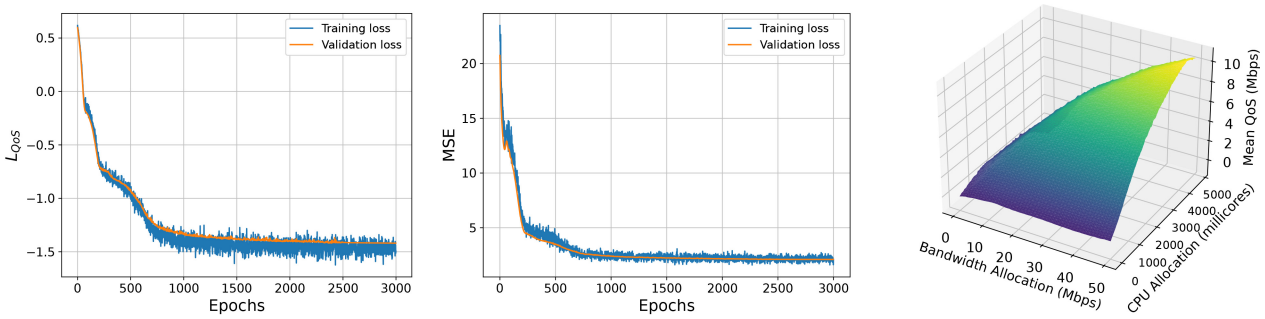
We divide the QoS dataset (*cf.*, Section V-D) into training and validation sets. Additionally, we perform the procedure outlined in Section V-D to gather a test set consisting entirely of off-grid points, *i.e.*, input combinations not present in the training or the validation sets. Subsequently, we train the model for 3,000 epochs with a learning rate of 0.001, and reduce the learning rate by a factor of 10 after 1,500 epochs. Figures 4a and 4b show the negative log probability loss (*i.e.*, L_{QoS} in (4)), and mean squared error (MSE), respectively, as the model trains. We can see that the MSE follows the same trend as L_{QoS}. Additionally, from the figures, we can see that the validation error does not deviate from the training error, which shows that the model is not overfitting to the training data. Once trained, the slice model requires an inference time of only 0.528ms, and achieves a L_{QoS} of -1.53, -1.72, and -1.67, an MSE of 1.58, 1.24, and 0.91, and a mean absolute error (MAE) of 0.69, 0.62 and 0.54 on the training, validation, and test datasets, respectively.

Finally, we visualize the trained slice model by plotting the mean QoS predicted at the entire range of input values in Fig. 4c. This figure shows the predicted QoS mean, averaged over 1-5 users/s. We can see that at lower CPU allocations, the QoS does not increase with bandwidth, as the gNB is the bottleneck. Once the CPU allocation exceeds 100 millicores, the effect of bandwidth on the QoS becomes significant.

2) Comparison Approaches: To evaluate the proposed approach, we implement the following state-of-the-art approaches:

Peak-alloc: This approach serves as a baseline where the network operator statically allocates resources to a slice based solely on peak slice traffic (5 users/s) and strict QoS requirements ($q_{thresh} = 5.0$, $\beta_{thresh} = 0.01$). To determine the optimal resource allocation for this scenario, we employ a brute-force method, specifically a fine-grained grid search, which took several hours to find the solution.

Atlas, Atlas$^+$: This approach, based on [7], uses a network simulator to train a Bayesian neural network (BNN) for learning the QoS degradation function in eqn. (1). BO with Thompson sampling and Lagrangian relaxation is then used for resource optimization. In our implementation, we replace the network simulator with our slice model. The original method in [7] works only for constant slice traffic (*i.e.*, peak traffic in the slice traffic distribution) because it requires pre-training the BNN with an expensive simulator, which is impractical for infinite possible traffic distributions. We propose using a DNN-based slice model, which makes the pre-training significantly faster and can be done after predicting the actual slice traffic distribution. This allows the method to handle slice traffic distributions. We refer to the original and the enhanced methods as Atlas and Atlas$^+$, respectively.



(a) Negative Log probability loss (L_{QoS} in (4)); Test L_{QoS}: −1.67

(b) Mean Squared Error (MSE) Test MSE: 0.62

(c) Slice model visualization

Fig. 4. Slice model training and visualization.

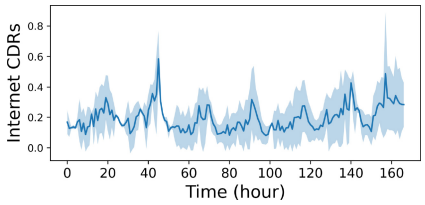


Fig. 5. Telecom Italia dataset traffic trend over a week. (Shaded region shows std. dev.).

CaDRL, CaDRL$^+$: Several works have proposed constrained DRL for dynamic resource scaling [5], [8], [13]. We implement the recent approach by Liu et al. [13], which uses Interior-point Policy Optimization (IPO) for this purpose, known as Constraint-aware Deep Reinforcement Learning (CaDRL). However, constrained DRL approaches converge to the worst-case scenario (*i.e.*, peak traffic) when trained to generalize over multiple slice traffic distributions, as shown by Sulaiman et al. [8]. We propose using a computationally inexpensive slice model to quickly train the policy from scratch once the actual slice traffic has been predicted, avoiding the need to generalize over multiple distributions. We refer to these methods as CaDRL and CaDRL$^+$, respectively.

