AdaSlicing: Adaptive Online Network Slicing under Continual Network Dynamics in Open Radio Access Networks

Ming Zhao, Yuru Zhang, Qiang Liu
University of Nebraska-Lincoln
qiang.liu@unl.edu

Ahan Kak, Nakjung Choi
Nokia Bell Labs
nakjung.choi@nokia-bell-labs.com

IEEE INFOCOM 2025 - IEEE Conference on Computer Communications | 979-8-3315-4305-1/25/$31.00 ©2025 IEEE | DOI: 10.1109/INFOCOM55648.2025.11044475

***Abstract*—Open radio access networks (e.g., O-RAN) facilitate fine-grained control (e.g., near-RT RIC) in next-generation networks, necessitating advanced AI/ML techniques in handling online resource orchestration in real-time. However, existing approaches can hardly adapt to time-evolving network dynamics in network slicing, leading to significant online performance degradation. In this paper, we propose *AdaSlicing*, a new adaptive network slicing system, to online learn to orchestrate virtual resources while efficiently adapting to continual network dynamics. The AdaSlicing system includes a new soft-isolated RAN virtualization framework and a novel AdaOrch algorithm. We design the AdaOrch algorithm by integrating AI/ML techniques (i.e., Bayesian learning agents) and optimization methods (i.e., the ADMM coordinator). We design the soft-isolated RAN virtualization to improve the virtual resource utilization of slices while assuring the isolation among virtual resources at runtime. We implement AdaSlicing on an O-RAN compliant network testbed by using OpenAirInterface RAN, Open5GS Core, and FlexRIC near-RT RIC, with Ettus USRP B210 SDR. With extensive network experiments, we demonstrate that AdaSlicing substantially outperforms state-of-the-art works with 64.2% cost reduction and 45.5% normalized performance improvement, which verifies its high adaptability, scalability, and assurance.**

***Index Terms*—Network Slicing, Open RAN, Network Autonomy, Emerging Applications**

I. Introduction

Network slicing is a key technique in 5G and Beyond to cost-efficiently and flexibly support emerging mobile applications, e.g., autonomous driving [1], extended reality [2], and Internet of Things [3]. With the advanced technology of infrastructure virtualization, multiple virtual networks (i.e., slices) can be concurrently instantiated and operated on common physical infrastructures (e.g., base stations and switches), with assured resource and performance isolation. By tailoring the parameters and resources of each slice, mobile network operators (MNOs) can effectively meet the diversified performance needs of slice tenants, such as end-to-end latency, security, and reliability. In network slicing, resource orchestration serves as the key role to dynamically manage virtualized resources [4], [5] in multiple technical domains (e.g., radio spectrum) to all slices for assuring their service-level agreements (SLAs).

Existing in-use orchestration solutions [6], [7], [8] heavily rely on human expertise throughout the life-cycle of each slice (e.g., performance modeling and fine-tuning), where orchestration actions are optimized in the coarse granularity, such as every hour. With the increasing momentum of open network initiatives (e.g., O-RAN [9], [10]) and an emphasis on a unified software approach to simplify operations amidst increasing complexity (e.g., UNEXT [11]), next-generation networks will expose high-dimensional states (e.g., thousands if not more) and allow nearly real-time control (e.g., subseconds), which enables more fine-grained orchestration in network slicing for further exploiting resource multiplexing [6]. To tackle the complex fine-grained orchestration problem, machine learning (ML) techniques [12] have been increasingly explored, such as deep reinforcement learning [13] and Bayesian learning [4], [5], and achieved great improvements, in terms of performance, autonomy, and scalability.

However, we found that existing works can hardly adapt to time-varying dynamics in network slicing, which can result in substantial performance degradation (e.g., violated SLAs and soared resource usage), especially during online resource orchestration. Generally, existing works rely on deep neural network (DNN)-parameterized agents to manage resource orchestration for all slices, where their fixed input and output space[1] limit the adaptability to diverse network dynamics. On the one hand, the active slices in the network are not stationary. Independent slice tenants may operate their slices dynamically, such as starting and stopping slices at different times throughout the day. On the other hand, the traffic pattern and application characteristics of each slice may change and evolve over time. As a result, existing DNN-parameterized agents have to be retrained and refined, leading to delayed adaptation to time-varying network dynamics.

In this paper, we propose a new adaptive network slicing system (*AdaSlicing*), to online learn while efficiently adapting to time-varying network dynamics. The key idea is to integrate AI/ML techniques and optimization methods during online resource orchestration. We design a new AdaOrch algorithm to minimize the total operating cost of supporting all slices, while assuring the performance requirements defined by their SLAs. On the one hand, we design a Bayesian learning agent to handle the resource orchestration for each slice, which will be continually updated with accumulated online experiences. On the other hand, we design a coordinator to coordinate all active slices in regard to the infrastructure capacity of virtual resources at runtime. Moreover, we design a new soft-isolated RAN virtualization to improve the virtual resource utilization of slices while assuring the isolation among virtual resources at runtime. We implemented AdaSlicing on an end-to-end O-RAN compliant network testbed by using OpenAirInterface RAN, Open5GS Core, and FlexRIC near-RT RIC, with Ettus USRP B210 SDR. With the extensive network experiments, we

[1]Although there are several DNN architectures with flexible inputs (e.g., LSTM and GNN), their training complexity and sample efficiency are widely concerned to be used for online network management [4], [14].

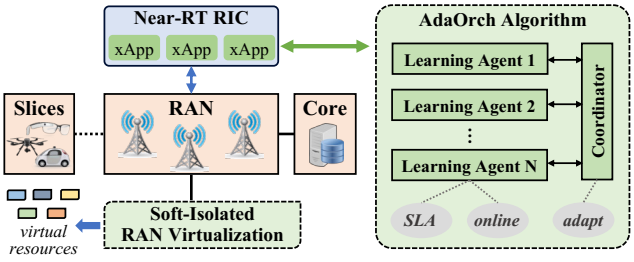

Fig. 1. The overview of AdaSlicing.

demonstrate that AdaSlicing can reduce 64.2% total operating cost while improving 45.5% normalized performance of slices, as compared to state-of-the-art solutions.

Overall, we propose *AdaSlicing*, a new adaptive online network slicing system, that can flexibly adapt to diverse time-varying network dynamics. The detailed contributions are:

- We design a new soft-isolated RAN virtualization framework that substantially improves virtual resource utilization of slices at runtime.
- We design a new AdaOrch algorithm that online learns and orchestrates virtual resources for slices with assured SLAs.
- We implement the AdaSlicing system on an O-RAN compliant mobile network testbed, with multiple slices and users.
- We conducted extensive experiments to evaluate AdaSlicing, in terms of adaptability and scalability.

II. *AdaSlicing* Overview

In Fig. 1, we overview the AdaSlicing system under the architecture of open radio access networks. It includes multiple slices, radio access network (RAN), and core network (CN), where the near-RT RIC hosts a wide range of xApps, such as performance monitoring xApps and the AdaOrch algorithm xApp. In AdaSlicing, two primary components are the soft-isolated RAN virtualization and the AdaOrch algorithm.

We design the *soft-isolated RAN virtualization* to improve the utilization of virtual resources at runtime while assuring the isolation among virtual resources (See Sec. V). Different from existing hard-isolated RAN virtualization, our key idea is to enable the sharing of unused virtual resources among slices, before the virtual-to-physical mapping, at runtime. We create two new kinds of virtual resources, including the soft-isolated virtual resource block (svRB) and sharing weight (SW), which can be dynamically orchestrated to all slices by the AdaOrch algorithm. Note that all unused virtual resources are shared proportionally according to the SW value of slices, which improves resource utilization at runtime and creates performance interdependence among all slices. In other words, in addition to the svRB, the shared virtual resources of a slice from the sparse vRB pool depend on not only its SW value but also the SW values of all other active slices.

