Flexible RAN Slicing in Open RAN with Constrained Multi-agent Reinforcement Learning

Mohammad Zangooei, Morteza Golkarifard, Mohamed Rouili, Niloy Saha, and Raouf Boutaba
{mzangooei, mgolkari, mrouili, n6saha, rboutaba}@uwaterloo.ca
University of Waterloo, Canada

***Abstract*—Network slicing enables the provision of customized services in next-generation mobile networks. Accordingly, the network is divided into logically isolated networks that share underlying resources but are tailored to meet the distinct service requirements of their users. However, allocating the minimum necessary resources to satisfy slices' requirements is challenging, particularly when the number of slices is variable or too large which is envisioned in Open RAN. State-of-the-art proposals leverage reinforcement learning (RL) algorithms; however, they suffer from over-provisioning and/or frequent violations of service-level agreement (SLA) due to the large and changing state and action spaces. This paper introduces a novel cooperative multi-agent RL algorithm for RAN slicing in Open RAN, designed to adapt to variable slice numbers and effectively scale as they grow. To train this model, we exploit a novel constrained RL algorithm that explicitly considers SLA constraints to maintain a decreasing SLA violation ratio during training. Our approach is compatible with the Open RAN architecture, allowing for feasible deployment in future mobile networks. CMARS surpasses RL methods in SLA satisfaction by 50% in large-scale slicing, using only 9% more resources. It has 8% fewer SLA violations and 19% lower resource consumption for a flexible number of slices.**

***Index Terms*—5G and beyond mobile networks, Open RAN, Radio resource management, Network slicing, Reinforcement learning.**

I. Introduction

5G and beyond mobile networks are expected to offer a variety of service types including enhanced mobile broadband (eMBB), ultra-reliable low-latency communications (uRLLC), and massive machine-type communications (mMTC) [1]. To facilitate this, network slicing is used to partition the network into multiple *slices* that can be individually managed, optimized, and secured to meet specific service-level agreements (SLAs) in terms of latency, throughput, reliability, etc. [2]–[4]. In this service model, third-party entities or *tenants* can request separate slices from a mobile network operator (MNO) with particular SLAs to serve their users based on the type of service they require [5]. MNOs need to minimize radio resource consumption while satisfying slices' SLAs in radio access networks (RAN) which is known as the *RAN slicing* problem.

Significant research on RAN slicing has been done in the past few years, however with limited to no deployment in practice. This can be attributed to the proprietary, vendor-specific, monolithic, and closed architectures of existing RAN solutions. To address these limitations, the *O-RAN Alliance* has recently proposed the Open RAN architecture featuring disaggregation, softwarization, interoperability, and standardization of different RAN elements and interfaces [6]. These specifications extend 3GPP standards for promoting openness in RAN architectures. Among others, the O-RAN architecture introduces radio intelligent controllers (RICs) that host various applications to monitor and control different elements of the RAN. For example, an application sitting on top of the RIC can be easily deployed for making RAN slicing decisions based on slice requests' arrival, service requirements, and network workload. Clearly, the Open RAN movement provides a unique opportunity for achieving higher levels of RAN programmability including flexible RAN slicing and dynamic radio resource management, and for accelerating research and practical deployment of RAN slicing in 5G and beyond mobile networks.

RAN slicing presents challenges due to the variability of radio channels, diverse SLA requirements, and limited radio resources. As a result, reinforcement learning (RL)-based approaches have emerged as the state-of-the-art solution, surpassing traditional heuristic methods [2], [3], [5], [7]–[14]. However, particular characteristics of RAN slicing in an Open RAN environment are not considered in the design of the existing RL-based RAN slicing algorithms in the literature. For example, O-RAN specifications enable slices to seamlessly join or leave the network during their life cycle [15]. Nonetheless, the existing RL-based RAN slicing methods fail to efficiently provide this flexibility and monitor, control, and prioritize service on a per-tenant basis [2], [3], [7]–[12]. Also, the integration of O-RAN into future generations of mobile networks is widely anticipated [16]. With the growing proliferation of diverse application scenarios, it is crucial to circumvent any constraints imposed on the number of slices. A higher slice count can negatively impact the performance of the existing RL-based methods since the state and action spaces are proportional to the number of slices (curse of dimensionality) [5], [9]. In this paper, we address the mentioned shortcomings with the following main contributions:

- We introduce CMARS, a novel **C**onstrained **M**ulti-**A**gent RL algorithm specifically designed for **R**AN **S**licing in Open RAN. CMARS effectively addresses resource allocation challenges outlined in O-RAN specifications, surpassing the limitations of existing RL-based approaches.
- We formulate RAN slicing problem as a stochastic optimization task, allowing multiple instances of each slice

Manuscript received 7 January 2023; revised 9 June 2023; accepted 6 September 2023.

type to join or leave the network. CMARS enables dynamic service provisioning by assigning dedicated agents to make resource allocation decisions for each tenant and employing a sequential decision-making procedure among them. We eliminate the need for model retraining when new slices join the network by allowing agents to reuse trained policies and enhance optimality by enabling agents to communicate their state.
- CMARS has been deliberately designed to inherently support a substantial number of slices. It achieves this by distributing resource allocation decisions for individual slices among multiple agents, thereby alleviating the curse of dimensionality and facilitating the exchange of experiences among the decision-making entities.
- CMARS follows a constrained RL scheme based on multi-agent proximal policy optimization Lagrangian [17] which explicitly considers slice performance constraints while minimizing resource consumption. Through cooperative and constrained RL methods as well as global criticism and reward scaling techniques, CMARS achieves notable improvements in efficiency and performance, outperforming baseline benchmarks.
- Finally, we extend the RAN slicing simulator from [5] and conduct extensive experiments to evaluate CMARS. The results demonstrate its superiority over state-of-the-art methods in various scenarios, including when the number of slices is fixed or dynamic, resources are scarce or abundant, and the number of slices is low or high.

The paper is organized as follows: section II presents related work, section III formulates the RAN slicing problem and our solution approach. section IV details our proposed CMARS method, section V discusses the integration of CMARS into O-RAN specifications, and section VI provides simulation results. Finally, section VII summarizes and concludes the paper.

II. Related work

Network slicing has received considerable attention in recent years [1], [18]. According to 3GPP [19], a network slice comprises a collection of network functions across the core, transport, and radio access networks. These functions are customized to fulfill diverse SLA requirements, such as latency, and throughput. Resource management in network slicing encompasses various resources, including communication, computation, radio, or a combination of them [20]–[23].

The management of radio resources is challenging due to the scarcity of radio resources, dynamicity of wireless channels, and interferences [13]. Various frameworks, such as queuing theory [24], game theory [25], [26], classic optimization [27], [28], and reinforcement learning [8], [12]–[14], [29], [30] have been utilized to address this problem. Despite outperforming other methods, existing RL-based solutions [8], [12], [14] suffer from sub-optimal performance in dealing with large or varying numbers of slices. Therefore, it is imperative for us to devise a novel RL framework that supports scalability and flexibility in accommodating network slices.

Specifically, most proposals in the literature only consider one big instance per slice type and pool together different tenants (a group of users under the same administrative entity) that request the same slice type [2], [3], [7]–[10], [12]. However, this approach suffers from a lack of per-tenant observability and fine-grain control. If the demand of a tenant increases unprecedentedly, it will negatively affect other tenants of the same slice, leading to SLA violations. To address this shortcoming, authors in [5], [11] proposed to individually consider different tenants instead of pooling them together. Although this approach solves the limitations of tenant pooling, it raises optimality concerns as the decision of each tenant is made without considering the resource demand and usage of other tenants. In contrast, we use cooperative multi-agent reinforcement learning where each agent decides the share of resources to be allocated to a specific slice based on the aggregated slices' statistics and receives correlated rewards.