3) Slice Traffic Model: To generate different slice traffic distributions, we use the Telecom Italia dataset [2], which contains anonymized telecommunication activity in Milan and the Province of Trentino. Focusing on an eMBB slice, we extract one week of Internet call detail records (CDRs), which are generated every time a user initiates or ends an Internet connection. Fig. 5 illustrates the normalized traffic variation for a randomly selected cell, showing significant hourly variation. Subsequently, we calculate the mean and max standard deviation of this data, referred to as σ_{mean} and σ_{max}, respectively. We then generate 10 different slice traffic distributions using a truncated normal distribution centered at 1-5 users/s with standard deviations of σ_{mean} and σ_{max}, referred to as dataset slice traffic distributions. Once a user connects to the slice, the user traffic is generated as described in Section V-D.

B. Simulation Results

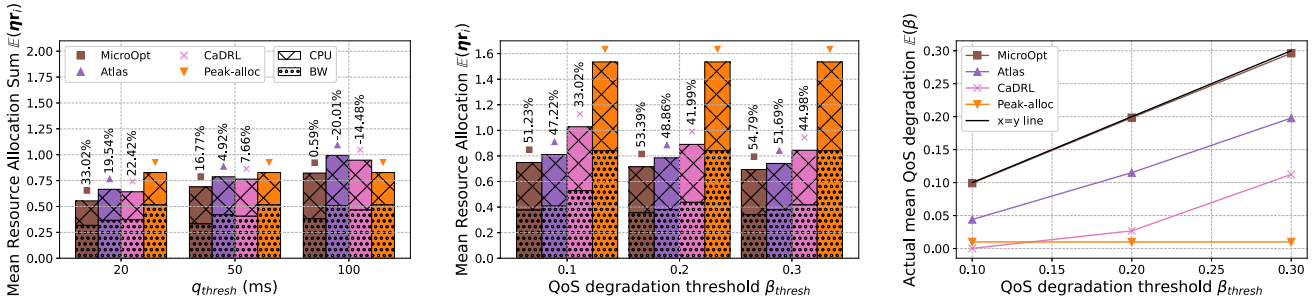
In this section, we compare different approaches for deriving the optimal resource allocation of the constrained optimization problem in (2), given the network model (*cf.*, Section IV-A), and the slice traffic $\boldsymbol{x}_i$ (*cf.*, Section VI-A3).

TABLE I
RESOURCE ALLOCATION AND QOS DEG. BOLD = BETTER)

Approach	Slice traffic ($\boldsymbol{x}_i$) (users/s)	CPU (millicores)	BW (Mbps)	QoS degr. $\mathbb{E}(\beta)$
Atlas	1	1167.60	**10.03**	0.06
	2	1648.72	**14.84**	0.07
	3	2452.79	29.79	0.00
	4	2589.30	30.89	0.05
	5	**2934.51**	42.11	0.07
CaDRL	1	1632.14	13.27	0.00
	2	2011.99	19.79	0.00
	3	2574.88	33.23	0.00
	4	3089.35	47.89	0.00
	5	3890.13	50.00	0.00
MicroOpt	1	**993.93**	11.73	**0.10**
	2	**1463.44**	15.77	**0.10**
	3	**2140.87**	**22.22**	**0.10**
	4	**2534.28**	**28.94**	**0.10**
	5	2945.72	**39.20**	**0.10**
Peak-alloc	5	3123.80	42.20	0.01

Constant Slice Traffic. First, we evaluate the different approaches at a constant slice traffic varying from 1-5 users/s, with a QoS threshold of $q_{thresh} = 5.0$ and a QoS degradation threshold of $\beta_{thresh} = 0.1$. Atlas$^+$ and CaDRL$^+$ are excluded from this evaluation because, with a constant number of users, they perform the same as their counterparts. Table I presents the corresponding resource allocation and QoS degradation achieved by the different approaches in this scenario. From the table, we can see that MicroOpt allocates the minimum resources while keeping the QoS degradation $\mathbb{E}(\beta)$ under the required threshold of 0.1. CaDRL, despite achieving similar resource allocation as the other approaches, fails to increase the QoS degradation above 0, resulting in the highest resource allocation. Atlas maintains a QoS degradation close to 0.1 in most scenarios, with resource allocation similar to MicroOpt. Finally, Peak-alloc allocates more resources compared to MicroOpt and Atlas, as it only finds the optimal solution for a QoS degradation of $\beta_{thresh} = 0.01$.

Next, we evaluate the different approaches across varying QoS and QoS degradation thresholds, omitting the tabular presentation of results for brevity. Fig. 6a and Fig. 6b show the normalized mean resource allocation and QoS



(a) Mean resource allocation vs. QoS threshold (q_{thresh}), averaged QoS degradation threshold (β_{thresh})

(b) Mean resource allocation vs. QoS degradation threshold (β_{thresh}), averaged over QoS threshold (q_{thresh})

(c) Mean QoS degradation $\mathbb{E}(\beta)$ vs. varying QoS degradation threshold β_{thresh}, averaged over QoS threshold (q_{thresh})

Fig. 6. Mean resource allocation sum $\mathbb{E}(\boldsymbol{\eta}\mathbf{r}_i)$ and mean QoS degradation $\mathbb{E}(\beta)$ at various parameter settings.

degradation for various approaches across different values of $q_{thresh} = [3.0, 4.0, 5.0]$ and $\beta_{thresh} = [0.1, 0.2, 0.3]$, along with the percentage improvement over the baseline Peak-alloc. Fig. 6a shows that the proportions of CPU and bandwidth resources within the overall resource allocations are both approximately 1/2. This is because we used equal weights for CPU and bandwidth when solving (2), *i.e.*, with $\boldsymbol{\eta} = \mathbf{1} \cdot [\frac{1}{50}, \frac{1}{4500}]$.