We design the *AdaOrch algorithm* to online orchestrate virtual resources for all slices under time-varying network dynamics (See Sec. IV). Different from existing orchestration solutions, our key idea is to integrate AI/ML techniques and optimization methods to improve adaptability while maintaining the autonomy of the AdaOrch algorithm. Specifically, it includes multiple learning agents, where each agent corresponds to online learning and orchestrates virtual resources for a slice. Note that, the AdaOrch algorithm can support heterogeneous learning agents, in terms of adopted AI/ML techniques (e.g., Bayesian learning and multi-armed bandit), as long as they match the same input/output space and also follow necessary training processes. It also includes a coordinator, designed based on the alternating direction method of multipliers (ADMM), to coordinate the infrastructure capacity of virtual resources for all slices. During online orchestration, each learning agent observes its local network context and makes the orchestration action for its corresponding slice to meet the performance requirement defined by the slice SLA. At runtime, only active slices are involved and iteratively communicated with the coordinator to achieve a consensus of resource capacity. On the one hand, these learning agents will be online updated according to newly obtained online experiences, which assures the continual learning capability to adapt to potential intra-slice network dynamics. On the other hand, the coordinator can support an arbitrary number of learning agents during each orchestration slot, which can flexibly adapt to possible inter-slice network dynamics.

III. System Model

We consider an O-RAN compliant mobile network, including the core network (CN) and radio access network (RAN) with multiple base stations (BSs)[2] and network slices. Slice tenants establish the service level agreement (SLA) with the mobile network operator (MNO) to support their slice users, with predefined performance requirements, such as the maximum latency and minimum throughput. The MNO creates multiple xApps in the near-RT RIC to dynamically orchestrate the virtual resources for all slices in the fine time granularity (e.g., every second). Here, we focus on two kinds of virtual resources: 1) the soft-isolated virtual resource blocks (svRB), and 2) the sharing weight (SW), which are detailed in Sec. V. We denote $\mathcal{I}$ as the set of network slices, where $i \in \mathcal{I}$ denotes the i-th slice.

Action Space. Denote $x_i^{(t)}$ and $w_i^{(t)}$ as the number of svRBs and the SW value of the i-th slice at the tth time slot, respectively. For the sake of simplicity, we further denote $\mathcal{X}^{(t)} = \{x_i^{(t)}, \forall i \in \mathcal{I}\}$ and $\mathcal{W}^{(t)} = \{w_i^{(t)}, \forall i \in \mathcal{I}\}$ as the set of these orchestration actions, respectively. Therefore, we define the action space of resource orchestration to the i-th slice as

$$A_i^{(t)} = \{x_i^{(t)}, w_i^{(t)}\}. \tag{1}$$

Performance Model. Under the soft-isolated RAN virtualization (see Sec. V), the final experienced vRBs of a slice depend on not only its orchestration action but also that of other slices (particularly their SW values). Hence, we define the performance function of the i-th slice as

$$P_i^{(t)} = f(x_i^{(t)}, w_i^{(t)} | s_i^{(t)}), \tag{2}$$

where we introduce $s_i^{(t)} = \sum_{j \in \mathcal{I}, j \neq i} w_j^{(t)}$ as the aggregated SW value of all other active slices. Here, the performance metrics of each slice can be multi-dimensional and heterogeneous,

[2]Without loss of generality, we focus on the resource orchestration problem in a single base station, where the AdaOrch algorithm can be easily extended to support the scenario of multiple base stations.

such as end-to-end latency and reliability. Here, more network context might be incorporated, if relevant and needed, to better represent the performance function of slices.

Cost Model. From the MNO perspective, the operating cost of running a slice highly depends on its usage of virtual resources. Here, we define operating cost of the i-th slice as

$$U_i^{(t)} = U_H \cdot x_i^{(t)} + U_S \cdot w_i^{(t)}, \tag{3}$$

where U_H and U_S denote the unit cost of svRB and SW, respectively.

Problem. The objective is to minimize the total cost of supporting all slices, while meeting their performance requirements defined by slice SLAs. Therefore, we formulate the resource orchestration problem in network slicing as

$$\mathcal{P}_0: \quad \min_{\{\mathcal{X}^{(t)}, \mathcal{W}^{(t)}\}} \quad \sum\nolimits_{i\in\mathcal{I}} U_i^{(t)} \tag{4}$$

$$s.t. \quad C1: \quad f(x_i^{(t)}, w_i^{(t)} | s_i^{(t)}) \geq Q_i, \forall i \in \mathcal{I}, \tag{5}$$

$$C2: \quad 0 \leq w_i^{(t)} \leq 1, \forall i \in \mathcal{I}, \tag{6}$$

$$C3: \quad 0 \leq x_i^{(t)} \leq H, \forall i \in \mathcal{I}, \tag{7}$$

$$C4: \quad 0 \leq \sum\nolimits_{i\in\mathcal{I}} x_i^{(t)} \leq H. \tag{8}$$

Here, the constraint $C1$ assures that the performance of each slice can be maintained, where the performance threshold of the i-th slice is denoted as Q_i. The constraint $C2$ and $C3$ define the boundary of the action space, including the maximum number of svRBs. Moreover, the constraint $C4$ assures that the infrastructure capacity (denoted by H) will not be exceeded at runtime.

Challenge. The challenge of addressing the above problem is mainly two-fold. On the one hand, the multi-dim performance of slices relates to a wide range of factors (e.g., traffic pattern and channel quality) and can hardly be represented in closed-form with respect to resource orchestration actions. Moreover, the performance function of slices could change over time, depending on their time-evolving application characteristics. As a result, it is difficult to use conventional math modeling approaches to represent the complex and time-evolving performance function. On the other hand, the active slices are non-stationary, depending on the operation strategy of independent slice tenants, such as starting and stopping slices at different times throughout the day. As a result, it is inefficient to handle the dynamics of active slices by using only parameterized agents with fixed input and output spaces.

IV. Proposed Solution

In this section, we develop the AdaOrch algorithm to efficiently solve the resource orchestration problem. The fundamental idea is to integrate AI/ML techniques and optimization methods to address the aforementioned adaptability challenge. On the one hand, we design a learning agent (based on the Bayesian optimization framework) that focuses on the resource orchestration of each slice, which will be online updated with accumulated experiences to track its potentially time-evolving performance function. On the other hand, we design a coordinator (based on the alternating direction method of multipliers (ADMM) framework) to coordinate all active slices in regard to the infrastructure capacity of virtual resources. In addition, we design the interface (e.g., state and action space and training mechanism) to enable the convergent interaction among learning agents and the coordinator towards the objective of adaptive network slicing.

Specifically, we first decouple the original problem $\mathcal{P}_0$ into multiple subproblems and a coordination problem, by leveraging the ADMM framework. We design each subproblem to correspond to the resource orchestration of individual slices, and the coordination problem to coordinate and assure the infrastructure capacity of virtual resources (i.e., $C4$). The coordination problem turns out to be convex and can be efficiently solved by off-the-shelf optimization toolboxs. To solve individual subproblems, we design a constrained Bayesian optimization method to online learn and orchestrate virtual resources for individual slices, while assuring their performance requirements. During online orchestration, these subproblems and the coordination problem will be solved alternatively, and eventually achieve a convergent optima, which acts as the final resource orchestration for all slices.

