Fast and Scalable Network Slicing by Integrating Deep Learning with Lagrangian Methods

Tianlun Hu*§, Qi Liao*, Qiang Liu†, Antonio Massaro‡ and Georg Carle§
*Nokia Bell Labs, Stuttgart, Germany
†University of Nebraska Lincoln, United States
‡Nokia Bell Labs, Paris, France
§Technical University of Munich, Germany
Email: *§tianlun.hu@nokia.com, *qi.liao@nokia-bell-labs.com, †qiang.liu@unl.edu,
‡antonio.massaro@nokia-bell-labs.com, §carle@net.in.tum.de

GLOBECOM 2023 - 2023 IEEE Global Communications Conference | 979-8-3503-1090-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/GLOBECOM54140.2023.10436849

***Abstract*—Network slicing is a key technique in 5G and beyond for efficiently supporting diverse services. Many network slicing solutions rely on deep learning to manage complex and high-dimensional resource allocation problems. However, deep learning models suffer limited generalization and adaptability to dynamic slicing configurations. In this paper, we propose a novel framework that integrates constrained optimization methods and deep learning models, resulting in strong generalization and superior approximation capability. Based on the proposed framework, we design a new neural-assisted algorithm to allocate radio resources to slices to maximize the network utility under inter-slice resource constraints. The algorithm exhibits high scalability, accommodating varying numbers of slices and slice configurations with ease. We implement the proposed solution in a system-level network simulator and evaluate its performance extensively by comparing it to state-of-the-art solutions including deep reinforcement learning approaches. The numerical results show that our solution obtains near-optimal quality-of-service satisfaction and promising generalization performance under different network slicing scenarios.**

I. Introduction

Network slicing has been widely investigated in 5G and beyond to support different network services in terms of cost-efficiency, flexibility, and assurance [1]. The ever-increasingly disaggregated network elements with fine-grained controllability may lead to volatile network dynamics in various aspects, e.g., admission and departure of slices in small time scales [2]. As a result, allocating radio resources to dynamic network slices becomes even more challenging.

The problem of resource allocation in network slicing has been extensively studied in the scenario of individual cells, where allocations are mostly optimized under the assumption that the resource demand of slices is known in advance. Existing works derive their solutions by formulating analytical closed-form models and solving the network slicing problem using constrained nonlinear optimization methods. In [3], the authors initially formulated and streamlined the slice-wise resource allocation problem by finding the upper and lower bound of network utility using the Lagrangian method. Subsequently, a sub-optimal solution was obtained using a greedy algorithm. Although the derived simplified model is effective, it is still tailored to specific slice configurations. In [4], authors proposed a flexible slice deployment solution with dynamic slice configurations, which formulated a slice model with adjustable parameters and solved resource partition with an optimization process. However, recent findings [5], [6] show that these approximated models cannot accurately represent diverse demand and performance of slices.

With recent advances in machine learning, reinforcement learning (RL) methods have been increasingly explored to tackle complex allocation problems in dynamic mobile networks. Zhou *et al.* [7] designed a multi-agent RL framework based on Q-Learning to determine the optimal joint resource allocation by using a coordinated Q-table, which alleviates the inter-slice resource constraints. However, this solution cannot scale to large state and action spaces. Our previous work [8] investigated coordinated multi-agent deep reinforcement learning (DRL) to handle the high-dimensional continuous action space and complex resource optimization in network slicing, where the inter-slice resource constraints are embedded in the designed architecture of neural networks. However, the proposed solution was explicitly trained for a fixed network scenario, and can hardly be generalized for different slice setups in terms of slice type and slice number. Liu *et al.* [9] introduced an approach to combine DRL and optimization process for slicing resource allocation in a single cell scenario. Yet, it also lacks the discussion of generalizing the solution to different multi-cell network scenarios with flexible slice setups.

In this paper, we present a novel algorithm, called integrated deep learning and Lagrangian method (IDLA), that optimizes slicing resource allocation and can be generalized to adapt to arbitrary slice combinations under time-varying dynamics. The main contributions of this work are listed as follows:

- We propose a novel framework that integrates deep learning models (that have approximation capability) and constrained optimization methods (that have strong generalization) and can generalize to arbitrary slice combinations.
- First, we derive a general deep neural network (DNN) model to approximate the slice network utility, that is capable of handling slices under different requirements.
- Then, by leveraging the efficient computation of the partial

This work was supported by the German Federal Ministry of Education and Research (BMBF) project 6G-ANNA.

Qiang Liu†'s work is partially supported by the US National Science Foundation under Grant No. 2212050.

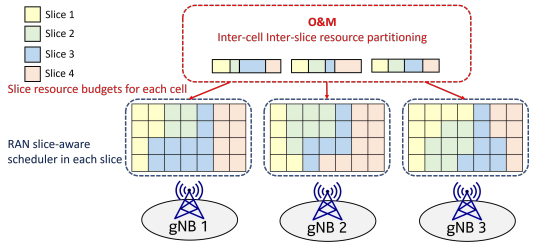

Figure 1: Dynamic slicing resource partition

derivatives of the slice utility function approximated by the DNN model, we design a Lagrangian method for resolving the optimal resource allocation on a per-slice basis, while adhering to inter-slice resource constraints.