From the figures, we can see a direct relationship between the QoS requirement (q_{thresh}) and resource allocation—as the QoS requirement decreases, the resource allocation also decreases. In contrast, Fig. 6b demonstrates an inverse relationship between resource allocation and the QoS degradation threshold (β_{thresh})—as the QoS degradation threshold increases, indicating a tolerance for higher degradation, a lower resource allocation is required. Across different parameter values, the optimal resource allocation follows a consistent pattern—MicroOpt allocates the lowest resources, followed by Atlas, CaDRL, and Peak-alloc. Averaging over all parameter values, the respective approaches allocate 0.719, 0.778, 0.920, and 1.534 units of resources. This shows that MicroOpt results in a 7.65%, 21.90%, and 53.14% decrease in resource usage compared to Atlas, CaDRL, and Peak-alloc, respectively.

Fig. 6c shows the QoS degradation achieved by the different approaches as they minimize resource allocation. The $x = y$ regression line shows the curve where the achieved QoS degradation equals the threshold. Since all the points lie below the $x = y$ regression line, it shows that all approaches achieve mean QoS degradation below the required threshold. Additionally, the mean QoS degradation follows the opposite trend to resource allocation—Peak-alloc achieves the lowest QoS degradation, followed by CaDRL, Atlas, and MicroOpt. Interestingly, unlike the previous tabular results, CaDRL achieves a higher QoS degradation and a lower resource allocation than Peak-alloc at more relaxed parameter settings (*i.e.*, lower q_{thresh}, and higher β_{thresh}). This is because CaDRL's algorithm highly prioritizes maintaining strict QoS degradation adherence over minimizing resource allocation. Therefore, it is able to perform better when the β_{thresh} is lower.

Dynamic Slice Traffic. To evaluate the different approaches when slice traffic varies within a resource reconfiguration interval, *i.e.*, when slice traffic is a distribution rather than constant, we use the slice traffic distributions dataset described in Section VI-A3. Fig. 7 shows the mean resource allocation and the percentage improvement over Peak-alloc, while Fig. 8 illustrates the mean QoS degradation by the different approaches at a QoS threshold of $q_{thresh} = 5.0$ and a QoS degradation threshold of $\beta_{thresh} = 0.1$.

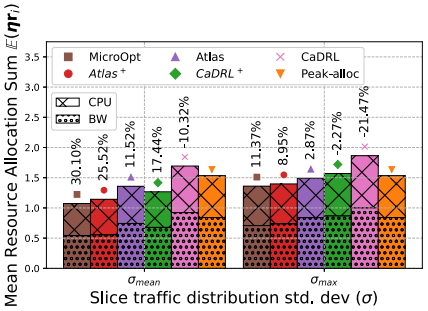


Fig. 7. Mean resource allocation vs. slice traffic std. dev.

From Fig. 7, it is evident that as the slice traffic std. dev. increases from σ_{mean} to σ_{max}, the corresponding resource allocation also increases. This is because a higher std. dev. indicates a broader distribution of slice traffic, which necessitates more resources to maintain the mean QoS for the entire range of users, including those in the tail of the distribution.

Focusing on the resource allocation across the different approaches in Fig. 7, we can see that it follows the same trend as in the constant traffic scenario—MicroOpt allocates the least resources, compared to Atlas and CaDRL. However, with dynamic slice traffic, Atlas$^+$ and CaDRL$^+$ allocate fewer resources than their counterparts. This is because, as described in Section VI-A2, these approaches consider the actual slice traffic distribution rather than just the peak traffic. In this scenario, MicroOpt shows a mean decrease of 4.23%, 14.23%, 14.60%, 31.61%, and 20.74% compared to Atlas$^+$, CaDRL$^+$, Atlas, CaDRL, and Peak-alloc, respectively, across the 10 different slice traffic distributions.

In Fig. 8, we observe that all the different approaches achieve QoS degradation below the required threshold of $\beta_{thresh} = 0.1$. Additionally, the QoS degradation follows the expected trend based on resource allocations, *i.e.*, approaches with higher resource allocation result in lower QoS degradation. In this scenario ($q_{thresh} = 5.0, \beta_{thresh} = 0.1$),

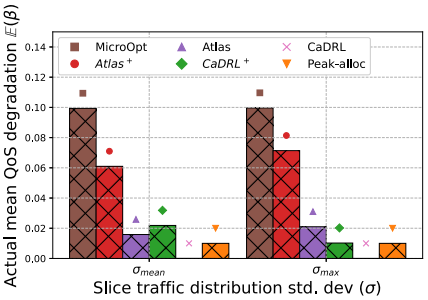


Fig. 8. Mean QoS degradation $\mathbb{E}(\beta)$ vs. slice traffic std. dev.

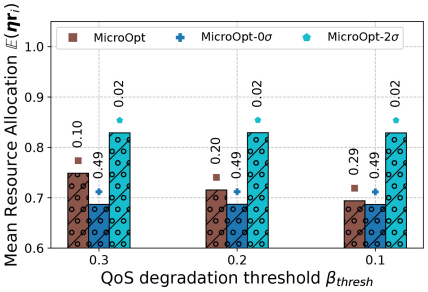


Fig. 9. Res. alloc (y-axis) and QoS deg. (β) (value above bar), at various β_{thresh}.