A. Problem Decomposition

First, we introduce auxiliary variables $z_i^{(t)}$ and enforce additional constraints by letting $z_i^{(t)} = x_i^{(t)}, \forall i \in \mathcal{I}$. Then, we can reformulate the problem $\mathcal{P}_0$ into the following problem

$$\mathcal{P}_1: \quad \min_{\{\mathcal{X}^{(t)}, \mathcal{Z}^{(t)}, \mathcal{W}^{(t)}\}} \quad \sum\nolimits_{i\in\mathcal{I}} U_i^{(t)} \tag{9}$$

$$s.t. \quad C1,\ C2,\ C3, \tag{10}$$

$$C4: \quad 0 \leq \sum\nolimits_{i\in\mathcal{I}} z_i^{(t)} \leq H, \tag{11}$$

$$C5: \quad x_i^{(t)} = z_i^{(t)}, \forall i \in \mathcal{I}, \tag{12}$$

where we rewrite the constraint $C4$ in $\mathcal{P}_0$ (which relates to $\mathcal{X}^{(t)}$) into a new constraint in this problem $\mathcal{P}_1$, which relates to only $\mathcal{Z}^{(t)}$. Here, we denote $\mathcal{Z}^{(t)} = \{z_i^{(t)}, \forall i \in \mathcal{I}\}$ as the set of auxiliary variables. The above problem reformulation aims to decouple the connection between the optimization variables of $\mathcal{X}^{(t)}$ in the original constraint $C4$ in $\mathcal{P}_0$, which will facilitate the separation of individual $x_i^{(t)}$ in each slice. Note that, the optimization variable $w_i^{(t)}$ in each slice is not included by any constraints, except its boundary between 0 and 1. In the reformulated problem $\mathcal{P}_1$, we have three kinds of optimization variables, i.e., $\mathcal{X}^{(t)}$, $\mathcal{Z}^{(t)}$, and $\mathcal{W}^{(t)}$.

Second, we derive the augmented Lagrangian function of $\mathcal{P}_1$ with scaled dual variables as

$$\mathcal{L}(\mathcal{X}, \mathcal{W}, \mathcal{Z}, \mathcal{Y}) = \sum_{i\in\mathcal{I}} (U_i^{(t)} + \frac{\rho}{2} \left\| x_i^{(t)} - z_i^{(t)} + y_i^{(t)} \right\|^2), \tag{13}$$

where ρ is a positive constant, and $\mathcal{Y}^{(t)} = \{y_i^{(t)}, \forall i \in \mathcal{I}\}$ is the set of scaled dual variables. Based on the ADMM framework [15], we can solve the problem $\mathcal{P}_1$ by alternatively addressing the following problems:

$$\mathcal{P}_2: \quad \mathcal{X}^{(t+1)}, \mathcal{W}^{(t+1)} = \arg \min_{\mathcal{X}^{(t)}, \mathcal{W}^{(t)} \in \{C_1, C_2, C_3\}} \mathcal{L}(\mathcal{X}, \mathcal{W}, \mathcal{Z}^{(t)}, \mathcal{Y}^{(t)}), \tag{14}$$

$$\mathcal{P}_3: \quad \mathcal{Z}^{(t+1)} = \arg \min_{\mathcal{Z}^{(t)} \in C_4} \mathcal{L}(\mathcal{X}^{(t+1)}, \mathcal{W}^{(t+1)}, \mathcal{Z}, \mathcal{Y}^{(t)}), \tag{15}$$

and updating the dual variables as follows:

$$y_i^{(t+1)} = y_i^{(t)} + (x_i^{(t+1)} - z_i^{(t+1)}), \forall i \in \mathcal{I}. \quad (16)$$

Here, problem $\mathcal{P}_2$ involves the actual resource orchestration for all slices, under the given auxiliary and dual variables in the last iteration. Then, problem $\mathcal{P}_3$ relates to the update of auxiliary variables, depending on the optimized orchestration actions of problem $\mathcal{P}_2$. Next, dual variables can be updated with the optimized orchestration actions and auxiliary variables. This iteration continues until the convergence of variables.

B. The Design of Coordinator

In the context of the AdaSlicing system, we centralize the solving of problem $\mathcal{P}_3$ and the update of dual variables into the coordinator. Specifically, we rewrite the problem $\mathcal{P}_3$ as

$$\mathcal{P}_4: \quad \min_{z_i^{(t)}} \quad \sum\nolimits_{i\in\mathcal{I}} \left\| x_i^{(t)} - z_i^{(t)} + y_i^{(t)} \right\|^2 \quad (17)$$

$$s.t. \qquad 0 \le \sum\nolimits_{i\in\mathcal{I}} z_i^{(t)} \le H, \quad (18)$$

where the first part of augmented Lagrangian is irrelevant to this optimization of auxiliary variables and thus omitted. We observe that this problem $\mathcal{P}_4$ is a standard quadratic integer programming problem, which is convex. Hence, we can utilize the off-the-shelf optimization toolbox to efficient solve it, such as CVX [16], [17]. By solving problem $\mathcal{P}_4$, we obtain the updated auxiliary variables $\mathcal{Z}^{(t+1)}$, and then update the dual variables accordingly.

C. The Design of Learning Agents

In the context of the AdaSlicing system, we solve the problem $\mathcal{P}_2$ in these learning agents. This is based on the observation that, the problem $\mathcal{P}_2$ is fully separable with respect to each slice, where the optimization of $x_i^{(t)}$ can be fully conducted in the i-th slice without any connections with other slices. For the sake of simplicity, we re-express the problem $\mathcal{P}_2$ into the following child problem $\mathcal{P}_5$ in the ith slice

$$\mathcal{P}_5: \quad \min_{\{x_i^{(t)}, w_i^{(t)}\}} \quad U_i^{(t)} + \frac{\rho}{2} \left\| x_i^{(t)} - z_i^{(t)} + y_i^{(t)} \right\|^2 \quad (19)$$

$$s.t. \quad \bar{C}1: \quad f(x_i^{(t)}, w_i^{(t)} | s_i^{(t)}) \ge Q_i, \quad (20)$$

$$\bar{C}2: \quad 0 \le w_i^{(t)} \le 1, \quad (21)$$

$$\bar{C}3: \quad 0 \le x_i^{(t)} \le H, \quad (22)$$

where all constraints are rewritten with respect to only the ith slice. Due to the unknown and potential time-evolving performance function in constraint $\bar{C}1$, it is challenging to solve the problem with conventional optimization based methods, such as linear and convex optimization [18].

Constrained Bayesian Optimization. Here, we design a new constrained Bayesian optimization method to address the above child problem $\mathcal{P}_5$. Bayesian optimization [19], [20] is the state-of-the-art global optimization framework, and is particularly efficient in handling blackbox problems with expensive querying costs. It is an iterative searching process, including a probabilistic surrogate model and an acquisition function. In each iteration, the surrogate model, e.g., Gaussian process [21], is trained to approximate the uncertainty of the black-box function, e.g., the performance function of slices $f(x_i^{(t)}, w_i^{(t)} | s_i^{(t)})$, according to previous experiences, e.g., orchestration-to-performance pairs. Then, the acquisition function, e.g., expected improvement (EI) [22], estimates the utility of different actions $\{x_i^{(t)}, w_i^{(t)}\}$ while balancing the exploration and exploitation. The next action can be determined by maximizing the acquisition function, under the boundary of action space (e.g., $\bar{C}2, \bar{C}3$). Along with the accumulation of online experiences, the surrogate model will be updated to be more accurate to represent the blackbox function, which guides the selection of future orchestration actions towards the optima.