- We evaluate the proposed algorithm in a system-level network simulator, where numerical results show that our algorithm obtains near-optimal quality of service (QoS) satisfaction and promising generalization performance as compared to the state-of-the-art solutions including the widely used DRL approaches.

This paper is organized as follows. We define the system model in Section II and formulate the slice-aware resource allocation problem in Section III. In Section IV we propose the solutions with DNN-based slice utility estimator and constrained nonlinear optimization. The numerical results are shown in Section V. We conclude this paper in Section VI.

II. System Model

We consider a discrete-time network system that comprises a set of cells denoted by $\mathcal{C} := \{1, 2, ..., C\}$. The set of slices in each cell $c \in \mathcal{C}$ can be time-varying, denoted by $\mathcal{S}_c(t) := \{1, 2, ..., S_c(t)\}$, where $S_c(t)$ is the number of slices served by cell c at time slot $t \in \mathbb{N}_0$. Each slice $s \in \mathcal{S}_c(t)$ in cell c needs to meet the predefined QoS requirements, e.g., throughput and delay requirements denoted by ϕ_s^* and d_s^* respectively. Note that although here the slices are defined by throughput and delay requirements, the problem formulation and the proposed approach in the following sections can be generalized to a broader set of requirements.

As illustrated in Fig. 1, Network Operation and Maintenance (O&M) dynamically partitions the inter-slice resource to provide per-slice resource budgets to each cell periodically. Within each cell, the radio access network (RAN) scheduler allocates physical resource blocks (PRBs) to individual services, using the provided resource budgets as upper-bound constraints. The focus of this paper is to solve inter-slice resource partitioning problem in network O&M, because we aim to develop a general slicing resource partitioning solution compatible with RAN schedulers from different network providers. At each time slot t, O&M optimizes slicing resource partitioning $\mathbf{x}_c(t)$ for each cell c, i.e., the ratio of the radio resource to allocate to each slice, given by

$$\mathbf{x}_c(t) := \left[x_{c,1}(t), \ldots, x_{c,S_c(t)}(t)\right] \in \mathcal{X}_c(t), \ \forall c \in \mathcal{C}, \quad (1)$$

$$\text{where } \mathcal{X}_c(t) := \left\{ [0,1]^{S_c(t)} \middle| \sum_{s \in \mathcal{S}_c(t)} x_{c,s}(t) \leq 1 \right\}. \quad (2)$$

Let $\mathbf{x}(t) := [\mathbf{x}_1(t), \ldots, \mathbf{x}_C(t)]$ be a collection of the per-cell slicing partitioning, the performance of each slice $s \in \mathcal{S}_c(t)$ in cell $c \in \mathcal{C}$ at time t is measured by the QoS satisfaction level $r_{c,s}(\mathbf{x}(t))$, defined as

$$r_{c,s}(\mathbf{x}(t)) := \min\left\{ \frac{\phi_{c,s}(\mathbf{x}(t))}{\phi_s^*}, \frac{d_s^*}{d_{c,s}(\mathbf{x}(t))}, 1 \right\}, \quad (3)$$

where $\phi_{c,s}(\mathbf{x}(t))$ and $d_{c,s}(\mathbf{x}(t))$ are the throughput and delay associated with slice s at cell c at time slot t, respectively. This performance metric takes the minimum between throughput and delay and is upper bounded by 1, such that both requirements need to be met to achieve the satisfaction level of 1.

Remark 1. *Theoretically, due to the inter-cell and possibly inter-slice interference, the achievable throughput and delay not only depends on the locally allocated resource to its own slice and cell, but also the resource occupation of other slices in the neighboring cells. Thus, in* (3)*, the QoS metric $r_{c,s}$ is written as a function of the global slicing partitioning $\mathbf{x}(t)$.*

III. Problem Formulation

Our objective is to find an efficient and scalable solution to optimize the utility of QoS satisfaction over all slices and cells by optimizing slicing resource partitioning at each time slot. The per-slot optimization problem is formulated in Problem 1.