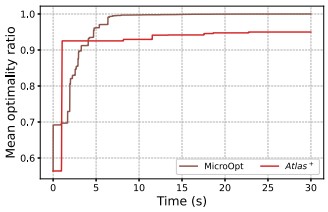


Fig. 10. Convergence for MicroOpt and Atlas$^+$.

RL-based approaches perform worse compared to the Peak-alloc approach. This is because RL-based approaches provide sub-optimal solutions, and when the parameters are similar to those of Peak-alloc, which is derived from a brute-force approach, the sub-optimality becomes evident.

C. Ablation Study

The slice model proposed in Section IV-A learns a QoS distribution for any given input, requiring the use of the reparameterization trick for gradient calculation. The output QoS can be a distribution due to factors such as user mobility, non-deterministic transmission medium, and network function behavior. In this section, we investigate the scenario where the slice model only learns a scalar QoS value, *i.e.*, the mean (μ) or two std. dev. below the mean (*i.e.*, $\mu - 2\sigma$) in order to ensure QoS satisfaction. This allows direct gradient computation but ignores the actual underlying distribution. We refer to these approaches as MicroOpt-0σ and MicroOpt-2σ, respectively.

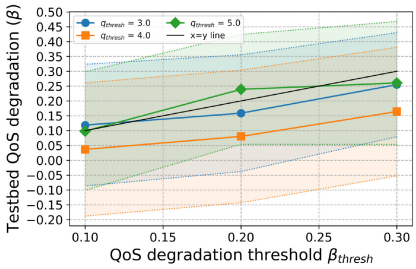
For evaluation, we solve the constrained optimization for constant slice traffic under different QoS and QoS degradation thresholds (*cf.*, Section VI-B). Fig. 9 shows the mean resource allocation and the corresponding QoS degradation achieved at various QoS degradation thresholds. From the figure, we observe that although MicroOpt-0σ allocates the least resources, it maintains a QoS degradation of 0.49 which is significantly higher than the thresholds. This occurs because, with a normal distribution, half of the QoS values fall below the mean, leading to a QoS degradation of 0.5. On the other hand, MicroOpt-2σ achieves small QoS degradation but allocates significantly higher resource than MicroOpt. Finally MicroOpt considers the QoS as a distribution, accounting for deviations from the mean or tail QoS. This leads to the highest resource saving while maintaining the QoS degradation under the required threshold. This ablation study highlights the importance of modeling QoS as a distribution.

D. Convergence, Optimality and Feasiblity

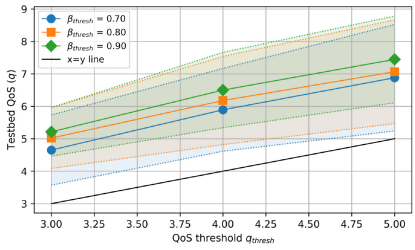
Recent advancements in slice management and orchestration frameworks [50] can support slice resource reconfiguration in short time scales. However, most existing resource scaling algorithms (*e.g.*, [5]) still require over an hour for convergence. Through empirical tests, our approach MicroOpt is shown to significantly mitigate the convergence time while attaining a resource allocation close to the optimal solution. This is achieved by employing fast network model queries combined with gradient descent optimization. In the dynamic traffic scenario, the average convergence times for Peak-alloc, CaDRL, and CaDRL$^+$ are 4.96h, 199.56s, and 201.67s, respectively. In contrast, MicroOpt, Atlas, and Atlas$^+$ converge significantly faster, with average times of 30.06s, 30.80s, and 27.83s, respectively.

Fig. 10 illustrates the mean optimality ratio, which represents the resource allocation ratio of the current solution to the best solution achieved by either approach over time. The fast initial improvement in both curves are attributed to the random search phase in MicroOpt and the pre-training of the BNN in Atlas$^+$. The figure shows that while Atlas$^+$ (using Bayesian Optimization) improves slowly after the initial solution, MicroOpt (using gradient descent) rapidly reaches near-optimal solutions. Specifically, MicroOpt achieves 99.60% of the final solution within just 7.5 seconds. These convergence times are measured on a CPU-based machine and can improve even further with GPUs, which offer significantly faster neural network inference times. This demonstrates that MicroOpt is better suited for fast resource allocation compared to current state-of-the-art methods.

Finally, it is important to emphasize that since the slice model learns a non-convex function, the primal-dual algorithm does not guarantee convergence to the optimal solution. However, the proposed approach ensures a feasible solution that satisfies QoS constraints, as long as the initial solution is feasible. This is achieved because the Lagrangian penalty variables in Algorithm 1 continuously increase until the gradient descent converges to a feasible solution. However, if a termination condition is reached–such as a time limit or a maximum number of iterations–the algorithm returns the most recent feasible solution. To ensure feasibility, starting with a 100% resource allocation (*i.e.*, $\mathbf{r} = [1.0, 1.0]$) is recommended if no better solution is identified during the initial random search.



(a) Testbed QoS degradation (β)



(b) Testbed QoS (q)

Fig. 11. Testbed evaluation of MicroOpt (solid lines shows the mean, and shaded region shows the ± 1 std).