TABLE I: Comparison of existing RL-based RAN slicing literature with our proposal

Property	[2]	[3]	[5]	[7]	[8]	[9]	[10]	[11]	[14]	Ours
Per-tenant observability	✗	✗	✓	✗	✗	✗	✗	✓	✗	✓
Constraint-awareness	✗	✗	✓	✓	✗	✗	✓	✗	✗	✓
Scalability in slice count	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
Flexibility in slice count	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓
Experience reuse	✗	✗	✗	✗	✓	✗	✓	✓	✗	✓
Real-world O-RAN Testbed	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗

Online training of RL agents from scratch in production networks can result in the agents taking random actions that may lead to poor performance and SLA violations [10]. To mitigate this issue, pre-trained agents can be leveraged to reduce SLA violations in the early phases of deployment [8], [10], [11]. For new slice instances that arrive in the network, we reuse the policy learned from similar agents' experiences. However, we acknowledge the necessity of initially training the policy for each slice type, which can be accomplished through offline RL methods such as behavior cloning and imitation learning [10] or prior online training on pilot testbeds [9].

Polese *et al*. [9] showed that online training can adapt models to deployment-specific characteristics but at the cost of temporarily reduced efficiency. In response, constrained RL has been shown to respect SLA constraints during online training [7], [10] as they separate the conflicting objectives of reducing resource consumption while increasing SLA satisfaction ratio. In this regard, we also employ constrained RL schemes in the context of multi-agent policy optimization.

Previous studies in the literature have primarily focused on simulation-based investigations of RL-based RAN slicing [8], [10], [11], [14]. However, a recent development by [9] introduced an RL-based RAN slicing method that has been deployed on a large-scale testbed conforming to O-RAN specifications. Our proposed RAN slicing algorithm is currently assessed in a simulated environment, with plans to integrate it into an Open RAN testbed in the future.

A summary of the all aforementioned related works is presented in Table I which also provides a comparison of the state-of-the-art RL-based RAN slicing for 5G with our proposed approach. To the best of our knowledge, our work is the first to consider all of the properties listed in Table I by using a novel cooperative constrained multi-agent approach.

This article has been accepted for publication in IEEE Journal on Selected Areas in Communications. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2023.3336156

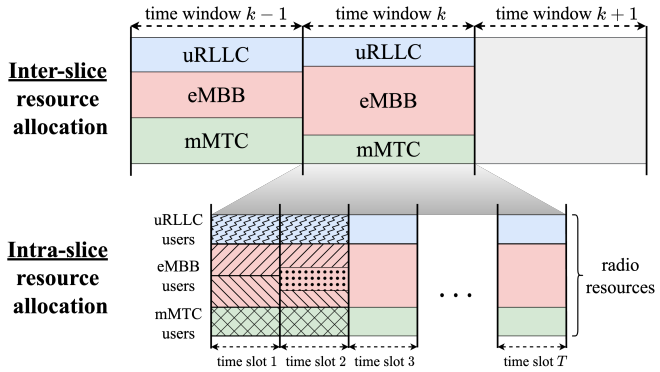


Fig. 1: Inter-slice and intra-slice resource allocation. The inter-slice allocation distributes PRBs among slices at the beginning of each time window. Colors represent the resources allocated to each slice in this figure. In each of the T time slots of a time window, the intra-slice allocation is determined by individual schedulers for each slice. In time slot 1 of time window k, there are a total of two eMBB users, which increases to three in the next time slot, while the number of uRLLC and mMTC users remains the same (one user each). The corresponding slice schedulers allocate resources to these users in each slot.

III. Problem Formulation and Solution Approach

Radio resources are modeled as a grid of blocks over two dimensions of time and frequency. These valuable resources, known as physical resource blocks (PRBs), are scheduled to carry users' data [2], [9]. RAN slicing aims to satisfy each tenant's service requirements by consuming the minimum amount of PRBs [3], [11], [14]. This is achieved by periodically adjusting the PRB allocation for each slice in an upcoming time window, followed by fine-grained distribution to users within each slice using a dedicated scheduler. These two sub-problems are respectively known as inter-slice and intra-slice RAN slicing problems [2]–[4] and are depicted in Fig. 1.

Following other proposals in the literature [2], [3], [5], [7], [8], [11], [12], [14], we consider a single-cell scenario, assuming a given amount of PRBs at the base station (BS) as it can be easily extended to multi-cell scenarios. Besides, we consider a single direction of data transmission in this paper for simplicity. With all of these assumptions, we present the RAN slicing problem in sections III-A and III-B. Then, we provide the system description in section III-C, and our RL-based solution approach to solve the problem in III-D. Summaries of the notations that are used in this section are presented in Table II and Table III.

A. Intra-slice resource allocation

Consider $\mathcal{S}_k$ as the set of slices in time window k, and R_k^s as the set of PRBs already allocated to slice s in time window k. Note that the decision on R_k^s has been made in the inter-slice allocation problem. On the other hand, assume a given set of users $\mathcal{U}_t$ in time slot t during the corresponding time window such that each user u is already associated with a specific slice s. In the intra-slice resource allocation problem, the only decision variable is the allocation of PRB j to user u at time slot t (also known as *scheduling* problem [2]–[4]).

The intra-slice allocation problem needs to be solved in a real-time manner that imposes strict time constraints on a potential algorithm. Therefore, heuristics such as *round robin* and *proportional fairness* (PF) are leveraged in real-world implementations [9]. Accordingly, we use a PF scheduler, but our proposed inter-slice allocation method is independent of the underlying intra-slice allocation scheme, and it works with any scheduling algorithm.

B. Inter-slice Resource Allocation

This problem decides the distribution of PRBs across different slices during the upcoming time window. This involves forecasting the future demand of each slice, calculating the minimum resource allocation required to meet the predicted demand, and implementing suitable policies for resource distribution. Each slice type has its own unique QoS requirements, including throughput, latency, and reliability. Therefore, it is necessary to differentiate service provisioning based on slice type. Additionally, tenants of the same slice type may have varying tolerances for SLA violations and are willing to pay different fees accordingly.

The fact that we explicitly consider a dynamic number of instances per slice type distinguishes our problem formulation from previous works [2], [3], [5], [7], [8], [11], [12], [14]. Particularly, only one instance of each slice type is modeled in these works. Nonetheless, this method suffers from a lack of per-tenant observability and control. On the other hand, multiple instances of the same slice type with different SLA thresholds are considered in [11]. However, such a formulation hinders offering a heterogeneous set of services which is critical in 5G and beyond networks.

We consider a finite set of slice types and performance levels within each type while multiple independent instances of them can be present in the network. We assume these instances may join or leave between time windows, but their connection status does not change during a time window. Consequently, we formulate the RAN slicing problem on time window k in equations 1a, 1b, and 1c.

In this formulation, vector R_k is the only decision variable that denotes the inter-slice allocation decision at time window k. Accordingly, the objective function (1a) is composed of two parts to minimize long-term resource consumption: summation of PRB usage in time window k and expectation of discounted PRB usage in the future time windows. Here, discount factor ζ models the effect of slicing decision in time window k on the following time windows which we argue should be considered here to achieve global optimality. Notably, the value of ζ depends on the dynamics of the system and needs to be empirically calculated. Intuitively, if no packets remain buffered between consecutive time windows, the value of ζ would be low.

The first set of constraints (1b), on the other hand, captures each slice's requirements in terms of SLAs in a way that the expected constraints violation should remain under a certain threshold. Specifically, the value of v_k^s, which is the violation of slice s in time window k, depends on the resource allocation as well as the future demand which is not known at decision time. Finally, constraint (1c) takes care of resource capacity at time step k.

$$\underset{R_k}{\text{minimize}} \quad \left\{ \sum_{s\in\mathcal{S}_k} R_k^s + \mathbb{E}\left[\sum_{j=1}^{\infty} \zeta^j \sum_{s\in S_{k+j}} R_{k+j}^s \right] \right\} \tag{1a}$$

$$\text{s.t.} \quad \mathbb{E}\left[v_k^s + \sum_{j=1}^{\infty} \zeta^j v_{k+j}^s \right] \leq \epsilon^s, \quad \forall s \in \mathcal{S}_k, \tag{1b}$$

$$\sum_{s\in\mathcal{S}_k} R_k^s \leq \mathcal{M}. \tag{1c}$$

It is worth mentioning that the precise relation between SLA satisfaction with resource allocation and demand is unknown. Additionally, the demand during the next time window is not known either. Nevertheless, RL methods are recognized as a promising approach to dealing with such uncertainties. As such, we resort to RL algorithms to efficiently solve the problem above.