Barrier Method. In the child problem $\mathcal{P}_5$, its objective function is deterministic with a closed-form expression, while its constraint $\bar{C}1$ is the unknown blackbox function. As vanilla Bayesian optimization hardly handle complex constraints (i.e., $\bar{C}1$), we use the barrier method [18] to convert the child problem $\mathcal{P}_5$ into unconstrained one. Specifically, we add a penalty function to the objective function, and rewrite the child problem $\mathcal{P}_5$ as

$$\mathcal{P}_6: \quad \min_{\{x_i^{(t)}, w_i^{(t)}, s_i^{(t)}\}} \quad U_i^{(t)} + \frac{\rho}{2} \left\| x_i^{(t)} - z_i^{(t)} + y_i^{(t)} \right\|^2 + \phi_i$$

$$s.t. \quad \bar{C}2, \bar{C}3, \quad (23)$$

where $\phi_i = -log(-(Q_i - f(x_i^{(t)}, w_i^{(t)} | s_i^{(t)}))$ is the log barrier penalty.

Gaussian Process. We adopt Gaussian Process (GP) as the surrogate model to approximate the time-evolving performance function of the slice, which is based on its superior advantages of sample efficiency and robustness. Gaussian Process [23] is a probabilistic, sample-efficient and wide-adopted non-parametric machine learning technique. With slight abuse of notation, we denote $\mathbf{x} = [x_i^{(t)}, w_i^{(t)}, s_i^{(t)}]$ as the combination of the orchestration action of this slice and aggregated SW of other slices. Gaussian Process can be defined by its mean function $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x}')$ as:

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})], \quad (24)$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))], \quad (25)$$

where $f(\mathbf{x})$ represents the unknown performance function of the slice to learn (i.e., $f(x_i^{(t)}, w_i^{(t)} | s_i^{(t)})$). Here, k is the kernel function, which determines the smoothness and other properties of the functions drawn from the Gaussian Process. For any new input $\mathbf{x}^*$, GP can generate in a normal distribution, with its mean denoted by $\mu(\mathbf{x}^*)$ and its variance by $\sigma^2(\mathbf{x}^*)$, which will be utilized by the acquisition function when selecting the next action.

To track the potential time-evolving performance function of slices, we introduce a fixed-size reply buffer for the GP during the online orchestration. Instead of using all observed experiences, we regress the GP by sampling experiences from the reply buffer, based on the priority of experiences. When a new online experience is obtained, it will be queued into the reply buffer with the highest sampling priority. Note that, we decay the priority of an experience according to its age-of-information, which will make the latest experiences to be

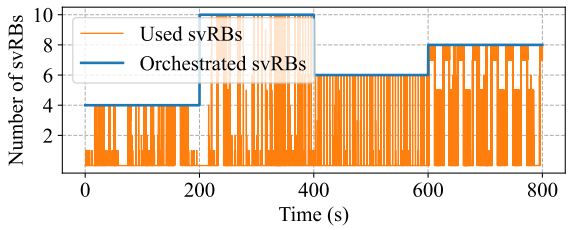

Fig. 2. An example of runtime utilization of vRBs under different applications. Here, we run mixed applications before 400s, only watch live video in [400s, 600s], and then perform speedtest.

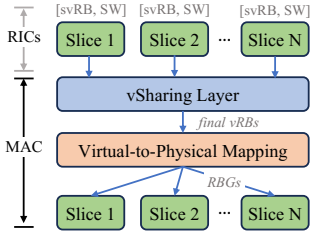

Fig. 3. The architecture of soft-isolated RAN virtualization.

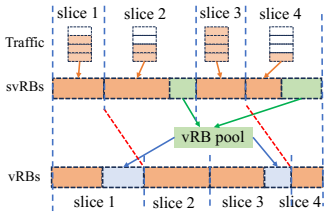

Fig. 4. An example of soft-isolated RAN virtualization.

sampled more frequently. If the reply buffer is full, the oldest experience will be removed, which helps to track the time-evolving performance function of slices. In the meantime, the fixed size of the reply buffer ensures that the computation complexity of GP regression is constant and can be performed in real-time.

Acquisition Function. For the acquisition function, there is a wide range of candidates, such as lower confidence bound (LCB), expected improvement (EI), and probability of improvement (PI). For example, EI aims to maximize the expected improvement under to-date observations, while balancing the exploration and exploitation. However, existing works [24], [25] show that acquisition functions may fall into local optima under different blackbox functions, i.e., there is no one-fit-all acquisition function. Hence, we adopt the gp_hedge [25] strategy in the AdaSlicing system. Its basic idea is to dynamically choose one of the candidate acquisition functions during the iterations of Bayesian optimization. The selection of acquisition functions is optimized by using an online multi-armed bandit algorithm, which demonstrated promising performance compared to individual fixed acquisition functions.

V. Soft-Isolated RAN Virtualization

In this section, we introduce the soft-isolated RAN virtualization in the AdaSlicing system.

RAN virtualization is the foundation of network slicing technique to virtualize physical infrastructures (e.g., base stations) into virtual radio resources (e.g., vRBs) and implement virtual resources at runtime. During online resource orchestration, these virtual resources can be flexibly orchestrated to different network slices periodically (e.g., every minute or hour in conventional networks). At runtime, virtual resources of slices will be mapped to physical resources (e.g., PRBs/RBGs) by using virtual-to-physical mapping [26], [27], [28]. With the mapped physical resources, each slice will conduct its intra-slice user scheduling in every millisecond. The objective of RAN virtualization is to achieve high isolation (e.g., resource) and low overhead (e.g., computation). On the one hand, high isolation ensures that virtual resources are isolated with each other, and thus provides performance isolation among slices. Here, we denote RAN virtualization as *hard isolated* when virtual resources are only exclusively mapped to physical resources, e.g., a vRB always corresponds to a set of PRBs in the MAC layer. On the other hand, low overhead ensures that the implementation of virtual resources can be performed at a minimal computation time.

In the context of open radio access networks (e.g., O-RAN), the virtual resources of network slices can be orchestrated in fine time granularity (e.g., as low as subseconds via near-RT RIC) to better track their time-varying traffic variations. However, we observe that the orchestrated virtual resources of slices are usually under-utilized in the scenario of hard-isolated RAN virtualization. Fig. 2 shows the number of vRB utilization of a network slice under different virtual resources and slice applications in a mobile network testbed. It can be seen that, the orchestrated vRBs are not fully utilized due to different patterns of slice traffic, where only active speedtest may saturate the orchestrated vRBs. This resource under-utilization can be attributed to 1) slices are usually orchestrated to have sufficient virtual resources to accommodate their peak traffic until the next orchestration slot; 2) their virtual resources are exclusively mapped and cannot be shared with other slices, even if they are not fully used at runtime.