E. Testbed Evaluation

The feasibility of solutions determined by the slice model may not necessarily translate to feasibility within the real-world network if the model is inaccurate. Therefore, even though the proposed approach achieves the least resource consumption while satisfying QoS degradation constraints (*cf.*, Fig. 6c) based on the slice model, these solutions must be validated on the testbed to ensure the accuracy of the slice model. For this purpose, we generate the slice traffic with σ_{mean} described in Section VI-A3 on our testbed by using automated scripts for the arrival and departure of UEs. The resource allocation is then determined by solving the constrained optimization problem using Algorithm 1.

Fig. 11a shows the QoS distribution achieved at different parameter values. From the figure we can see that in all the different cases, the mean QoS stays above the QoS threshold ($x = y$ line). However, as QoS degradation threshold increases, the QoS distribution approaches q_{thresh}. In all parameter settings, even though the mean QoS stays above q_{thresh}, a part of the QoS distribution does fall below it which causes QoS degradation. We show this QoS degradation in Fig. 11b. The figure indicates a positive correlation between the QoS degradation threshold and the mean QoS degradation. From the figure, we can see that as simulation progresses, the QoS degradation fluctuates around the QoS degradation threshold, and the mean converges close to the QoS degradation threshold. It is worth noting that the problem statement (*cf.*, Section III) requires the mean QoS degradation ($\mathbb{E}(\beta_i^s)$) to be above the threshold β_{thresh}. From Fig. 11b, we can see that most parameter satisfy the QoS degradation constraint (*i.e.*, the mean QoS degradation is below the QoS degradation threshold). However, for specific configurations ($\beta_{thresh}, q_{thresh} = (0.1, 3.0)$ *and* $(0.2, 5.0)$), it slightly exceeds the threshold. This occurs when the slice model overestimates QoS, leading to higher than expected QoS degradation. This limitation is inherent to approaches using surrogate models of the real network and can be mitigated by introducing a safety margin, fine-tuning the solution during online operation [7], or switching to a safe solution if the QoS degradation exceeds the threshold [11].

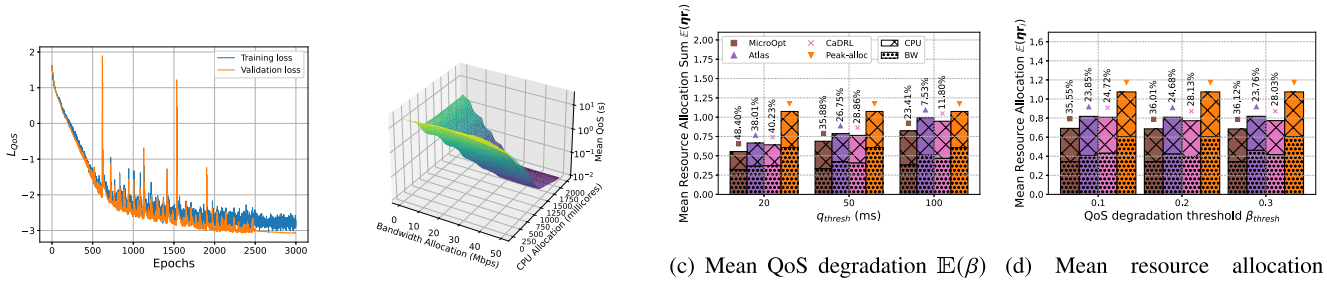
To gauge the extent of our slice model's under-prediction, we evaluated the different losses of the model using a newly gathered dataset that includes evaluation slice traffic, the resource allocations performed, and the corresponding QoS achieved. On this data, the model exhibited a L_{QoS} of –1.53, an MSE of 1.14, and an MAE of 0.72. These values are consistent with the training, validation, and test losses, indicating the model's reliability. The alignment of these metrics across different datasets suggests that as the slice model undergoes further fine-tuning, it can more accurately predict QoS degradation. Consequently, the 5G testbed's QoS degradation will better align with the set thresholds.

F. Generalization

In the preceding discussion, we evaluated MicroOpt's performance using the traffic described in Section V-D, with throughput (or packet loss) as the key performance indicator (KPI). However, since 5G networks are designed to support diverse use cases, real-world traffic patterns may differ from the evaluated distribution, and the relevant KPIs may vary accordingly. Therefore, in this section, we assess MicroOpt's performance under Poisson-distributed traffic and redefine QoS in terms of packet delay.[3] We choose Poisson traffic because it provides a widely-used [9], [26] and generalized traffic model, allowing us to evaluate MicroOpt's performance without needing separate datasets for different 5G use-cases such as gaming, video streaming, or autonomous driving. Additionally, we consider delay as the QoS metric, as it–alongside the previously evaluated throughput KPI–captures the key requirements of the two most common 3GPP slice types: eMBB and URLLC. For evaluation, we gather a new dataset using the procedure discussion in Section V-D. Our testbed achieves a min delay of about 10ms, similar to other research-grade testbeds, *e.g.*, [7], [11], [24]. Therefore, we evaluate MicroOpt for QoS thresholds of 20ms, 50ms and 100ms.

Fig. 12a illustrates the training loss throughout model training. We observe a similar trend as in Fig. 4, where the loss steadily decreases as training progresses, and both training and validation losses converge to similar values within 3000 epochs. Fig. 12b presents the corresponding network model visualization, showing the expected relationship between resource allocation and packet delay–as resource allocation increases, packet delay decreases, eventually converging to around 10ms at 2000 millicores and 40 Mbps bandwidth allocation. However, an unexpected trend emerges at lower resource allocations, where packet delay can reach several seconds. This suggests that instead of being dropped, packets entering the network may be queued for extended periods

[3]Strictly following the definition in Section III, q_{thresh} should be defined as $-1 * \texttt{packet delay}$, where higher values indicate better performance.