C. System Description

In this paper, we conform to the RAN slicing model leveraged in [5], [31] which is the only work in the literature considering distinct tenants. Specifically, they consider two types of slices, namely eMBB and mMTC, while the SLAs for each instance are defined based on the particular service requirements and preferences of the corresponding tenant.

In the context of **mMTC** slices, each device is assigned a predetermined number of packet repetitions based on its estimated pathloss. Furthermore, there is typically a maximum limit imposed on the number of active devices that the slice can support [5], [31]. The expected key performance indicators (KPIs) in this scenario are defined in terms of maximum average delay. Consequently, the observed variables in this model encompass the number of simultaneous active devices, the average delay per UE, and the number of remaining packet repetitions per UE.

eMBB slices set specific limits to the average number of PRBs consumed by each type of traffic (guaranteed bit rate (GBR) and non-GBR). KPIs in this case are the average data rate and the maximum queue length [5], [31]. Furthermore, the observation comprises a set of variables for each type of traffic (GBR and non-GBR), since each type is associated with a specific QoS requirement in the SLA. These variables provide information per traffic type: incoming and delivered traffic rate, resource consumption, queue length, and signal-to-noise ratio (SNR).

We extend this model according to [9], introducing **uRLLC** slices, a crucial service type in 5G and beyond networks. We consider an SLA of maximum queue length for uRLLC users following a non-GBR traffic model as it impacts users' experienced latency.

Our model includes multiple instances per slice type to accommodate diverse performance levels. Each slice type has a fixed set of performance levels in the corresponding SLAs. These slice-specific SLAs define different thresholds for KPIs. Tenants can select a supported performance level from their slice type, based on their service requirements (see Table V for an example).

TABLE II: Summary of notation used in sections III-A and III-B

Symbol	Definition
u	user index
j	PRB index
k	time window index
t	time slot index
$\mathcal{U}_t$	set of users in time slot t
$\mathcal{S}_k$	set of slices in time window k
R_k^s	set of PRBs allocated to slice s in time window k, decided in the corresponding inter-slice problem
$\mathcal{M}$	number of available PRBs
v_k^s	SLA violation of slice s during time window k
ϵ^s	SLA violation threshold of slice s
ζ	discount factor of future time windows

D. Preliminaries of our RL Solution Approach

1) Cooperative Multi-agent Reinforcement Learning: In single-agent RL methods, the agent interacts with the environment under a specific Markov decision process (MDP) model; it observes the environment *state* based on which it takes an appropriate *action* to maximize the cumulative discounted *reward* it receives from the environment. In fact, the agent develops a policy to decide about the appropriate action in a given state [32]. Similarly, multiple agents interact with a shared environment in multi-agent RL (MARL) methods [33].

MARL schemes potentially enhance the scalability of the RAN slicing algorithm by decoupling resource allocation decisions among agents. In single-agent RL frameworks, slice statistics are concatenated to form the MDP state, leading to the curse of dimensionality when the number of slices increases. In contrast, MARL approaches allow independent state space sizes for each agent, mitigating the curse of dimensionality. Moreover, MARL enables experience reuse among agents which improves performance efficiency (see section IV-A for details).

In our proposed approach, all agents are consolidated within the same physical entity, eliminating inter-agent signaling and management overhead found in distributed multi-agent settings. However, in this consolidated setting, sequential decision-making by different agents requires multiple neural network invocations, unlike a centralized single-agent approach. Each individual neural network invocation takes less than a millisecond due to the small size and simplicity of the architectures used. While the cumulative effect of multiple invocations increases overall execution time in CMARS, it remains manageable within practical bounds.

In general, MARL methods can be classified into two categories: cooperative and competitive. In the cooperative category, all agents collaborate towards a shared objective, while in the competitive category, agents strive to maximize their individual rewards at the expense of others [34]. Competitive agents prioritize their own rewards while minimizing the rewards of other agents, potentially leading to a decline in overall system performance [35]. In our framework, we have developed a cooperative game model, where different agents receive rewards that are correlated with the overarching system goal.

2) Constrained Cooperative MARL: RL methods are known to suffer from taking detrimental actions during training due to random exploration. Constrained RL methods have gained attention in the literature to deal with this challenge and perform safe learning [7], [10]. These methods introduce a cost parameter to the MDP to measure the cost of an action and guide the agent toward avoiding actions with high costs.

Furthermore, the literature on RL-based RAN slicing confirms the effectiveness of these constrained RL approaches as they separate the conflicting objectives of meeting slice SLAs and consuming minimum resources [7], [10]. Therefore, we devise a constrained cooperative MARL method to solve the RAN slicing problem in this paper.

Constrained RL approaches introduce additional computations to enforce constraints, potentially increasing computational overhead and slowing down learning. However, we consider this overhead manageable since it occurs only during training. Once the model is trained, it can be invoked with input variables in various scenarios, regardless of the training algorithm used.

Consider a constrained multi-agent Markov decision process (CMMDP) [17] defined by the tuple $\langle \mathcal{N}, \mathcal{S}, \mathcal{A}, \rho^0, \mathcal{P}, r, \gamma, C, c \rangle$. In this definition, $\mathcal{N} = \{1, 2, \cdots, n\}$ is the set of agents, $\mathcal{S}$ is the finite state space, $\mathcal{A} = \prod_{i=1}^{n} \mathcal{A}^i$ is the joint action space, ρ^0 is the initial state distribution, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability function, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $\gamma \in [0, 1]$ is the discount factor, $C = \{C^i\}_{i \in \mathcal{N}}$ is the set of cost functions, and $c = \{c^i\}_{i \in \mathcal{N}}$ is the set of corresponding cost thresholds.

At step k, each agent $i \in \mathcal{N}$ is at state $s_k \in \mathcal{S}$ and take action $a_k^i \in \mathcal{A}^i$ according to the policy $\pi^i(\cdot|s_k)$. After which, the environment will give the joint reward of r_k to agents and move them to the new state s_{k+1} with probability $P(s_{k+1}|s_k, a_k)$ while each agent i pays the cost of C^i. The joint policy π together with the transition probability function $\mathcal{P}$ and the initial distribution ρ^0 produce a marginal state distribution at step k, ρ_π^k. Accordingly, the state value function V_π and the state-action value function Q_π are defined as follows:

$$V_\pi(s) \triangleq \mathbb{E}_{a_{0:\infty} \sim \pi, s_{1:\infty} \sim \mathcal{P}} [\sum_{k=0}^{\infty} \gamma^k r_k | s_0 = s], \tag{2}$$

$$Q_\pi(s, a) \triangleq \mathbb{E}_{a_{0:\infty} \sim \pi, s_{1:\infty} \sim \mathcal{P}} [\sum_{k=0}^{\infty} \gamma^k r_k | s_0 = s, a_0 = a]. \tag{3}$$

Here, $A_\pi(s, a) \triangleq Q_\pi(s, a) - V_\pi(s)$ is the advantage function. Moreover, we leverage fully cooperative MARL algorithms where different agents collaborate to maximize the expected discounted total reward $\mathcal{J}_r$ while trying to satisfy each agent i's safety constraint $\mathcal{J}_c^i$:

$$\mathcal{J}_r(\pi) \triangleq \mathbb{E}_{s_{0:\infty} \sim \rho_\pi^{0:\infty}, a_{0:\infty} \sim \pi} \left[\sum_{k=0}^{\infty} \gamma^k r_k \right], \tag{4}$$

$$\mathcal{J}_c^i(\pi) \triangleq \mathbb{E}_{s_0 \sim \rho^0, a_{0:\infty} \sim \pi, s_{1:\infty} \sim \mathcal{P}} \left[\sum_{k=0}^{\infty} \gamma^k C^i(s_k, a_k) \right] \leq c^i. \tag{5}$$

3) Multi-Agent Constrained Policy Optimisation: Policy-based methods offer the advantage of bounded policy improvement, making them suitable for employing constrained RL approaches [7]. This is particularly valuable in addressing the inherent trade-off between minimizing resource utilization and maximizing SLA satisfaction in RAN slicing as previously mentioned. Additionally, empirical studies have demonstrated that policy-based methods outperform value-learning approaches in cooperative multi-agent settings [36]. In light of these pieces of evidence, we stick to policy-based RL approaches to propose a MARL RAN slicing algorithm.