Problem 1 (Global Problem)**.**

$$\max_{\mathbf{x}(t)} \ U(\mathbf{r}(\mathbf{x}(t)))$$
$$\text{subject to } \mathbf{r}(\mathbf{x}(t)) := \left[r_{c,s}(\mathbf{x}(t)) : c \in \mathcal{C}, s \in \mathcal{S}_c(t)\right], \quad (4)$$
$$(1), (2), (3), \ \forall t.$$

Note that the utility function can be defined based on various system designs. For example, a common utility function to consider the fairness is the sum of the logarithmic function of the local performance metric:

$$U(\mathbf{r}(\mathbf{x}(t))) := \sum_{s \in \mathcal{S}_c(t), c \in \mathcal{C}} \log\left(r_{c,s}(\mathbf{x}(t)) + 1\right). \quad (5)$$

In this paper, we use (5) as an example of the utility function, however, by leveraging the superior approximation capability of deep learning, our proposed approach can be applied to a wide range of utility functions. The challenge of solving Problem 1 is multifaceted. Firstly, the utility function's complexity poses a challenge to function approximation, particularly due to limited measurements in O&M. In contrast to RAN, where user and channel feedback can be collected with fine time granularity (e.g., in milliseconds), O&M only collects averaged cell- and slice-level key performance indicators (KPIs) with a coarse granularity (e.g., in minutes). Consequently, deriving closed-form expressions becomes extremely challenging. Secondly, the flexible slice configurations and inter-slice constraints further complicate the problem, resulting in slow convergence and poor adaptability of deep learning-based approaches. Finally, O&M's high scalability demand, e.g., up to over 100k cells, makes it challenging to use either large global deep learning models or collaborative multi-agent local models that require extensive exploration to learn from scratch.

IV. Proposed Solutions

In this section, we propose IDLA algorithm, to address the aforementioned challenges. First, we design and train a DNN to approximate the per-slice utility function. Then, with the derived slice-based utility model, we decompose Problem 1 into distributed cell-based resource allocation problems with inter-slice resource constraints. This decomposition allows the IDLA algorithm to adapt to a flexible number of slices per cell. Next, we use the Lagrangian method to solve the constrained decomposed problem, where the partial derivatives can be efficiently computed based on the DNN-based utility model. Finally, by leveraging the automatic differentiation engine of deep learning libraries, we improve the efficiency of the Lagrangian method.

A. Slice-based QoS Prediction using DNN

The complexity of the global utility function in (5) is caused by the dependency of each local utility $U_{c,s}(r_{c,s}(\mathbf{x}(t)))$ on the global slice resource partition $\mathbf{x}(t)$. Our idea is to investigate whether each local per-slice QoS satisfaction level $r_{c,s}(t)$ can be approximated by a single general DNN $f_{\boldsymbol{\theta}}(x_{c,s}(t), \mathbf{z}_{c,s}(t)), \forall c, s$ based on the local observations only, including the allocated slice resource $x_{c,s}(t)$ and a set of slice-based KPIs $\mathbf{z}_{c,s}(t)$, i.e., to find

$$f_{\boldsymbol{\theta}}(x_{c,s}(t), \mathbf{z}_{c,s}(t)) \approx r_{c,s}(\mathbf{x}(t)), \forall s \in \mathcal{S}_c(t), c \in \mathcal{C}, \quad (6)$$

where the DNN is parameterized by $\boldsymbol{\theta}$.

Data Collection: The DNN model serves as a general slice-based QoS estimator, trained on slice-wise data collected from network KPIs of different cells and slice configurations. Such data are collected in network O&M periodically, e.g., every 15 minutes, as a standard practical setting. Based on the experts' prior knowledge, to predict the slice utility $r_{c,s}(t)$, computed with the achievable throughput $\phi_{c,s}(t)$ and latency $d_{c,s}(t)$, $\forall c, s$, the following network KPIs are highly correlated:

- Per-slice required throughput ϕ_s^* and required delay d_s^*;
- Per-slice PRB utilization ratio $p_{c,s}(t)$, defined as the ratio of the PRBs occupied by the slice, which can be seen as the input of the allocated resource $x_{c,s}(t)$. This is because, if resource $x_{c,s}(t) := p_{c,s}(t)$ was allocated to the slice, the corresponding achieved throughput and delay would be the same as $\phi_{c,s}(t)$ and $d_{c,s}(t)$, respectively;
- The previous H states of per-slice average number of active users $\mathbf{v}_{c,s}^{(H)}(t) := [v_{c,s}(t-H), \ldots, v_{c,s}(t-1)]$;
- The previous H states of per-slice average channel quality indicator (CQI) $\mathbf{q}_{c,s}^{(H)}(t) := [q_{c,s}(t-H), \ldots, q_{c,s}(t-1)]$.

Note that we collect multiple historical states of the average number of active users and CQI, in the hope that the historical slice states not only capture temporal correlation, but also reflect some hidden information extracted from the missing global states, e.g., experienced inter-cell and inter-slice interference. Also, we follow the realistic assumption that for model inference, the real-time $v_{c,s}(t)$ and $q_{c,s}(t)$ are unknown while only the previous states within $[t-H, t-1]$ are available.

Thus, a set of the local observations as the input samples during a time period $[1, T]$ is then denoted by:

$$\mathcal{X}_{t=1}^{T} := \{(x_{c,s}(t), \mathbf{z}_{c,s}(t)) : \text{ for } t = 1, \ldots, T, \forall c, s\}, \quad (7)$$

$$\text{where } \mathbf{z}_{c,s}(t) := [\mathbf{v}_{c,s}(t), \mathbf{q}_{c,s}(t), \phi_s^*, d_s^*] \in \mathbb{R}^{2H+2}, \quad (8)$$

while the set of output samples is denoted by:

$$\mathcal{Y}_{t=1}^{T} := \{r_{c,s}(t) : \text{ for } t = 1, \ldots, T, \forall c, s\}, \quad (9)$$

where the QoS satisfaction level $r_{c,s}(t)$ is computed by (3) based on the observed throughput $\phi_{c,s}(t)$ and delay $d_{c,s}(t)$.