(a) Negative Log probability loss

(b) Slice model visualization

(c) Mean QoS degradation $\mathbb{E}(\beta)$ vs. varying QoS degradation threshold β_{thresh}, averaged over QoS threshold (q_{thresh})

(d) Mean resource allocation vs. QoS degradation threshold (β_{thresh}), averaged over QoS threshold (q_{thresh})

Fig. 12. Slice model and resource allocation results at various parameter settings.

before processing, indicating a complex trade-off between delay and packet loss in resource-constrained conditions.

Finally, Fig. 12c and Fig. 12d illustrate the resource allocation achieved by different approaches across various QoS thresholds and QoS degradation thresholds. As seen in Fig. 12, there is an expected correlation between these thresholds and resource allocation–stricter thresholds generally require more resources. Among the evaluated approaches, MicroOpt consistently achieves the lowest resource allocation. Interestingly, unlike previous results, the Atlas approach does not perform as well in this case. Across all parameter settings, CaDRL outperforms Atlas by allocation lesser resources. Therefore, MicroOpt achieves the least resource allocation followed by CaDRL, Atlas and finally, Peak-alloc. We see expected trends from the counterparts to Fig. 7 and Fig. 8. For brevity, we omit their detailed discussion here.

VII. Conclusion

In this paper, we presented the MicroOpt framework, a novel approach for end-to-end dynamic resource allocation in 5G and beyond network slices. The framework leverages a DNN with the reparameterization trick to learn a differentiable slice model, which is then used in a primal-dual optimization algorithm to minimize the resource allocation under QoS constraints. We evaluated the proposed framework in multiple scenarios and showed that it can achieve up to 21.9% reduction in resource allocation compared to the state-of-the-art approaches, while also satisfying the QoS degradation constraints. Finally, we deployed an open-source 5G testbed with data collection and scaling capability for validating the results and showed that the proposed solution is feasible in various scenarios.

Our future work will explore incorporating a feedback mechanism to address inaccuracies in traffic prediction or the slice model. Additionally, investigating resource allocation in scenarios with multiple slices and resource contention is essential for understanding how the MicroOpt framework handles competition for scarce resources. Finally, we plan to scale up the testbed to include a full transport network and an integrated radio physical layer.

Acknowledgment

The authors thank Niloy Saha for his assistance in setting up the 5G testbed.

References

[1] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5G: Survey and challenges," *IEEE Commun. Mag.*, vol. 55, no. 5, pp. 94–100, May 2017.

[2] G. Barlacchi et al., "A multi-source dataset of urban life in the city of milan and the province of Trentino," *Sci. Data*, vol. 2, no. 1, pp. 1–5, Oct. 2015. [Online]. Available: http://dx.doi.org/10.1038/sdata.2015.55

[3] M. A. Habibi, B. Han, M. Nasimi, and H. D. Schotten, "The structure of service level agreement of slice-based 5G network," 2018, *arXiv:1806.10426*.

[4] H. N. Qureshi, M. Manalastas, S. M. A. Zaidi, A. Imran, and M. O. Al Kalaa, "Service level agreements for 5G and beyond: Overview, challenges and enablers of 5G-healthcare systems," *IEEE Access*, vol. 9, pp. 1044–1061, 2020.

[5] Q. Liu, N. Choi, and T. Han, "Constraint-aware deep reinforcement learning for end-to-end resource orchestration in mobile networks," in *Proc. IEEE Int. Conf. Netw. Protocols (ICNP)*, 2021, pp. 1–11.

[6] J. Li, J. Liu, T. Huang, and Y. Liu, "DRA-IG: The balance of performance isolation and resource utilization efficiency in network slicing," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2020, pp. 1–6.

[7] Q. Liu, N. Choi, and T. Han, "Atlas: Automate online service configuration in network slicing," in *Proc. Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2022, pp. 140–155.

[8] M. Sulaiman, M. Ahmadi, M. A. Salahuddin, R. Boutaba, and A. Saleh, "Generalizable resource scaling of 5G slices using constrained reinforcement learning," in *Proc. IEEE/IFIP Netw. Oper. Manag. Symp. (NOMS)*, 2023, pp. 1–9.

[9] M. Ferriol-Galmés et al., "RouteNet-fermi: Network modeling with graph neural networks," *IEEE/ACM Trans. Netw.*, vol. 31, no. 6, pp. 3080–3095, May 2023.

[10] T. Hu, Q. Liao, Q. Liu, A. Massaro, and G. Carle, "Fast and scalable network slicing by integrating deep learning with Lagrangian methods," 2024, *arXiv:2401.11731*.

[11] Q. Liu, N. Choi, and T. Han, "OnSlicing: Online end-to-end network slicing with reinforcement learning," in *Proc. ACM Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2021, pp. 141–153.

[12] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," 2013, *arXiv:1312.6114*.

[13] Y. Liu, J. Ding, and X. Liu, "A constrained reinforcement learning based approach for network slicing," in *Proc. IEEE Int. Conf. Netw. Protocols (ICNP)*, 2020, pp. 1–6.