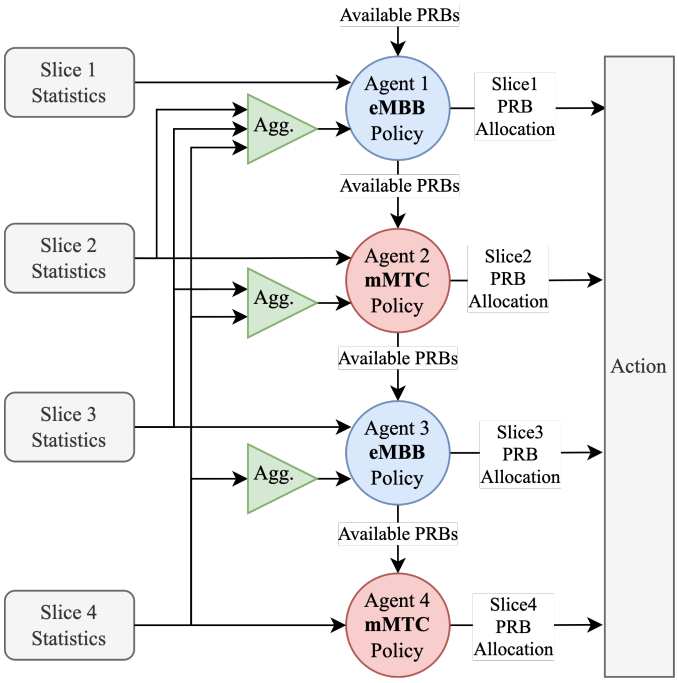


Fig. 2: Sequential decision-making in cooperative MARL with policy reuse among agents of the same slice type.

Researchers have shown that the joint advantage function of multiple agents can be decomposed into the sum of their local advantages [17], [37]. This property guarantees the improvement of joint policy with a sequential policy-update scheme among different cooperating agents. Inspired by [17], we parameterize each agent's policy $\pi_{\theta^i}^i$ by a neural network θ^i and formulate the optimization problem that agent i should solve to update its policy in step k as follows:

$$\theta_{k+1}^i = \text{argmax}_{\theta^i} \mathbb{E}_{s \sim \rho_{\pi_{\theta_k}}, a^{1:i-1} \sim \pi_{\theta_{k+1}^{1:i-1}}^{1:i-1}, a^i \sim \pi_{\theta^i}^i} \left[A_{\pi_{\theta_k}}^i \left(s, a^{1:i-1}, a^i \right) \right], \tag{6a}$$

$$\text{s.t.} \quad \mathcal{J}_c^i(\pi_{\theta_k}) + \mathbb{E}_{s \sim \rho_{\pi_{\theta_k}}, a^i \sim \pi_{\theta_k^i}^i} \left[A_{\pi_{\theta_k}}^i \left(s, a^i \right) \right] \leq c^i, \tag{6b}$$

$$\mathbb{E}_{s \sim \rho_{\pi_{\theta_k}}} \left[\mathrm{D}_{\mathrm{KL}}(\pi_{\theta_k}^i(\cdot|s), \pi^i(\cdot|s)) \right] \leq \delta. \tag{6c}$$

To solve this optimization problem, we use a method called multi-agent proximal policy optimization Lagrangian (MAPPOL) [17] which is, to the best of our knowledge, the only cooperative constrained multi-agent RL algorithm at the time of writing this paper. This method employs Lagrangian multipliers to embed cost constraints (6b) in the objective function and proximal policy optimization (PPO)-clip to replace the Kullback–Leibler (KL)-divergence constraint (6c) with the clip operator. More details of this method are not elaborated upon in this paper as it is beyond the scope of the current discussion.

IV. CMARS Algorithm

In this section, we present our CMARS algorithm for inter-slice resource allocation in RAN slicing using constrained

TABLE III: Summary of notation used in section III-D

Symbol	Definition
$\mathcal{N}$	set of agents
n	number of agents
$\mathcal{S}$	finite state space
$\mathcal{A}$	joint action space
$\mathcal{P}$	transition probability function
r	reward function
γ	discount factor
C	cost function set
c	cost thresholds
ρ^0	initial state distribution
δ	KL-divergence threshold
$\pi^i(a\|s)$	policy of agent i
θ^i	parameters of agent i's policy
$V_\pi(s)$	state value function
$Q_\pi(s,a)$	state-action value function
$A_\pi(s,a)$	advantage function
$\mathcal{J}_r(\pi)$	expected discounted total reward
$\mathcal{J}_c^i(\pi)$	expected discounted total cost of agent i

MARL. We provide a general abstraction of the method in section IV-A applicable to any RAN slicing setting, and its application to our system model (section III-C) is discussed in section IV-B. The RL model training algorithm is detailed in section IV-C. While we focus on two slice types (eMBB and mMTC) for simplicity, adding other slice types is straightforward without affecting CMARS performance, as shown in the evaluation section VI.

A. General Reinforcement Learning Model

Agents resource sharing. The decisions of the agents are mutually dependent as they consume a shared set of resources in RAN slicing. We propose a sequential decision-making scheme in every MDP step where each agent takes the available PRBs as input, allocates a portion of them to the corresponding slice instance, and passes the remaining PRBs to the next agent. This procedure is repeated for all agents and eventually forms the slices' share of PRBs as the action of the MDP. To avoid selfish behavior and order dependency, we employ a cooperative model with interrelated rewards. This allows us to penalize selfish actions that harm overall system performance and encourage collaboration among agents to achieve system objectives. Additionally, shuffling the agents' order in each step further mitigates order dependency.

State communication. An agent's decision should account for other slices' demands to avoid resource under- or over-provisioning. To facilitate this, when a slice policy is invoked, it is provided with aggregated statistics of upcoming slices. This aggregated information along with the slice local observations, forms the agent's *local state*. Notably, aggregation (rather than concatenation) is independent of slice counts which brings flexibility in slice accommodation.

Policy reuse among agents. To enable flexibility in slice accommodation, we utilize cooperative MARL schemes with each agent responsible for a specific slice. When a slice joins the network, a new agent is introduced to the sequential decision-making process. This agent can reuse a pre-trained policy to avoid retaining a new model from scratch. The agent is finally withdrawn once the corresponding slice departs the network. Remarkably, distinct policies are used for various slice types due to their differing SLA-related goals and behaviors. However, instances of the same slice type can share a policy to expedite learning by reducing the overall problem's parameters and improve sample efficiency by reusing environment interaction samples.

Instances of the same slice type might expect different SLA levels (refer to section III-C). In this regard, we introduce a new parameter into the set of slice local observations. This parameter communicates information about the required SLA level to the policy. Consequently, the policy becomes capable of distinguishing between various expected SLA levels and their resource requirements. As such, even though a common policy is applied to slices of the same type, the associated state variable will assume distinct values corresponding to the SLA level.