Local Utility Approximation: We learn a general slice QoS estimator $f_{\boldsymbol{\theta}} : \mathbb{R}^{2H+3} \to \mathbb{R} : (x_{c,s}, \mathbf{z}_{c,s}) \mapsto r_{c,s}$, as defined in (6), such that the local utility in (5) can be approximated by:

$$U_{c,s}(r_{c,s}(\mathbf{x}(t))) \approx \log(f_{\boldsymbol{\theta}}(x_{c,s}(t), \mathbf{z}_{c,s}(t)) + 1). \quad (10)$$

With the collected data (7) and (9), we can train a multi-layer perceptron (MLP) with $[x_{c,s}(t), \mathbf{z}_{c,s}(t)] \in \mathcal{X}_{t=1}^{T}$ as inputs and $r_{c,s}(t) \in \mathcal{Y}_{t=1}^{T}$ as output. Because $f_{\boldsymbol{\theta}}(\cdot)$ is a general distributed model that can apply to any slice, at each time t the samples from any cell and slice can contribute to model training, leading to a much higher sample efficiency and faster learning speed than training a large number of local models for distinct cells and slices. Moreover, a general model, that includes the throughput and delay requirements into the input features, can handle flexible slice configurations, even with unseen requirements.

Remark 2. *More details of data augmentation for unseen samples of different slice configurations will be given in Section V-B. Moreover, in Section V-B we validate the viability of learning* (6) *not only on the simulated data, but also on a collected real dataset from a commercial LTE network.*

B. Lagrangian Method for Slicing Resource Partitioning

With DNN-based utility approximation (10) at hand, we can decompose Problem 1 into independent per-cell optimization problems with intra-cell and inter-slice resource constraints. For each cell $c \in \mathcal{C}$ at time t, optimization problem is written as:

Problem 2 (Decomposed Local Problem)**.**

$$\max_{\mathbf{x}_c} \; F(\mathbf{x}_c) := \sum_{s \in \mathcal{S}_c(t)} \log\Big(f_{\boldsymbol{\theta}}\big(x_{c,s}(t), \hat{\mathbf{z}}_{c,s}(t)\big) + 1\Big)$$
$$\text{subject to } (1), (2), \; \forall t, \forall c \in \mathcal{C}, \quad (11)$$

where $\hat{\mathbf{z}}_{c,s}(t)$ *is the local observations defined in* (8)*.*

Problem 2 is a classical constrained non-linear optimization problem. Note that the objective in (11) is a monotonic non-decreasing function over $\mathbf{x} \in \mathbb{R}_+$, i.e., we have $F(\mathbf{x}_c') \geq F(\mathbf{x}_c)$ if $\mathbf{x}_c' \geq \mathbf{x}_c$ (entrywise greater). Therefore, the optimal solution to the problem with the equality constraint is also an optimal solution to the original problem, and we can solve it by using the Lagrange multiplier method. Since the problem is independently formulated for each time slot t and cell $c \in \mathcal{C}$, hereafter in this subsection we omit the index of t for brevity.

For each cell $c \in \mathcal{C}$, the Lagrangian is given by:

$$\mathcal{L}(\mathbf{x}_c, \lambda_c) := \sum_{s \in \mathcal{S}_c} \log f_{\boldsymbol{\theta}}(x_{c,s}) + \lambda_c\big(1 - \sum_{s \in \mathcal{S}_c} x_{c,s}\big), \quad (12)$$

where $f_{\boldsymbol{\theta}}(x_{c,s}) := f_{\boldsymbol{\theta}}\big(x_{c,s}(t), \hat{\mathbf{z}}_{c,s}(t)\big)$ is the learned DNN in (11), and $\lambda_c \in \mathbb{R}^+$ is the real non-negative Lagrangian multiplier. Then, we can solve the primal and dual problems:

$$\mathbf{x}_c^*(\lambda_c) = \arg\max_{\mathbf{x}_c \in \mathbb{R}^+} \mathcal{L}(\mathbf{x}_c, \lambda_c), \tag{13}$$

$$\lambda_c^* = \arg\min_{\lambda_c \geq 0} \mathcal{L}(\mathbf{x}_c^*(\lambda_c), \lambda_c), \tag{14}$$

by computing the partial derivatives with respect to each variable and performing Gradient Descent (GD) iteratively:

$$\begin{cases} x_{c,s}^{(i+1)} := \left[x_{c,s}^{(i)} + \delta_x^{(i)} \cdot \dfrac{\partial \mathcal{L}_c\left(\mathbf{x}_c^{(i)}, \lambda_c^{(i)}\right)}{\partial x_{c,s}^{(i)}} \right]_+, \forall s \in \mathcal{S}_c \\ \lambda_c^{(i+1)} := \left[\lambda_c^{(i)} - \delta_\lambda^{(i)} \cdot \left(1 - \sum_{s \in \mathcal{S}_c} x_{c,s}^{(i+1)} \right) \right]_+, \end{cases} \tag{15}$$

where i is the index of iteration, δ_x and δ_λ are the positive updating rates of $x_{c,s}, \forall s \in \mathcal{S}_c$ and λ_c, respectively, and $[x]_+$ is equivalent to $\max\{x, 0\}$. The partial derivative of $\mathcal{L}_c$ with respect to $x_{c,s}$, $\forall s \in \mathcal{S}_c$ is given by:

$$\frac{\partial \mathcal{L}_c\left(\mathbf{x}_c^{(i)}, \lambda_c^{(i)}\right)}{\partial x_{c,s}^{(i)}} = \frac{1}{f_{\boldsymbol{\theta}}\left(x_{c,s}^{(i)}\right) + 1} \cdot \frac{\partial f_\theta\left(x_{c,s}^{(i)}\right)}{\partial x_{c,s}^{(i)}} - \lambda_c^{(i)}. \tag{16}$$

C. Efficient Implementation to Improve the Performance of Lagrange Multiplier Method

One major limitation of the Lagrangian methods is that, if the function is non-linear and non-convex, there might exist multiple solutions or folds on the functional surface, and searching on one path may easily get stuck in a local optima. To overcome this, we exploit the automatic differentiation engine of deep learning libraries, and design a robust search strategy.

By using the automatic differentiation module *torch.autograd* of PyTorch [10], we can efficiently compute the partial derivative of the trained function with respect to any input variables on tensors, e.g., the partial derivative $\partial f_\theta\left(x_{c,s}^{(i)}\right)/\partial x_{c,s}^{(i)}$ in (16). This allows fast parallel computing of multiple searching paths. Thus, we propose the following search strategy:

(1) Based on the assumption that the network states between two successive time steps change smoothly, we propose to initialize the starting points for the optimization of each time slot t with the optimized solution of the previous time slot $t-1$, i.e., $\mathbf{x}_c^{(0)}(t) := \mathbf{x}_c^*(t-1)$;

(2) To find a better (possibly local) optima, we take P neighboring points near $\mathbf{x}_c^{(0)}(t)$ and run the GD optimizations from all P initial points in parallel. After GD optimizations have finished, we select the best solution among them.

The proposed IDLA algorithm is summarized in Algorithm 1, where $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, $i^{(\max)}$, and η denote the normal distribution for taking neighboring points with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, the maximum iteration steps, and criterion for stopping iteration, respectively.

Algorithm 1 IDLA Algorithm

for $t \in \mathcal{T}$ and $c \in \mathcal{C}$ **do**
  $i \leftarrow 0$
  $\mathbf{x}_c^{(i)}(t) \leftarrow \begin{cases} \textit{default action}, & \text{if } t = 0 \\ \mathbf{x}_c^*(t-1), & \text{Otherwise} \end{cases}$
  Take P neighboring points as:
  $\mathbf{x}_{c_p}^{(i)}(t) := \mathbf{x}_c^{(i)}(t) + \boldsymbol{\epsilon}$, $p \in [1, ..., P]$ with $\boldsymbol{\epsilon} \in \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$
  Parallelly compute for all $p \in [1, ..., P]$:
  Initialize Lagrangian multiplier $\lambda_{c_p}^{(i)}$
  Initialize update rate $\delta_{x_p}^{(i)} > 0, \delta_{\lambda_p}^{(i)} > 0$
  while $i \leq i^{(\max)}$ and $\|\mathbf{x}_{c_p}^{(i)}(t) - \mathbf{x}_{c_p}^{(i-1)}(t)\| \geq \eta$ **do**
    Compute partial derivative with (16) $\forall s \in \mathcal{S}_c(t)$
    Update optimization variables and Lagrangian multipliers with (15)
    Decrease update rate $\delta_{x_p}^i, \delta_{\lambda_p}^i$
    i += 1
  end while
  $\mathbf{x}_{c_p}^*(t) \leftarrow \mathbf{x}_{c_p}^{(i)}(t)$
  Choose the best solution among all P points that provides the highest utility:
  $\mathbf{x}_c^*(t) := \arg\max_{\mathbf{x}_{c_p}^*(t)} \sum_{s \in \mathcal{S}_c(t)} \log\left(f_{\boldsymbol{\theta}}\left(x_{c_p,s}^*(t)\right)\right)$.
end for

V. Performance Evaluation

In this section, we evaluate the performance of the proposed algorithm by implementing it in a system-level network simulator [11], which can imitate real network systems well with configurable user mobility, and slicing services. We compare the real-time processing performance of the IDLA scheme with two state-of-art schemes including a cell-wise DRL scheme and a traffic-aware baseline that allocates resources proportionally to data traffic demand per slice. We also compare it with an oracle scheme obtained by brute force optimization with the theoretically optimal performance. In addition, we explore the flexibility of the IDLA algorithm when facing slice configuration changes and its transferability from sample-collecting network configurations to a new network configuration.