[14] M. Alsenwi, N. H. Tran, M. Bennis, S. R. Pandey, A. K. Bairagi, and C. S. Hong, "Intelligent resource slicing for eMBB and URLLC coexistence in 5G and beyond: A deep reinforcement learning based approach," *IEEE Trans. Wireless Commun.*, vol. 20, no. 7, pp. 4585–4600, Jul. 2021.

[15] R. Ali, Y. B. Zikria, A. K. Bashir, S. Garg, and H. S. Kim, "URLLC for 5G and beyond: Requirements, enabling incumbent technologies and network intelligence," *IEEE Access*, vol. 9, pp. 67064–67095, 2021.

[16] S. Zhang, "An overview of network slicing for 5G," *IEEE Wireless Commun.*, vol. 26, no. 3, pp. 111–117, Jun. 2019.

[17] A. Encinas-Alonso, C. M. Lentisco, I. Soto, L. Bellido, and D. Fernandez, "A slicing model for transport networks with traffic burst control and QoS compliance for traffic flows," *IEEE Open J. Commun. Soc.*, vol. 6, pp. 2152–2176, 2025.

[18] A. T. Z. Kasgari and W. Saad, "Stochastic optimization and control framework for 5G network slicing with effective isolation," in *Proc. IEEE Annu. Conf. Inf. Sci. Syst. (CISS)*, 2018, pp. 1–6.

[19] J. X. Salvat, L. Zanzi, A. Garcia-Saavedra, V. Sciancalepore, and X. Costa-Perez, "Overbooking network slices through yield-driven end-to-end orchestration," in *Proc. ACM Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2018, pp. 353–365.
[20] R. Li et al., "Deep reinforcement learning for resource management in network slicing," *IEEE Access*, vol. 6, pp. 74429–74441, 2018.
[21] F. Wei, S. Qin, G. Feng, Y. Sun, J. Wang, and Y.-C. Liang, "Hybrid model-data driven network slice reconfiguration by exploiting prediction interval and robust optimization," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 2, pp. 1426–1441, Jun. 2022.
[22] A. Baumgartner, T. Bauschert, A. A. Blzarour, and V. S. Reddy, "Network slice embedding under traffic uncertainties—A light robust approach," in *Proc. 13th Int. Conf. Netw. Service Manage. (CNSM)*, 2017, pp. 1–5.
[23] A. Baumgartner, T. Bauschert, A. M. C. A. Koster, and V. S. Reddy, "Optimisation models for robust and survivable network slice design: A comparative analysis," in *Proc. GLOBECOM IEEE Global Commun. Conf.*, 2017, pp. 1–7.
[24] A. Balasingam, M. Kotaru, and P. Bahl, "Application-level service assurance with 5G RAN slicing," in *Proc. 21st USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2024, pp. 841–857.
[25] V. Sciancalepore, K. Samdanis, X. Costa-Perez, D. Bega, M. Gramaglia, and A. Banchs, "Mobile traffic forecasting for maximizing 5G network slicing resource utilization," in *Proc. INFOCOM IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.
[26] Q. Yang et al., "DeepQueueNet: Towards scalable and generalized network performance estimation with packet-level visibility," in *Proc. ACM SIGCOMM*, 2022, pp. 441–457.
[27] O. Iacoboaiea, J. Krolikowski, Z. B. Houidi, and D. Rossi, "From design to deployment of zero-touch deep reinforcement learning WLANs," *IEEE Commun. Mag.*, vol. 61, no. 2, pp. 104–109, 2023.
[28] M. Sulaiman, A. Moayyedi, M. Ahmadi, M. A. Salahuddin, R. Boutaba, and A. Saleh, "Coordinated slicing and admission control using multi-agent deep reinforcement learning," *IEEE Trans. Netw. Service Manag.*, vol. 20, no. 2, pp. 1110–1124, Jun. 2023.
[29] V. Sciancalepore, X. Costa-Perez, and A. Banchs, "RL-NSB: Reinforcement learning-based 5G network slice broker," *IEEE/ACM Trans. Netw.*, vol. 27, no. 4, pp. 1543–1557, Jul. 2019.
[30] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "DeepCog: Cognitive network management in sliced 5G networks with deep learning," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2019, pp. 280–288.
[31] C. Marquez, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "How should I slice my network? A multi-service empirical evaluation of resource sharing efficiency," in *Proc. Int. Conf. Mobile Comput. Netw. (MobiCom)*, 2018, pp. 191–206.
[32] K. Papagiannaki, N. Taft, Z.-L. Zhang, and C. Diot, "Long-term forecasting of Internet backbone traffic: Observations and initial models," in *Proc. INFOCOM 22nd Annu. Joint Conf. IEEE Comput. Commun. Soc.*, 2003, pp. 1178–1188.
[33] C. Bishop, *Mixture Density Networks*, Aston Univ., Birmingham, U.K., 1994.
[34] A. Paszke et al., "Automatic differentiation in PyTorch," in *Proc. 31st Conf. Neural Inf. Process. Syst.*, 2017, pp. 1–4.
[35] A. Graves, "Stochastic backpropagation through mixture density distributions," 2016, *arXiv:1607.05690*.
[36] S. P. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.
[37] A. Cotter, H. Jiang, and K. Sridharan, "Two-player games for efficient non-convex constrained optimization," in *Proc. Algorithmic Learn. Theory*, 2019, pp. 300–332.
[38] J. D. Lee, I. Panageas, G. Piliouras, M. Simchowitz, M. I. Jordan, and B. Recht, "First-order methods almost always avoid strict saddle points," *Math. Program.*, vol. 176, pp. 311–337, Jul. 2019.
[39] E. Busseti, W. M. Moursi, and S. P. Boyd, "Solution refinement at regular points of conic problems," *Comput. Optim. Appl.*, vol. 74, pp. 627–643, Dec. 2019.
[40] P. L. Donti, D. Rolnick, and J. Z. Kolter, "DC3: A learning method for optimization with hard constraints," 2021, *arXiv:2104.12225*.
[41] "Sulaimanalmani/k8s_srsran_open5gs: Containerized/kubernetes deployment of E2E 5G testbed using srsRaN and Open5gs." Github.com. Accessed: Feb. 14, 2025. [Online]. Available: https://github.com/sulaimanalmani/k8s_srsran_open5gs.
[42] "5G-srsRAN project." Accessed: May 17, 2025. [Online]. Available: https://www.srsran.com/
[43] "Introduction—srsRAN 4G 23.11 documentation." Accessed: May 19, 2025. [Online]. Available: https://docs.srsran.com/projects/4g/en/latest/usermanuals/source/srsue/source/1_ue_intro.html
[44] "Open5GS." Accessed: Feb. 14, 2025. [Online]. Available: https://open5gs.org/
[45] (Linux Found., San Francisco, CA, USA). *OpenVSwitch, Version 2.9.8*. 2023. [Online]. Available: https://www.openvswitch.org/
[46] (Open Netw. Found., New Delhi, India). *ONOS, Version 2.5.7-rc1*. 2023. [Online]. Available: https://github.com/opennetworkinglab/onos
[47] "Dropbox." dropbox.com. Accessed: Feb. 14, 2025. [Online]. Available: https://www.dropbox.com/scl/fi/vgv2v8tidcduz39gozn3f/ytdl.pcap?rlkey=gipq1lpghzupfwj41coni8gir&st=l96qtq90&dl=0.
[48] N. Saha, N. Shahriar, R. Boutaba, and A. Saleh, "MonArch: Network slice monitoring architecture for cloud native 5G deployments," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp. (NOMS)*, 2023, pp. 1–7.
[49] "Sulaimanalmani/5G-resource-allocation-dataset: Dataset and resources for 5G network resource allocation models." Github.com. Accessed: Feb. 14, 2025. [Online]. Available: https://github.com/sulaimanalmani/5G-resource-allocation-dataset.git.
[50] "Autoscaling workloads." 2024. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/autoscaling/