Non-stationarity problem. In MARL scenarios, agents' policies evolve over training which makes the environment non-stationary to each agent. This phenomenon may lead to an infinite cycle of adapting to other agents' policies. One common method to deal with this issue is leveraging a *global critic* in an actor-critic setting. This critic is aware of all agents' observations while the actor only has access to the observation of one agent. Importantly, no overhead is associated with global critic employment during the inference phase since the agent's actions are determined by the actor while the critic is only used in training [38].

A concatenation of agents' local observations can be fed to the global critic, but this is not suitable for our case. Precisely, we assume fluctuation in the number of slices which in turn leads to the variation in the critic input size, adding to the difficulty of training. Instead, we compute aggregated statistics of agents' local observations as the *global state*. Such aggregation solves the problem of variation in the critic's input size as it remains independent of the number of agents.

An overview of the proposed cooperative MARL framework is depicted in Fig. 2 which involves four slice instances: two eMBB and two mMTC. Agents are created for each slice type, processing the instances sequentially. Agents receive available PRBs, slice statistics, and aggregated statistics of other slices as input. They allocate PRBs to the corresponding slice and pass the remaining PRBs to the next agent. We also shuffle the sequence of agents in every step to avoid order dependency in decision-making.

B. Constrained Multi-agent Markov Decision Process

In this section, we apply our general model to the RAN slicing problem (see section III-C) and define the corresponding CMMDP. The decision-making process is described in Algorithm 1.

Local state. As mentioned in section IV-A, we compute count-independent aggregated metrics to inform an agent about the demand of subsequent slices. These metrics include the number of mMTC and eMBB slices, the average number of users in mMTC slices, average GBR traffic in eMBB slices, average non-GBR traffic in eMBB slices, and the number of remaining PRBs. These metrics are concatenated with the local observation of each slice and the slice's expected SLA level to derive the agent's local state (see Alg. 1 Line 8).

Global state. Statistical features of different slices are aggregated to provide the global critic with a global state

Algorithm 1: Per Step Decision Making (PSDM)

Input: slice observation o_s^k, $\forall s \in \mathcal{S}_k$, number of PRBs $\mathcal{M}$, policies $\{\pi_{\text{mmtc}}, \pi_{\text{embb}}\}$
Output: slice allocated PRBs R_k^s, slice local state ls_s, $\forall s \in \mathcal{S}_k$, slice global state gs
Initialization:
1 $S_{\text{mmtc}} \leftarrow \{s \in \mathcal{S}_k : \text{s is of mMTC type}\}$
2 $S_{\text{embb}} \leftarrow \{s \in \mathcal{S}_k : \text{s is of eMBB type}\}$
3 Aggregated observations of mMTC: $o_{\text{mmtc}}^{\text{agg}} \leftarrow \{|S_{\text{mmtc}}|, \min_{s\in S_{\text{mmtc}}} o_s^k, \max_{s\in S_{\text{mmtc}}} o_s^k, \text{mean}_{s\in S_{\text{mmtc}}} o_s^k\}$
4 Aggregated observations of eMBB: $o_{\text{embb}}^{\text{agg}} \leftarrow \{|S_{\text{embb}}|, \min_{s\in S_{\text{embb}}} o_s^k, \max_{s\in S_{\text{embb}}} o_s^k, \text{mean}_{s\in S_{\text{embb}}} o_s^k\}$
5 global state: $gs \leftarrow o_{\text{mmtc}}^{\text{agg}} \cup o_{\text{embb}}^{\text{agg}}$
6 local augmentation: $o^+ \leftarrow \{$ slices count and users count mean of $o_{\text{mmtc}}^{\text{agg}}$, slices count, GBR traffic mean, and non-GBR traffic mean of $o_{\text{embb}}^{\text{agg}}$, and $\mathcal{M}\}$
Decision making:
7 **for** $s \in \text{Shuffle}(\mathcal{S}_k)$ **do**
8 local state: $ls_s \leftarrow o_s^k \cup o^+ \cup \{$expected SLA level of slice $s\}$
9 **if** $s \in S_{\text{mmtc}}$ **then** $R_k^s \leftarrow \pi_{\text{mmtc}}(ls_s)$
10 **else if** $s \in S_{\text{embb}}$ **then** $R_k^s \leftarrow \pi_{\text{embb}}(ls_s)$
11 Update o^+ by excluding observation o_s^k and resource allocation R_k^s

that supports flexibility in the number of slices. These statistics include the number of instances per slice type along with the minimum, maximum, and average of their local states (Alg. 1, Line 5).

Action. An agent determines the allocation of PRBs to its slice (Alg. 1, Line 9 and Line 10). The agent's available actions are determined by the number of PRBs remaining after previous agents' decisions in the sequential process (Alg. 1, Line 11). The collective decisions of all agents form the action for the MDP step. To prevent order dependency, the order of agents is randomly shuffled in each step (Alg. 1, Line 7).

Reward and Cost. In cooperative MARL, agents cooperate with each other towards a common goal that is formalized in terms of cost and reward functions. We define the cost and reward functions of agent s^* in step k as follows:

$$\text{cost}_k^{s^*} = 0.4 \times v_k^{s^*} + 0.6 \times \Sigma_{s\in\mathcal{S}_k\setminus s^*} v_k^s, \tag{7}$$

$$\text{reward}_k^{s^*} = 0.8 \times (u_k^{s^*}) + 1.2 \times \underset{s\in\mathcal{S}_k\setminus s^*}{\text{mean}} (u_k^s) - 1 \tag{8}$$

Here, v_k^s indicates if SLAs of slice s are violated in step k (as defined in section III-C), and u_k^s is PRB utilization ratio, *i.e.*, the number of used PRBs over the number of allocated PRBs, of slice s in step k. The reward and cost functions reflect the performance of all slices, with a higher coefficient assigned to the corresponding slice's performance metric, so that the agent can better understand the impact of its own actions (the exact value of the coefficients is determined by experimentation). Besides, we subtract 1 from the summation in the reward function to normalize the range of values from -1 to 1 for reward-scaling purposes [39].

C. CMARS Algorithm

Now that all the involved components have been introduced, we explain the training procedure summarized in Alg. 2. Lines 1–11 show an iteration of the training algorithm. In each iteration, an episode of interactions with the environment takes place as shown in Lines 2–5, and agents' policies are then updated according to the MAPPOL scheme in Lines 6–11.

The policy consists of a global V-value, two actors, and two V-cost functions based on a critic network ψ, two actor networks $\{\pi_{\text{mmtc}}, \pi_{\text{embb}}\}$, and another two cost networks $\{\phi_{\text{mmtc}}^0, \phi_{\text{embb}}^0\}$, respectively. According to the type of slice the agent is associated with, the corresponding set of parameters will get updated when it is the agent's turn. The detailed

Algorithm 2: Training

Input: number of available PRBs $\mathcal{M}$, episode length L,
Output: policies π_{mmtc} and π_{embb}
Initialization: replay buffer $\mathcal{B}$, resource allocation R^0, policies π_{mmtc}^0 and π_{embb}^0 (consists of actor networks θ_{mmtc}^0 and θ_{embb}^0, global critic networks ψ^0, and cost networks ϕ_{mmtc}^0 and ϕ_{embb}^0)
Training:
1 **while** π_{mmtc} and π_{embb} not converged **do**
2 **for** $k \leftarrow 0$ to $L-1$ **do**
3 Get observations $o^k \leftarrow \{o_s^k : \forall s \in \mathcal{S}_k\}$, reward r^k, and cost c^k from environment
4 $R^{k+1}, ls^k, gs^k \leftarrow \text{PSDM}(o^k, \mathcal{M}, \pi_{\text{mmtc}}^k, \pi_{\text{embb}}^k)$
5 Add slice s experience to $\mathcal{B}^s$
6 Sample a random minibatch B^s from $\mathcal{B}, \forall s \in \mathcal{S}_k$
7 Compute advantage function $\hat{A}(ls, gs, R)$ based on global critic network
8 Compute cost advantage functions $\hat{A}_s(ls, gs, R_s)$ based on slice type cost network
9 **for** $s \in \text{Shuffle}(\mathcal{S}_k)$ **do**
10 **if** $s \in S_{\text{mmtc}}$ **then** $\text{PU}(\pi_{\text{mmtc}}, B^s, \hat{A}(ls, gs, R), \hat{A}_s(ls, gs, R_s))$
11 **else if** $s \in S_{\text{embb}}$ **then** $\text{PU}(\pi_{\text{embb}}, B^s, \hat{A}(ls, gs, R), \hat{A}_s(ls, gs, R_s))$