A. Network Setting

We built a network system consisting of 4 three-sector base stations with the operating frequency band of 2.6 GHz, i.e., $C = 12$ cells. We defined 4 types of services, where the slice combination $\mathcal{S}_c(t)$ can be configurable and time-varying. Each service has different requirement as average user throughput ϕ_s^* for $s = 1, 2, 3, 4$ defined as $\{2, 1, 1.5, 0.5\}$ MBit/s, respectively. respectively. All cells are provided with the same bandwidth of 20 MHz. In addition, to imitate the real user traffic, we apply a varying traffic mask $\tau_s(t) \in [0, 1]$, which is collected from a real network system, for each slice $s \in \mathcal{S}_c(t)$ to reflect the daily periodic pattern of per-slice user traffic. In Fig. 2 we present the first 200 steps of the traffic mask. In the experiments, each step corresponds to 15 minutes in real time.

B. Sample Collection and Network QoS Estimator Training

Before the training process of network QoS estimator $f_{\boldsymbol{\theta}}(\cdot)$, we collected the training samples from the built network scenario in the simulator following the pipeline introduced in section IV-A.

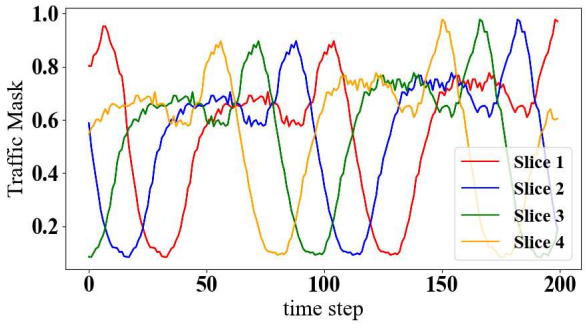

Figure 2: Traffic mask to imitate the dynamic slice traffic

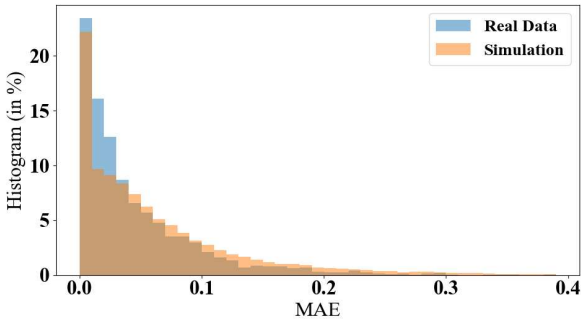

Figure 3: Network QoS estimator MAE histogram

Data Augmentation. We implemented a data augmentation strategy to cover a wider range of (unseen) sample space. The data augmentation strategy is summarized as follows:

(1) For the per-slice samples that achieve lower network QoS than the requirements, i.e., for $r_{c,s}(t) < 1$, we generated augmented samples by replacing the QoS requirements (ϕ_s^*, d_s^*) in the input training sample with the achieved QoS $(\phi_{c,s}(t), d_{c,s}(t))$ and replacing the QoS satisfaction $r_{c,s}(t)$ (sample output) with 1. Because, if the achieved $(\phi_{c,s}(t), d_{c,s}(t))$ were given as requirements, then the requirement would be met.

(2) Conversely, for the per-slice samples that achieve no less network QoS than the requirement, i.e., for $r_{c,s}(t) = 1$, we generated augmented samples by replacing the slice resource partition $x_{c,s}(t)$ in the input training samples with a random value $x'_{c,s}(t) \in [x_{c,s}(t), 1]$. Because the QoS is upper bounded by 1 based on (3), if more resource was given to the slice, due to the monotonicity of $r_{c,s}$ over $x_{c,s}$, the achieved $r_{c,s}$ would be 1 as well.

Model training. Then, for the training of QoS estimator $f_{\boldsymbol{\theta}}(\cdot)$, we built the DNN model with MLP architecture consisting of 4 hidden layers with the number of neurons $(36, 24, 16, 16)$. As proposed in IV-A, to better capture the temporal correlation, we used $h = 5$ steps of the historical network reports for estimator training, i.e., the training input $[x_{c,s}, \mathbf{z}_{c,s}] \in \mathbb{R}^{13}$. The training data was collected from the network environment defined in V-A. The model was trained for 200 epochs on 75% training samples and 25% testing samples with Adam optimizer with respect to mean absolute error (MAE) loss.

Fig. 3 shows the histogram of the MAE of network QoS estimator after training was completed. To validate the viability of training a utility estimator, We investigated the DNN model not only on the simulated data, but also on a dataset collected from a real commercial LTE network. By incorporating historical network reports, the estimator can provide accurate network QoS predictions based on the given slice resource partition. The average MAE was 0.0639 and 0.0573 for the simulation and real dataset respectively. The close performances indicate that our method is valid for handling real network systems.