Muhammad Sulaiman (Graduate Student Member, IEEE) received the B.S. degree in electrical engineering from the National University of Sciences and Technology, Pakistan, in 2019. He is currently pursuing the Ph.D. degree with the University of Waterloo. His research focuses on autonomous 5G/6G network management and orchestration using AI. His work was nominated for IEEE/IFIP NOMS best paper award in 2022 and 2023.

Mahdieh Ahmadi received the B.S. degree in computer engineering from the University of Tehran, Iran, in 2013, and the M.Sc. and Ph.D. degrees in computer engineering from the Sharif University of Technology, Iran, in 2015 and 2020, respectively. From 2022 to 2023, she was a Postdoctoral Researcher with the David R. Cheriton School of Computer Science, University of Waterloo, Canada. She is currently with Ericsson R&D, Canada.

Bo Sun received the B.E. degree from the Harbin Institute of Technology, Harbin, China, in 2013, and the Ph.D. degree from the Hong Kong University of Science and Technology, Hong Kong, in 2018. He is a Postdoctoral Fellow with the University of Waterloo. His research focuses on online optimization and decision-making under uncertainty with applications to real-world networked systems.

Mohammad A. Salahuddin (Member, IEEE) received the Ph.D. degree in computer science from Western Michigan University in 2014. He is currently a Research Assistant Professor of computer science with the University of Waterloo. His research interests include the Internet of Things, content delivery networks, network softwarization, network security, and cognitive network management. His coauthored research publications have received numerous awards, including the ACM SIGMETRICS Best Student Paper, the IEEE/IFIP NOMS Best Papers, and the IEEE CNOM Best Paper. He serves on the TPC for international conferences. He is a Reviewer for various peer-reviewed journals, magazines, and conferences.

Raouf Boutaba (Fellow, IEEE) received the M.Sc. and Ph.D. degrees in computer science from Sorbonne University in 1990 and 1994, respectively. He is currently a University Professor and the Director with the David R. Cheriton School of Computer Science, University of Waterloo, Canada. He also holds the Rogers Chair in network automation. He is the Founding Editor-in-Chief of the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT from 2007 to 2010 and served as an Editor-in-Chief of IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS from 2018 to 2021. He is a Fellow of the Engineering Institute of Canada, the Canadian Academy of Engineering, and the Royal Society of Canada.

Aladdin Saleh received the Ph.D. degree in electrical and electronic engineering and the M.B.A. degree in international management from the university of London, U.K. He is currently priming research and innovation activities with Rogers communications, including the joint research partnership with the University of Waterloo on 5G and emerging technologies. He has over 20 years of industry experience in Mobile Telecom, Canada. He taught and conducted research on next-generation wireless networks at several universities as a Full-Time Professor, an Adjunct Professor, and a Visiting Researcher. In addition to his role at Rogers, he is currently an Adjunct Professor with the Cheriton School of Computer Science, University of Waterloo.