Algorithm 3: Policy Update (PU)

Input: old policy π_{old} (consist of actor-network θ_{old}, global critic network ψ_{old}, and cost network ϕ_{old}), mini-batch B, advantage function $\hat{A}(ls, gs, R)$, slice cost advantage function $\hat{A}_s(ls, gs, R_s)$, PPO update count e_{ppo}, PPO policy change threshold κ, step sizes $\alpha_\theta, \alpha_\lambda$, discount factor γ, cost constraint threshold c
Output: policy π_{new} including $\theta_{\text{new}}, \psi_{\text{new}}, \phi_{\text{new}}$
Initialization: Lagrangian multiplier λ, Lagrangian modification step of objective construction M_s
Updating:
1 **for** $e \leftarrow 1$ to e_{ppo} **do**
2 Differentiate the Lagrangian PPO-clip objective $\Delta_\theta \leftarrow \nabla_\theta \frac{1}{B} \sum_{b=1}^{B} \sum_{t=1}^{L} \min(\frac{\pi_\theta(a_t|ls_t)}{\pi_{\theta_{old}}(a_t|ls_t)} M_s, \text{clip}\left(\frac{\pi_\theta(a_t|ls_t)}{\pi_{\theta_{old}}(a_t|ls_t)} M_s, 1 \pm \kappa\right))$
3 Update actor parameters $\theta \leftarrow \theta + \alpha_\theta \Delta_\theta$
4 Approximate constraint violation $d \leftarrow \frac{1}{B.L} \sum_{b=1}^{B} \sum_{t=1}^{L} \psi(gs_t) - c$
5 Differentiate constraint $\Delta\lambda \leftarrow -\frac{1}{B} \sum_{b=1}^{B} \left(d(1-\gamma) + \sum_{t=0}^{L} \frac{\pi_\theta(a_t|ls_t)}{\pi_{\theta_{old}}(a_t|ls_t)} \hat{A}_s(ls, gs, R_s)\right)$
6 Update Lagrangian multiplier $\lambda \leftarrow \text{ReLU}(\lambda - \alpha_\lambda \Delta\lambda)$
7 Update actor network, global critic network, and cost network:
8 $\theta_{\text{new}} \leftarrow \theta$
9 $\psi_{\text{new}} \leftarrow \text{argmin}_\psi \frac{1}{B.L} \sum_{b=1}^{B} \sum_{t=1}^{L} (V_\psi(gs) - r_t)^2$
10 $\phi_{\text{new}} \leftarrow \text{argmin}_\phi \frac{1}{B.L} \sum_{b=1}^{B} \sum_{t=1}^{L} (V_\phi(gs) - c_t)^2$

procedure of CMARS policy updating is presented in Alg. 3, inspired by [17].

Specifically, Lagrangian multipliers and actor-network parameters are updated multiple times in Lines 1–6. The PPO-clip in Line 2 takes care of the KL-divergence of the new policy from the previous policy while the probability of constraint violation is considered in updating the multipliers as shown in Line 5. Subsequently, critic and cost networks are updated in Lines 8–10 based on the received reward and cost values.

V. Integration of CMARS in Open RAN Architecture

In this section, we discuss how CMARS is used for RAN slicing in Open RAN architecture. Open RAN refers to the architectural concept in the industry that promotes the openness of RAN [40]. We specifically consider the Open RAN specifications proposed by the O-RAN alliance. This alliance, consisting of operators and vendors, actively contributes to defining RAN specifications that complement the existing 3GPP standards to realize Open RAN [41].

Near-real-time RAN Slicing. We envision deploying CMARS using RAN intelligent controllers (RICs) [9]. CMARS functions as a centralized RAN slicing xApp running on top of the near-real-time (Near-RT) RIC [42]. The xApp communicates with the RAN data plane through service models like KPM [43] over the E2 interfaces to receive metrics. Resource allocation decisions are made by the xApp and transmitted to the data plane using appropriate service models [44]. In accordance with O-RAN specifications [45], trained CMARS models are stored in a catalog within the non-real-time (Non-RT) RIC [46]. These models are retrieved from the Non-RT RIC via the A1 interface for use in the corresponding xApp within the Near-RT RIC.

CMARS Training. RL agents can undergo training either offline or online prior to their deployment. Offline training involves supervised learning using a trace of state-action pairs, but its effectiveness depends on the trace generation algorithms [9]. Conversely, online training enables real-time learning through interactions with the environment, rendering it more versatile and adaptable to diverse scenarios. Such online training before deployment can be achieved through various tools, including a pilot testbed [9], digital twins [47], or a simulation environment [48]. We envision such an online training process for CMARS before its actual deployment.

Dynamic Slice Creation and Termination. We implemented policy-sharing for slices of the same type in our framework. A single policy per slice type is stored in Non-RT RIC which is carried over to the Near-RT RIC upon request and shared among different instances of that type. When a new slice is created, it is assigned the policy associated with its slice type, benefiting from the accumulated knowledge and experience of other instances. Our policy-sharing mechanism also facilitates the effective utilization of knowledge from terminated slices. Experiences gained by each instance contribute to the training of the corresponding policy, avoiding the need to start training from scratch. For online learning, experiences from distinct instances are utilized to continuously update the policy, leading to continuous improvement.

TABLE IV: Simulator settings [5], [49]

eMBB GBR traffic model	
UE arrival	Poisson process (2 user/min)
UE connection time	Exponential (mean = 30 sec)
Bit rate	0.5 Mb/s
eMBB non-GBR traffic model	
UE arrival	Poisson process (5 user/min)
UE connection time	Exponential (mean = 30 sec)
Burst arrivals	Poisson process (1 burst/min)
Burst length	Exponential (mean = 500 packets)
Burst bit rate	1 Mbps
mMTC traffic model	
mMTC devices	1000
Transmission periods	{10, 50, 100, 150, 250, 500, 1000} sec
Packet repetitions	{2, 4, 8, 16, 32, 64, 128}
Packet size	1000 bits
uRLLC non-GBR traffic model	
UE arrival	Poisson process (20 user/min)
UE connection time	Exponential (mean = 5 sec)
Burst arrivals	Poisson process (2 burst/min)
Burst length	Exponential (mean = 100 packets)
Burst bit rate	500 Kbps
Radio channel parameters	
Transmission power	30 dBm
BS antenna gain	15 dBi
BS antenna pattern	Section 4.2.1, TS 36.942 [50]
Cell range	2 Km
Noise figure	9 dB
Interference + noise	-110 dBm
Carrier frequency	2GHz
Propagation model	Macro cell urban [50]
Fading model	Pedestrian A, Typical Urban, Vehicular A [51]