C. Performance Comparison

With the derived slice QoS estimator $f_{\boldsymbol{\theta}}(\cdot)$ in hand, we further implement the optimization proposed in IV-C to obtain optimal resource partitions. We compare the performance of the following schemes:

- **IDLA** scheme: our proposed algorithm in Algorithm 1 with $P = 5$ neighboring start points, where the offset ϵ for each point follows a normal distribution $\mathcal{N}(0, 0.05)$.
- **DRL** scheme: a distributed Twin Delayed Deep Deterministic policy gradient (TD3) algorithm-based DRL approach similar to our previous work [8], which solves cell-wise optimal slicing resource partitions regarding the reward defined by minimum of network QoS (10) among all slices.
- **Traffic** scheme: a traffic-aware baseline that dynamically adapts slicing resource partitions in each cell proportionally to the current per-slice traffic amount, assuming perfect knowledge of traffic amount.
- **Oracle** scheme: an oracle scheme that provides the near-optimum for the constrained optimization problem. It is derived by using brute-force search for the optimal utility based on the pretrained $f_{\boldsymbol{\theta}}(\cdot)$ over all potential combinations of $\mathbf{x}_c \in \mathcal{X}_c$ with an interval of 0.05.

To evaluate the performance of IDLA against the other schemes, we implemented an online experiment in the network simulator with dynamic slice configuration, i.e., during the processing of the schemes, we changed the combination of network slices. For a fair comparison, we divide the whole online process into 3 time periods, denoted by $\mathcal{H}_0$, $\mathcal{H}_1$, and $\mathcal{H}_2$ respectively:

- $\mathcal{H}_0$ $(t \in [0, 1000))$: First, we set the network system with 3 slices with combination $\mathcal{S}_c(t) := [1, 2, 4], t \in \mathcal{H}_0, c \in \mathcal{C}$. Both **IDLA** and **Oracle** schemes are under the stage of sample collection, while **DRL** is under the exploration phase for buffer collection without agent training. The **Traffic** scheme provides slice resource partitioning proportional to instantaneous slice traffic demands.
- $\mathcal{H}_1$ $(t \in [1000, 3000))$: The network keeps the same slice configuration as $\mathcal{H}_0$. The **IDLA** and **Oracle** schemes optimize resource allocation based on the pre-trained $f_{\boldsymbol{\theta}}(\cdot)$ over the samples collected within $\mathcal{H}_0$, and **DRL** enter the phase of online training, with the samples collected within $\mathcal{H}_0$ also stored in the replay buffer.
- $\mathcal{H}_2$ $(t \in [3000, 5000])$: At $t = 3000$, the network slice configuration changes to slice combination $\mathcal{S}_c(t) := [1, 2, 3, 4], t \in \mathcal{H}_2, c \in \mathcal{C}$, i.e., we introduce a new slice

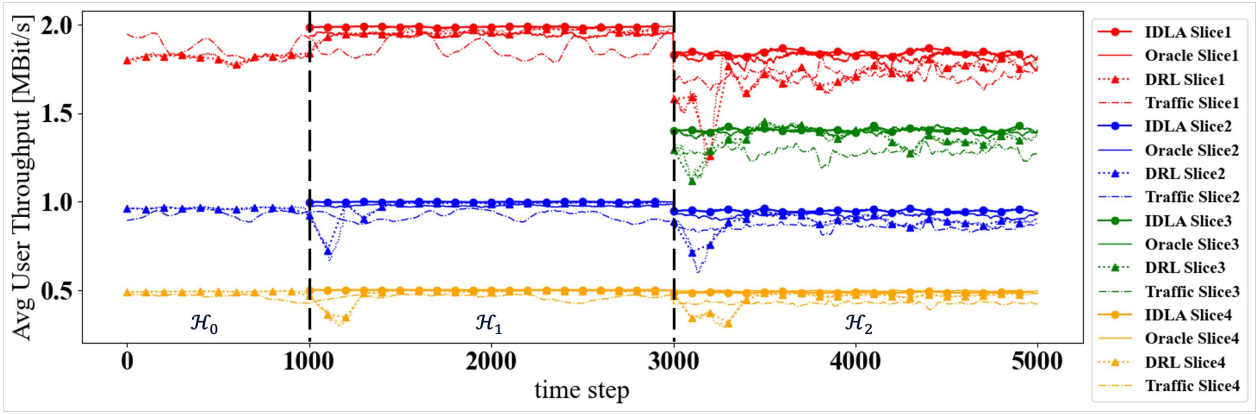

Figure 4: Comparison of average user throughput among schemes

with the corresponding user group into the network system with the same resource constraints.