Therefore, we propose a soft-isolated RAN virtualization, as shown in Fig. 3, to improve the runtime utilization of virtual resources while assuring their isolation. Soft-isolation also serves as an enabler for the 3GPP-defined network resource model [29], which is considered the *de facto* standard for RAN slice resource management. The key idea is to enable the sharing of unused virtual resources among slices, before the virtual-to-physical mapping, at runtime. Specifically, we design a new virtual resource sharing (vSharing) layer to share expectantly excessive virtual resources among slices, where the inputs are the orchestration action of all slices and the outputs are the number of vRBs of all slices. First, we create two new kinds of virtual resources, including soft-isolated virtual resource blocks (svRBs) and sharing weights (SWs). Here, we assume that a virtual resource block (vRB) always corresponds to a fixed set of physical resources, i.e., one downlink resource block group (RBG) and one uplink physical resource block (PRB), in the virtual-to-physical mapping. Second, based on the orchestrated number of svRBs of each slice, we estimate the expectantly needed number of vRBs for intra-slice user scheduling, as illustrated in Fig. 4. This can be determined by aggregating all user traffic (e.g., as stored in the per-user RLC buffer) associated with a given slice. Third, we determine the total number of unused vRBs of all slices and build a vRB pool, where we skip these slices whose expectantly needed vRBs exceed their orchestrated svRBs. Fourth, we proportionally share the vRB pool to these slices with overflowed user traffic, according to their SWs.

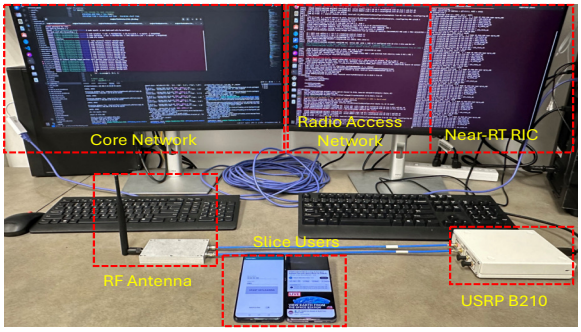

Fig. 5. The overview of AdaSlicing testbed.

Thus, we can obtain the final number of vRBs for all slices, including 1) for slices with overflowed traffic: summing up their orchestrated svRBs and the shared unused vRBs; 2) for slices without overflowed traffic: their expectantly needed vRBs. Finally, the virtual-to-physical mapping will be invoked to map the final vRBs of slices to physical resources (i.e., PRBs and RBGs).

VI. System Implementation

In this section, we describe the testbed implementation of the AdaSlicing system, including hardware, software, and architecture, and introduce the application of slices.

A. Testbed Specifications

We implement the AdaSlicing system on an end-to-end network slicing testbed, including the radio access network, core network, and near-real-time RIC, as illustrated in Fig. 5. We implement the RAN by using OpenAirInterface (OAI) [30] (v2022.41). The RAN is hosted in an Intel i7-14700K desktop (64G RAM) with a low-latency kernel of Ubuntu 22.04, which connects an Ettus USRP B210 as the RF front-end. We operate the base station at band 7 with 10MHz radio bandwidth (i.e., 50 physical resource blocks). We use a Faraday cage to containerize all smartphones for eliminating other radio interference. We implement the CN with Open5GS [31] (v2.7.0), which is hosted in another Intel i7-10700 desktop (32G RAM). The RAN and CN is connected with 1Gbps Ethernet cable. We implement the near-RT RIC with FlexRIC [26] (v1.0.0), which supports 4G slicing capability. We implement the AdaOrch algorithm with Python 3.11, which runs on the CN desktop. We adopt the scikit-optimize (v0.10.1) with the Gaussian Process estimator, which relies on the *GaussianProcessRegressor* module in *sklearn* toolkit [32]. Detailed testbed specifications are listed in Table I.

B. Slice Applications

We implemented an Android application[3] for each slice. As AdaSlicing focuses on inter-slice resource allocation, we use only one smartphone as the mobile user for each slice, for the sake of tractability. It is basically a video streaming application (downlink heavy), where the edge server (collocated in CN desktop) continuously sends video frames to individual mobile users. Note that, we change the application parameters for each slice (but unknown to AdaSlicing), in terms of the data size of video frames. The performance metrics of slice applications are 1) throughput and 2) frame-per-second, which are reported to near-RT RIC every second.

[3]AdaSlicing is fully compatible with other slice applications, as long as the near-RT RIC can periodically retrieve the performance of individual slices.

TABLE I
The detailed testbed specifications.

Component	Hardware	Software
Core Network	Intel Core i7-10700 Desktop	Open5GS
Open RAN	Intel Core i7-14700K Desktop	OpenAirInterface
SDR	Ettus USRP B210	UHD v4.5.0.0
Near-RT RIC	Intel Core i7-14700K Desktop	FlexRIC v1.0.0
UEs	OnePlus 9 5G	Andriod 11

VII. Performance Evaluation

In this section, we conduct extensive experiments to evaluate the performance of the AdaSlicing system from different perspectives. The goal is to answer the following questions: 1) how does AdaSlicing perform as compared to state-of-the-art network slicing systems? 2) how does AdaSlicing optimize online resource allocation in detail? 3) how does the soft-isolated RAN virtualization improve the resource utilization of slices as compared to existing hard isolation? and 4) to what extent, AdaSlicing can adapt to time-varying network dynamics?

We compare AdaSlicing with the following systems:

- **GBO**: GBO uses a global Bayesian optimization to optimize resource orchestration for all slices, under hard-isolated RAN virtualization. To assure the SLA of slices, its objective is penalized by using the the barrier method if their performance requirements are violated.
- **Atlas**: Atlas [5] is a state-of-the-art network slicing system, under hard-isolated RAN virtualization. Atlas focuses on the resource allocation of individual slices, which are optimized by using a Bayesian optimization independently. As it does not considered the resource capacity of infrastructures, we slightly modify it to enforce a simple scaling if the resource capacity is exceeded at runtime.
- **ExSearch**: Exhaustive search (ExSearch) uses the exhaustive search method to optimize the network slicing, under hard-isolated RAN virtualization, where it selects the action with the minimal cost while satisfying the performance requirement of all slices. Note that, it requires the whole action-to-performance dataset to be available in advance, which is impractical in real-world networks.

During the performance evaluation, we use the following experiment parameters, which are generally selected based on the realistic network capacity of the testbed. We create three slices and each slice has one smartphone user. The performance threshold of all slices $Q_i, \forall i \in \mathcal{I}$ are 12 Megabits-per-second (Mbps) throughput and 10 FPS. Without loss of generality, we use $U_H = 1, U_S = 1$ for weighting both svRB and SW. We use different compression qualities of .jpg in Android applications to stream their video frames to the edge server. The maximum number of svRBs is $H = 12$, where the minimum of 1 svRB is assigned to keep the slice alive. In the GP, we utilize the *Matern* kernel and *gp-hedge*

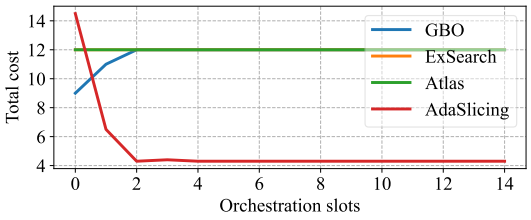

Fig. 6. The convergence of total cost under systems.

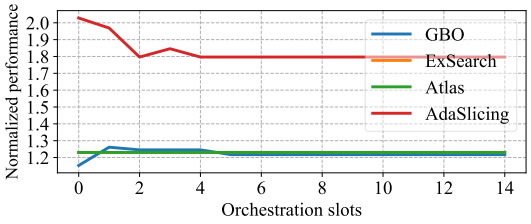

Fig. 7. The convergence of normalized performance under systems.

strategy includes expected improvement (EI), probability of improvement (PI), and low confidence bound (LCB).