TABLE V: Different SLA performance levels per slice type [5], [49] and CMARS parameters

eMBB slice SLA	**L1**	**L2**	**L3**
GBR authorized capacity (RBs/subframe)	16	22	30
non-GBR QoS compliant capacity (RBs/subframe)	20	27	35
Max. average queue per GBR user (Kb/UE)	380	220	60
Max. average queue per non-GBR user (Kb/UE)	1300	700	100
Min. average throughput per GBR user (Mbps)	0.4	1	6
Min. average throughput per non-GBR user (Mbps)	0.8	5	12
uRLLC slice SLA	**L1**	**L2**	**L3**
Max. average queue per non-GBR user (Kb/UE)	100	50	10
mMTC slice SLA	**L1**	**L2**	**L3**
Max. per user delay (ms)	400	300	200
CMARS Parameters			
Actor learning rate	8×10^{-5}		
Critic learning rate	1×10^{-3}		
Lagrangian coefficient	10		
Lagrangian learning rate	1×10^{-7}		
Safety bound	1×10^{-4}		
Gamma	0.4		

VI. Performance Evaluation

A. Simulation Environment Setting

We conducted experiments to compare the proposed method with state-of-the-art RL baselines using an extended Python-based RAN slicing simulation environment [49]. Particularly, the simulator was enhanced to incorporate the uRLLC slice type and dynamic creation and termination of slices. The experiments considered mMTC, eMBB, and uRLLC slice types, with their respective traffic characteristics detailed in Table IV. GBR and non-GBR users were modeled in eMBB slices using constant and variable bit rate flows, following a Poisson process (Table IV). Similarly, users in uRLLC slices follow a non-GBR traffic model but with different parameters compared to the non-GBR users in eMBB slices. mMTC slices consisted of 1000 users with transmission periods and packet repetitions

TABLE VI: Classes and scenarios used in experiments with a fixed number of slices.

	Class 1				**Class 2**				**Large-scale**			
Scenario	1	2	3	4	5	6	7	8	9	10	11	12
PRBs	15	30	40	80	30	60	80	200	90	140	200	270
eMBB-L1	0	0	1	1	0	0	1	1	2	2	2	2
eMBB-L2	1	1	1	1	1	1	1	1	1	1	2	2
eMBB-L3	0	0	0	0	0	0	0	0	2	2	1	1
mMTC-L1	0	0	1	1	0	0	1	1	3	3	2	2
mMTC-L2	1	1	1	1	1	1	1	1	2	2	2	2
mMTC-L3	0	0	1	1	0	0	1	1	2	2	2	2
uRLLC-L1	-	-	-	-	0	0	1	1	-	-	3	3
uRLLC-L2	-	-	-	-	1	1	1	1	-	-	2	2
uRLLC-L3	-	-	-	-	0	0	1	1	-	-	2	2

randomly chosen from predefined sets listed in Table IV.

The simulator incorporates NS-3-generated datasets with specific configurations to represent realistic channel conditions and user mobility patterns accurately. It utilizes the macro cell propagation model configuration for urban areas specified in TR 36.942 [50] to determine the nominal received power at the UE. Frequency-selective fading is introduced by drawing samples from datasets containing fading traces of moving UEs, following the fading/mobility models defined in TS 36.104 [51] (e.g., pedestrian A, typical urban, and vehicular A). The communication model parameters are included in Table IV. When a new UE is introduced, the simulator randomly selects a dataset and generates a random integer as the index to draw SNR samples. This approach ensures variability in channel conditions and captures the dynamic nature of wireless environments, thereby enhancing the validity of the simulation.

The simulation procedure models UE buffer management, MAC scheduling, modulation and coding scheme, and probabilistic bit transmission over the air. A proportional fair scheduler is employed within each slice to allocate PRBs among the UEs based on their buffer state reports and SNR estimations. Once the resources are allocated, the transmitter selects an appropriate modulation and coding scheme (MCS) aiming for a target block error rate (BLER) below 0.1, which determines the transport block size *i.e.*, the number of bits transmitted from the UE queue. Finally, actual bit transmission is stochastically determined based on the reception probability of the MCS scheme. Successful transmission deducts the corresponding number of transmitted bits from the UE's queue length. It is worth noting that while the NS-3 data remains static for a given run, the simulation enforces variability in channel conditions and UE buffer states.

The RAN slicing decision is periodically made on a time window of T time slots, where we assume $T = 50$ ms based on [5]. Table V presents SLAs for each slice type, categorized into performance levels L1, L2, and L3, with corresponding parameters used in CMARS.

B. Baseline Algorithms

Algorithms. We consider four RL methods as baseline algorithms: a model-based RL scheme named KBRL [5], two model-free single-agent RL methods, proximal policy optimization (PPO) [5], [7], [8], twin delayed DDPG (TD3) [5], [52], and a multi-agent RL approach that we call MARS (Multi-agent RAN Slicing). MARS shares fundamental characteristics with CMARS, albeit without employing a constrained RL approach.

KBRL is the state-of-the-art RAN slicing RL-based algorithm that utilizes classifiers predicting whether the SLA of a given slice is violated for any specific state-action pair. TD3 is an off-policy RL method that offers a high sample efficiency by maintaining separate policies for development and sample generation. PPO is an on-policy RL algorithm that updates the agent's policy in a controlled manner to achieve a stable performance improvement. We also investigated the advantage actor-critic RL algorithm, the predecessor of TD3, and constrained policy optimization, however, the results were not significant for inclusion. Furthermore, we specifically designed MARS as a policy-based cooperative trust region policy optimization (TRPO) multi-agent RL algorithm, inspired by the work of [37]. MARS incorporates sequential decision-making and policy sharing similar to CMARS. The name "MARS" emphasizes its similarity to CMARS while distinguishing its lack of constrained RL approach.

MDP. Regarding the single-agent RL baselines (PPO and TD3), the underlying MDP is formulated as follows: the *state* is constructed by concatenating the local observations of various slices, as defined in section III-C. These local observations are the same as those used in CMARS. The *action* space consists of different choices for resource allocation per slice based on the total available resources. The *reward* function as step k is defined as follows:

$$\text{reward}_k^{\text{single}} = \begin{cases} -1000 \times \sum_{s \in \mathcal{S}_k} v_k^s & \text{if } \sum_{s \in \mathcal{S}_k} v_k^s > 0, \\ \text{unallocated PRB count} & \text{otherwise.} \end{cases} \tag{9}$$

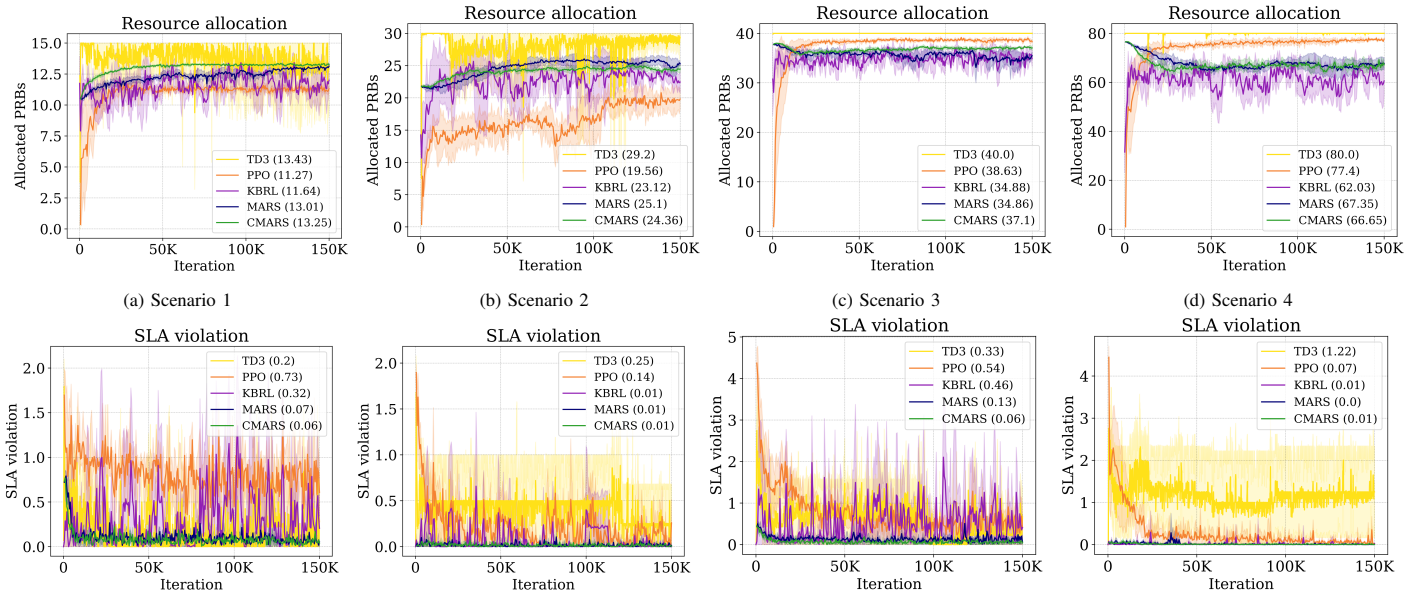
The local and global *state* functions of MARS, as well as the *action* space, are the same as those of CMARS. However, the *reward* function is formulated differently, taking inspiration from CMARS.