In Fig. 4, we compare the averaged per-slice user throughput $\phi_{c,s}(t)$ referring to its requirements ϕ_s^* over all cells for $s \in \mathcal{S}_c(t)$ of all schemes during the entire process $\{\mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2\}$. Note that in $\mathcal{H}_0$ and $\mathcal{H}_1$, there are only 3 slices with index $[1, 2, 4]$, while later in period $\mathcal{H}_2$, we add a new slice with index 3. After the offline training of QoS estimator $f_{\boldsymbol{\theta}}(\cdot)$, the **IDLA** provides the best performance among all schemes and faster convergence than **DRL** scheme in both online processing phases $\mathcal{H}_1$ and $\mathcal{H}_2$. Moreover, **IDLA** quickly adapts to the new slice configuration (with an added slice) in $\mathcal{H}_2$ and provides robust performance. On the contrary, due to the poor scalability of the cell-wise agent, the **DRL** scheme needs to retrain the model when the slice configuration changes. It is worth noting that since the total network resource remains the same after adding a new slice, the user throughputs in other slices decrease correspondingly to serve the users in the new slice.

To compare the converged performance of all schemes, in Fig. 5 we compare the empirical cumulative distribution function (CDF) of the converged network QoS satisfaction level of all schemes under both slice configurations. The **IDLA** scheme provides the highest probability of QoS satisfaction under both slice configurations with 0.973 and 0.629 respectively, while **Oracle** and **DRL** schemes achieved similar converged QoS satisfaction. Note that, theoretically, with brute-force search **Oracle** scheme should find a near-optimal solution if the utility estimator can be learned with 100% accuracy. However, in this experiment, the performance of **Oracle** is not as good as **IDLA**, due to the estimation error of the utility estimator and the discretization of the searching grid space. In general, **IDLA** provides the best performance in terms of convergence rate, converged performance, and scalability.

VI. Conclusion

In this paper, we propose a novel framework to integrate the strong generalization of the Lagrangian method and the superior approximation capability of the deep learning. We developed IDLA algorithm to solve the resource partitioning problem in network slicing with assured inter-slice resource constraint. The results show that our proposed approach can provide near-optimal performance with fast convergence and high generality

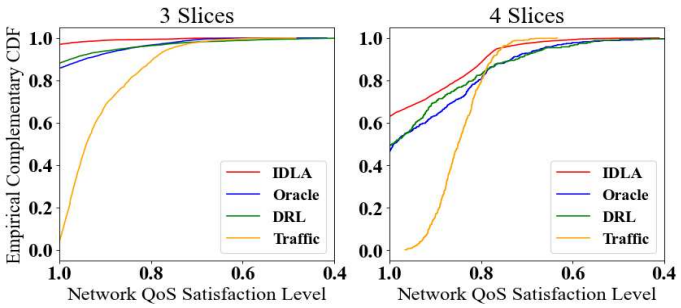

Figure 5: Comparison of network utility

in comparison with state-of-the-art solutions. In addition, we show the scalability of the proposed methods by deploying the derived model in different network scenarios with varying slicing configurations. With the slice-wise resource scheduler, our proposed algorithm provides high scalability and generality for fast and efficient deployment in real network systems.

References

[1] 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 *ACM CoNEXT*, 2018, pp. 353–365.

[2] S. D'Oro, L. Bonati, F. Restuccia, M. Polese, M. Zorzi, and T. Melodia, "Sl-EDGE: Network slicing at the edge," in *MobiHoc*, 2020, pp. 1–10.

[3] M. K. Motalleb, V. Shah-Mansouri, S. Parsaeefard, and O. L. A. López, "Resource allocation in an open RAN system using network slicing," *IEEE Transactions on Network and Service Management*, vol. 20, pp. 471–485, 2023.

[4] M. Leconte and et al., "A resource allocation framework for network slicing," *IEEE INFOCOM*, pp. 2177–2185, 2018.

[5] L. Qiang, C. Nakjung, and H. Tao, "Constraint-aware deep reinforcement learning for end-to-end resource orchestration in mobile networks," in *IEEE ICNP*, 2021, pp. 1–11.

[6] ——, "OnSlicing: Online end-to-end network slicing with reinforcement learning," in *ACM CoNEXT*, 2021, pp. 141–153.

[7] H. Zhou and et al., "RAN resource slicing in 5G using multi-agent correlated Q-learning," *IEEE PIMRC*, pp. 1179–1184, 2021.

[8] T. Hu, Q. Liao, Q. Liu, D. Wellington, and G. Carle, "Inter-cell slicing resource partitioning via coordinated multi-agent deep reinforcement learning," in *IEEE ICC*, 2022.

[9] Q. Liu, T. Han, N. Zhang, and Y. Wang, "Deepslicing: Deep reinforcement learning assisted resource allocation for network slicing," *IEEE GLOBECOM*, pp. 1–6, 2020.

[10] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NIPS Autodiff Workshop*, 2017.

[11] Nokia Siemens Networks, *White paper: Self-Organizing Network (SON): Introducing the Nokia Siemens networks SON suite-an efficient, future-proof platform for SON.* Technical report, October, 2009.