A. Overall Performance

In this subsection, we show the overall performance (i.e., total cost and slice performance) achieved by all systems. Fig. 6 and Fig. 7 show the convergence of total cost and normalized performance obtained by different systems, respectively. Here, we define the *normalized performance* of a slice as the ratio between its performance $f(x_i, w_i|s_i)$ and the threshold Q_i, where multiple performance metrics will be averaged if applicable. We can see that, AdaSlicing achieves a fast convergence speed with only 5 iterations. This can be attributed to the high sample efficiency of individualized learning agents and the strong robustness of the coordinator in AdaSlicing. Table II shows the numerical results of the cost and slice performance after the convergence of all systems. We can see that, AdaSlicing obtains the lowest cost with the highest normalized performance at the same time, among all systems. As compared to the state-of-the-art Atlas, AdaSlicing reduces 64.2% cost, particularly the needed number of svRBs is reduced by 66.7%, and also improves 45.5% normalized performance in the meantime. In other words, although other comparison systems use more virtual resources (i.e., svRBs), their achieved normalized performances are still lower than that of AdaSlicing. This can be attribute to the soft-isolated RAN virtualization in AdaSlicing, which justifies the necessity of sharing virtual resources among slices at runtime.

B. AdaSlicing Dissection

In this subsection, we dissect the AdaSlicing system to show the details behind its achieved performance in Table II. First, Fig. 8 shows the convergence curve of overall cost, and that of all three agents in the 4th orchestration slot. It can be seen that, the overall cost starts high and then quickly decreases towards convergence. This is resulted from both the initialization of auxiliary and dual variables in the coordinator, and the inaccurate representation of learning agents in their early stages. In AdaSlicing, all learning agents and the coordinator will communicate to achieve the consensus and

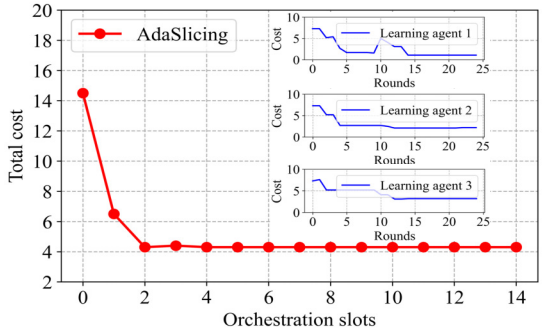

Fig. 8. The convergence of total cost in the AdaSlicing system (the red curve), including three detailed convergence curves under constrained Bayesian optimization (blue curves) at the 4-th orchestration slot.

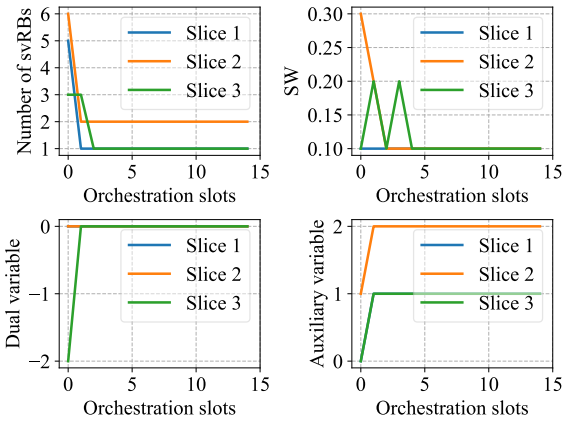

Fig. 9. The convergence of detailed variables.

generate the optimal orchestration action in each orchestration slot. Moreover, we design to reuse previous experiences in individual learning agents to accelerate their convergence in later orchestration slots, which can be observed in these subplot curves of learning agents. Fig. 9 show the detailed convergence of the orchestration actions, and these auxiliary and dual variables. It can be seen that, after the initialization of dual variables (-5), they are updated to be -2, and then converges to be 0. This is because that, the optimized number of svRBs by learning agents and the auxiliary variables are exactly equivalent after the second orchestration slot, which leads to no update to dual variables (See Eq. 16). These convergence curves verify the effectiveness of the coordinator in coordinating these learning agents in AdaSlicing.

C. Soft-Isolated RAN Virtualization

In this subsection, we evaluate the soft-isolated RAN virtualization in AdaSlicing, as compared to existing hard-isolated solutions. Here, we create two test slices, each has one mobile user, where we measure the throughput by simultaneously starting the online speedtest tool for both slice users. Fig. 10 shows the experienced throughput of two slices under both soft-isolated and hard-isolated RAN virtualization. In both experiments, slice 0 and 1 are assigned with [7 RBGs/svRBs, 0.2 SW] and [5 RBGs/svRBs, 0.1 SW], respectively. It can be seen that, under hard-isolated RAN virtualization (i.e., the top figure), the experienced throughput of both slices are static and generally proportional to their orchestrated number of

TABLE II
DETAILED CONVERGED ORCHESTRATION OF ALL SYSTEMS.

Methods	Cost	Individual cost	svRBs	SW	Normalized Performance	Throughput (Mbps)	FPS
GBO	12	(1, 5, 6)	(1, 5, 6)	-	1.22	(4.40, 12.40, 14.16)	(16, 18, 12)
ExSearch	12	(4, 4, 4)	(4, 4, 4)	-	1.23	(10.24, 10.24, 10.24)	(20, 17, 11)
Atlas	12	(4, 4, 4)	(4, 4, 4)	-	1.23	(10.24, 10.24, 10.24)	(20, 17, 11)
AdaSlicing	**4.3**	**(1.1, 2.1, 1.1)**	**(1, 2, 1)**	**(0.1, 0.1, 0.1)**	**1.79**	**(17.04, 18.40, 16.80)**	**(27, 22, 15)**

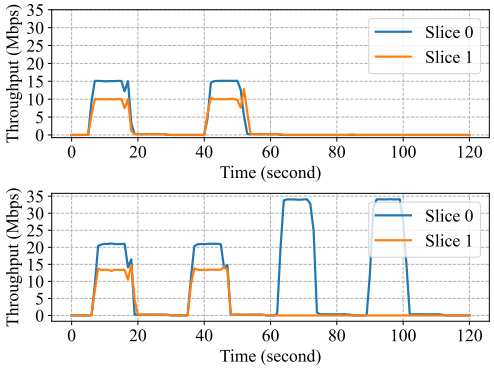

Fig. 10. The experienced downlink throughput under the hard-isolated (top) vs soft-isolated (bottom) RAN virtualization.

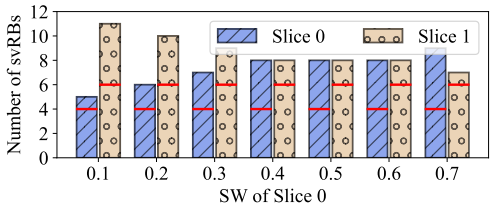

Fig. 11. The impact of SW in sharing virtual resources. Here, the read line in each bar is the originally assigned number of svRBs.

RBGs. In contrast, under soft-isolated RAN virtualization (i.e., the button figure), the experienced throughput of both slices are increased, which attributes to unused virtual resources in the RAN. Besides, the additionally gained throughput of slice 0 is nearly twice than that of slice 1, which proportionally corresponds to their SWs. At the later part of the bottom figure (i.e., 62 seconds), we only start the speedtest in the slice 0. We can see that, its experienced throughput soars up to 33.6 Mbps, which means it enjoys basically all the virtual resources from slice 1 and any other unused resources in RAN. Fig. 11 further shows the impact of different SWs in sharing the virtual resources among slices. Here, we fix the SW of the slice 1 as 0.3, and assign slice 0 and 1 with 4 and 6 svRBs, respectively. Given the total 16 svRBs in 10MHz RAN, there are 6 svRBs remained unused, which will be shared by slice 0 and 1. We can see that, the higher SW in the slice 0, its experienced number of svRBs increases accordingly. For example, when the SW of slice 0 is 0.2, it shares 40% from the unused 6 svRBs, i.e., 2.4 svRBs, which is grounded to be 2 svRBs. These results show that, the soft-isolated RAN virtualization can assure the isolation among virtual resources, while improving the resource utilization at runtime.

D. Scalability and Adaptability

In this subsection, we evaluate the AdaSlicing system under different scenarios, in terms of scalability and adaptability.

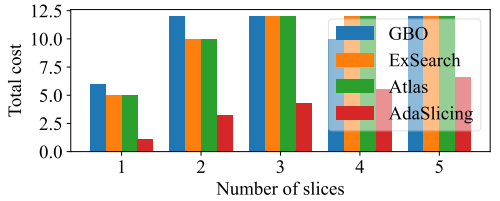

Fig. 12. The total cost under different number of slices.

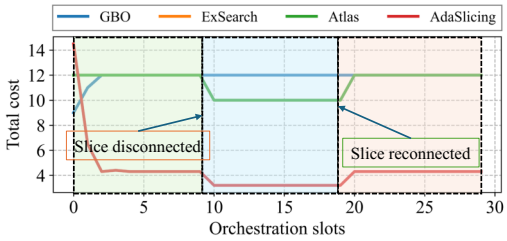

Fig. 13. The total cost under time-varying network dynamics.

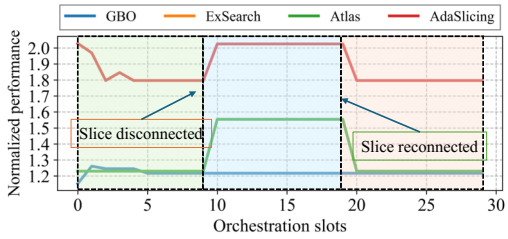

Fig. 14. The normalized performance under time-varying dynamics.

Fig. 12 shows the total cost of all systems under different number of slices (there are a maximum 5 slices due to the limited capacity of the testbed). As the number of slices increases in RAN, the more virtual resources are needed to support their users under the slice SLAs. We can see that, AdaSlicing can achieve the lowest total cost under all scenarios, which verifies the high scalability of AdaSlicing in handling large scale network slicing scenarios. Here, the total cost of other systems reach to the maximum svRBs in the system, while their normalized performances are decreased up to 53.9%, if supporting more than 3 slices. Fig. 13 and Fig. 14 show the convergence of total cost and normalized performance under time-varying number of active slices. Here, we disconnect slice 3 at the 10th orchestration slots and re-connect it back at the 20th orchestration slots, to emulate the network dynamics. It can be seen that, AdaSlicing quickly adapts to the departure of slice 3 within only one orchestration slot, where the total cost is reduced by 25.6 % while improving 12.8 % normalized performance. This is achieved by adaptively coordinating these auxiliary and dual variables in the coordinator in AdaSlicing. As slice 3 is reconnected, AdaSlicing can also adapt and converge back to the optima in a few orchestration slots. In contrast, GBO is designed to accommodate the peak traffic

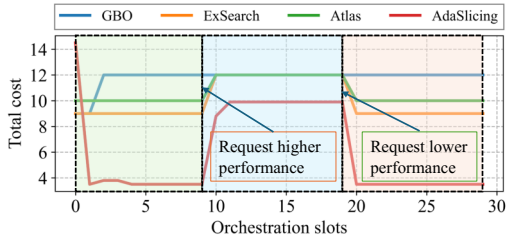

Fig. 15. The total cost under changing slice demands.

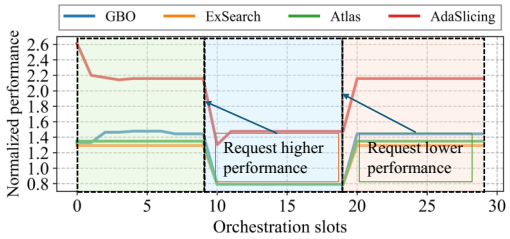

Fig. 16. The normalized performance under changing slice demands.

of slices, its total cost cannot be adapted to time-varying network dynamics. In addition, Fig. 15 and Fig. 16 show the total cost and normalized performance of all systems under time-varying performance threshold of slices. Here, we set the threshold of slices as [8Mbps, 10FPS], and increase them to [20Mbps, 15FPS] at the 10th orchestration slot, and reset it back at the 20th orchestration slot. We can see that, AdaSlicing can quickly adapt to this change in a few orchestration slots, by orchestrating more virtual resources to slices. Here, the decrease of normalized performance in Fig. 16 is because we divide the slice performance by the increased threshold. These experimental results justify the high adaptability of AdaSlicing, in terms of reacting to time-varying network dynamics in real-world networks.

VIII. Related Work

Open Radio Access Network. Open RAN has shown the increasing momentum in revolutionizing and defining the next generation mobile network [33]. FlexRIC [26] is an open-source flexible and efficient software development kit (SDK), and has been gradually adopted to build specialized and multi-service SD-RAN controllers. HexRIC [28] is a purpose-built next-generation network controller for the O-RAN ecosystem, featuring with the robust messaging infrastructure and AI/ML operation framework under the architecture of separated control and user plane. ColO-RAN [34] is the first publicly-available large-scale O-RAN testing framework, that is developed based on the Colosseum wireless network emulator with scaled software-defined radio and computational capabilities. Open RAN initiatives create unlimited possibilities in next-generation mobile network (e.g., rApps, xApps, dApps), where advanced AI/ML techniques are necessitated to handle ever-increasing complex fine-grained network management.

Machine Learning for Networking. AI/ML techniques have revealed convincing potential in dealing with complex and time-correlated network systems. To assure the SLA of end-to-end slices, Liu *et al.* proposed EdgeSlice [35], a decentralized deep reinforcement learning (DRL) approach, to dynamically orchestrate multi-domain radio and computing resources. To support increasing computing and storage demand of O-RAN compliant networks, Maxenti *et al.* proposed ScalO-RAN [36], a control framework to allocate and scale O-RAN applications under the given application-specific latency requirement. To facilitate spectrum sharing among network operators in O-RAN compliant networks, Bonati *et al.* proposed NeutRAN [37], a zero-touch framework to automate operator onboarding, supported by a new optimization engine and fully virtualized infrastructure. OrchestRAN [38] is a network intelligence orchestration framework towards the Open RAN paradigm, and aims to automatically optimize the set of data-driven algorithms while assuring time requirements and avoiding conflicts. Most policies of existing AI/ML-assisted network managements are trained with offline environments (e.g., simulators) or limited online data observation from real-world networks. Recent observations revealed that offline policies could suffer from simulation-to-reality discrepancy, leading to non-trivial performance degradation when applied to real-world networks.

Online Network Management. To address the simulation-to-reality gap, Zhang *et al.* [39] proposed OnRL to online update the DRL policy via interacting with real-world networks, to improve the performance of real-time mobile video telephony. To enable online network configuration in wireless mesh networks (WMNs), Shi *et al.* [7] proposed a new transfer learning-based algorithm that bridges the simulation-to-reality gap, according to both offline and online datasets. Hu *et al.* [40] proposed a new neural-assisted algorithm to optimize radio resources to slices, by introducing a DNN to approximate the complex performance function of heterogeneous slices. Liu *et al.* [5] proposed Atlas, an online network slicing system, to automate the service configuration of slices with assured slice SLAs, via safe and sample-efficient learn-to-configure approaches. However, these online learning works heavily rely on parameterized DNN agents with fixed input and output spaces, and cannot efficiently adapt to potential time-varying network dynamics in real-world networks.

IX. Conclusion

In this paper, we presented AdaSlicing, a new adaptive network slicing system, that online learns to orchestrate virtual resources while efficiently adapting to time-varying network dynamics. We designed the soft-isolated RAN virtualization that significantly improves the virtual resource utilization of slices without breaking the promise of resource isolation. We designed the AdaOrch algorithm that minimizes the total operating cost of supporting all slices with assured slice SLAs, via online resource orchestration. We found that the integration of AI/ML techniques and optimization methods could combine their individual advantages (e.g., high approximation capability and robust convergence) during online resource orchestration in real-world networks.

Acknowledgement

This work is supported by the US National Science Foundation under Grant No. 2321699, No. 2333164, and No. 2428427.

REFERENCES

[1] H. Bagheri, M. Noor-A-Rahim, Z. Liu, H. Lee, D. Pesch, K. Moessner, and P. Xiao, "5g nr-v2x: Toward connected and cooperative autonomous driving," *IEEE Communications Standards Magazine*, vol. 5, no. 1, pp. 48–54, 2021.

[2] Y. Guan, X. Hou, N. Wu, B. Han, and T. Han, "Deepmix: mobility-aware, lightweight, and hybrid 3d object detection for headsets," in *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, ser. MobiSys '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 28–41. [Online]. Available: https://doi.org/10.1145/3498361.3538945

[3] T. Qiu, J. Chi, X. Zhou, Z. Ning, M. Atiquzzaman, and D. O. Wu, "Edge computing in industrial internet of things: Architecture, advances and challenges," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 4, pp. 2462–2488, 2020.

[4] Q. Liu, N. Choi, and T. Han, "OnSlicing: online end-to-end network slicing with reinforcement learning," in *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*, 2021, pp. 141–153.

[5] ——, "Atlas: automate online service configuration in network slicing," in *Proceedings of the 18th International Conference on emerging Networking EXperiments and Technologies*, 2022, pp. 140–155.

[6] 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 *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, 2018, pp. 191–206.

[7] J. Shi, M. Sha, and X. Peng, "Adapting wireless mesh network configuration from simulation to reality via deep learning based domain adaptation," in *NSDI*, 2021, pp. 887–901.

[8] 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 *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*, 2018, pp. 353–365.

[9] O.-R. ALLIANCE, "O-ran alliance," June 2022 [Online]. [Online]. Available: https://www.o-ran.org/

[10] M. Polese, L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "Understanding o-ran: Architecture, interfaces, algorithms, security, and research challenges," *IEEE Communications Surveys & Tutorials*, 2023.

[11] S. Azimeh, V. Csaba, and G. Markus, "UNEXT – A unified networking experience," Nokia Bell Labs, White Paper, 2023.

[12] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[13] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[14] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proceedings of the ACM special interest group on data communication*, 2019, pp. 270–288.

[15] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[16] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[17] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd, "A rewriting system for convex optimization problems," *Journal of Control and Decision*, vol. 5, no. 1, pp. 42–60, 2018.

[18] S. Boyd and L. Vandenberghe, "Convex optimization, 25 cambridge university press," *Cambridge, England*, 2010.

[19] P. I. Frazier, "A tutorial on bayesian optimization," *arXiv preprint arXiv:1807.02811*, 2018.

[20] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, 2012.

[21] C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer school on machine learning*. Springer, 2003, pp. 63–71.

[22] E. Brochu, V. M. Cora, and N. De Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *arXiv preprint arXiv:1012.2599*, 2010.

[23] C. Williams and C. Rasmussen, "Gaussian processes for regression," *Advances in neural information processing systems*, vol. 8, 1995.

[24] T. d. P. Vasconcelos, D. A. de Souza, C. L. Mattos, and J. P. Gomes, "No-past-bo: Normalized portfolio allocation strategy for bayesian optimization," in *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2019, pp. 561–568.

[25] M. Hoffman, E. Brochu, N. De Freitas *et al.*, "Portfolio allocation for bayesian optimization," in *UAI*, 2011, pp. 327–336.

[26] R. Schmidt, M. Irazabal, and N. Nikaein, "Flexric: An sdk for next-generation sd-rans," in *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*, 2021, pp. 411–425.

[27] Q. Liu and T. Han, "Virtualedge: Multi-domain resource orchestration and virtualization in cellular edge computing," in *Proceedings of IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 2019, pp. 1051–1060.

[28] V.-Q. Pham, H.-T. Thieu, A. Kak, and N. Choi, "Hexric: Building a better near-real time network controller for the open ran ecosystem," in *Proceedings of the 24th International Workshop on Mobile Computing Systems and Applications*, 2023, pp. 15–21.

[29] 3rd Generation Partnership Project (3GPP), "5G; Management and orchestration; 5G Network Resource Model (NRM)," Technical Specification (TS) 28.541, May 2024, release 18.7.0.

[30] OpenAirInterface Software Alliance. [Online]. Available: https://gitlab.eurecom.fr/oai/openairinterface5g

[31] Open5GS. [Online]. Available: https://github.com/open5gs/open5gs

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[33] A. Kak, H.-T. Thieu, V.-Q. Pham, R. K. Sheshadri, N. Choi, Y. Guan, M. Yin, and T. Han, "Aweran: Making a case for application-aware radio access network slicing," in *Proceedings of the 17th ACM Workshop on Wireless Network Testbeds, Experimental evaluation & Characterization*, 2023, pp. 41–48.

[34] M. Polese, L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "Colo-ran: Developing machine learning-based xapps for open ran closed-loop control on programmable experimental platforms," *IEEE Transactions on Mobile Computing*, vol. 22, no. 10, pp. 5787–5800, 2022.

[35] Q. Liu, T. Han, and E. Moges, "Edgeslice: Slicing wireless edge computing network with decentralized deep reinforcement learning," in *Proceedings of IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, 2020, pp. 234–244.

[36] S. Maxenti, S. D'Oro, L. Bonati, M. Polese, A. Capone, and T. Melodia, "Scalo-ran: Energy-aware network intelligence scaling in open ran," *arXiv preprint arXiv:2312.05096*, 2023.

[37] L. Bonati, M. Polese, S. D'Oro, S. Basagni, and T. Melodia, "Neutran: An open ran neutral host architecture for zero-touch ran and spectrum sharing," *IEEE Transactions on Mobile Computing*, 2023.

[38] S. D'Oro, L. Bonati, M. Polese, and T. Melodia, "Orchestran: Orchestrating network intelligence in the open ran," *IEEE Transactions on Mobile Computing*, 2023.

[39] H. Zhang, A. Zhou, J. Lu, R. Ma, Y. Hu, C. Li, X. Zhang, H. Ma, and X. Chen, "Onrl: improving mobile video telephony via online reinforcement learning," in *MobiCom*, 2020, pp. 1–14.

[40] T. Hu, Q. Liao, Q. Liu, A. Massaro, and G. Carle, "Fast and scalable network slicing by integrating deep learning with lagrangian methods," in *GLOBECOM 2023-2023 IEEE Global Communications Conference*. IEEE, 2023, pp. 6346–6351.