$$\text{reward}_k^{\text{MARS},s} = \begin{cases} -1000 \times \sum_{s \in \mathcal{S}_k} v_k^s & \text{if } \sum_{s \in \mathcal{S}_k} v_k^s > 0, \\ 0.5 \times \text{mean}_{s \in \mathcal{S}_k}(u_k^s) & \text{otherwise.} \\ +0.5 \times \min_{s \in \mathcal{S}_k}(u_k^s) & \end{cases} \tag{10}$$

We acknowledge that the reward functions are not exactly consistent across baselines and CMARS. However, we underscore the general consistency among them in the sense that notions of both resource utilization and SLA satisfaction are incorporated in all baselines, aligned with the literature [13]. Indeed, our principal objective is to obtain the best possible performance for each baseline. This impelled us to systematically experiment with diverse reward function formulations and decide on the best reward function based on empirical evaluations. This variance in reward function definitions stems from the inherent structural distinctions between different frameworks of single-agent, multi-agent, and constrained multi-agent reinforcement learning.

Implementation. The KBRL algorithm was implemented using the original paper's source code [49], while PPO and TD3 were implemented using *stable baselines* framework [53]. We conducted each experiment 4 times with different random seeds to mitigate randomness. The performance comparison of algorithms focuses on the number of allocated PRBs and SLA violations. The average value and standard deviation range are presented based on multiple runs and 500-step episodes. The following sections compare the performance of the algorithms

This article has been accepted for publication in IEEE Journal on Selected Areas in Communications. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2023.3336156



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3 (d) Scenario 4

(e) Scenario 1 (f) Scenario 2 (g) Scenario 3 (h) Scenario 4

Fig. 3: Performance of algorithms in **Class 1** scenarios with a fixed number of slices, showing allocated PRBs and SLA violations. The average of each curve over the last 10k time steps is included in front of each algorithm on the legend box.

in the training phase with a fixed number of slices and in the inference phase with a dynamic number of slices.

C. Learning with a Fixed Number of Slices

This section compares the performance of different algorithms during the training phase with a fixed number of slices. Two reasons motivate this: first, to allow a fair comparison with previous literature that assumes a fixed number of slices in both training and inference; second, to prepare our model for inference in scenarios with a dynamic number of slices. In other words, our model is capable of inference in scenarios with a dynamic number of slices by training only with a fixed number of slices (we delve into more detail in the following subsection).

Table VI presents the configuration of various scenarios with a fixed number of slices, providing information on the number of PRBs and slice instances for each performance level (L1, L2, and L3) defined in Table V per slice type. In **Class 1** scenarios, we focus on the impact of slice instances and available PRBs on the performance of the algorithms, considering eMBB and mMTC slices since the KBRL algorithm does not originally support uRLLC [5]. In **Class 2** scenarios, we extend the analysis to include uRLLC slices and examine the same influencing factors. Finally, we evaluate the scalability of the algorithms in **Large-scale** scenarios, which involve a larger number of slices. Moreover, to highlight the impact of resource availability, pairs of similar scenarios are defined in each class. These pairs are only different in the number of available resources while the slice count is the same.

Class 1 experiments. The results of these scenarios are shown in Fig. 3. In this class, CMARS improves state-of-the-art by violating 41% fewer SLAs while consuming only 6% more resources. Across all scenarios, MARS and CMARS consistently yield the least amount of SLA violations. Both of these algorithms employ cooperative multi-agent RL approaches with sequential decision-making and policy sharing. It is noteworthy that the incorporation of constrained RL strategies in CMARS provides advantages that are significantly pronounced in other scenarios involving large-scale and dynamic slicing as discussed later.

KBRL, PPO, and TD3 not only result in higher levels of violations with significant fluctuations, but they are also highly sensitive to the availability of resources. The performance of KBRL is mainly determined by the number of available PRBs. Particularly, it violates 6 times more SLAs on average in scenarios 1 and 3 while giving closer performance to CMARS in scenarios 2 and 4. Moreover, the performance of KBRL does not improve over time which shows its limited learning capabilities. We hypothesize that these deficiencies are rooted in the fact that KBRL only considers immediate rewards in one time step rather than optimizing for a long-term summation of rewards. In contrast, MARS and CMARS account for future rewards which increases their learning capacity. In particular, they initially exhibit similar performance to KBRL but gradually improve to satisfy more slice SLAs while consuming fewer PRBs.

PPO causes 10 times higher SLA violations compared to CMARS in scenarios 1 and 2 while there are unallocated resources in the systems that could have been utilized to decrease SLA violation. PPO not only allocates 8% more resources than CMARS on average in scenarios 3 and 4, but it also gives 6 times more SLA violations. TD3 exhibits the highest resource consumption in this class with significant fluctuations in SLA satisfaction, indicating unsatisfactory performance. The poor performance of these single-agent RL algorithms can be attributed to the increased complexity arising from the consolidating slice decision-making within a single agent, in contrast to the distribution of this task among multiple

agents in KBRL, MARS, and CMARS.

Class 2 experiments. Fig. 4 depicts the results of the four scenarios in this class, investigating the impact of introducing a new slice type (uRLLC) on the algorithms' performance. CMARS surpasses state-of-the-art in these scenarios, significantly reducing SLA violations by 80% while experiencing only 20% increase in resource consumption. KBRL exhibits strong initial performance in scenarios 6 and 7 but experiences a notable decline in SLA satisfaction subsequently. This phenomenon, absent in **Class 1**, leads us to infer that KBRL encounters difficulties when novel slice types are introduced to the network. We speculate that this issue is the result of a lack of state communication across learners of different slices. Similarly, MARS gives a non-stable SLA satisfaction ratio in the final stages of scenario 8. Generally, MARS consumes slightly fewer resources compared to CMARS by 16.7% but at the cost of 43% higher SLA violations over all scenarios in this class. These observations highlight the effectiveness of constrained RL approaches within CMARS in dealing with variant slice types as CMARS separates the objectives of decreasing resource consumption and increasing SLA satisfaction. The effect of the available resources on the algorithms' performance follows the same trend as in **Class 1** scenarios.

Large-scale experiments. We consider four scenarios with a large number of slices (refer to Table VI) to evaluate the scalability of CMARS compared to the baselines. The results of this set of experiments are depicted in Fig. 5. Generally, CMARS starts with a number of SLA violations in the early stages of training but converges to a negligible number while consuming a modest amount of resources compared to the baselines. CMARS consistently maintains its performance even when uRLLC slices are introduced in scenarios 11 and 12. On average, CMARS enhances state-of-the-art by violating 50% less SLAs while consuming 9% more resources across all scenarios. The scalability of CMARS is the result of three design decisions: (i) policy sharing across slice instances of the same type that reduces the overall number of problem parameters and increases sample efficiency by reusing agents' interactions with the environment, (ii) separating the objectives of resource consumption and SLA violations using reward and cost functions following a constrained RL approach, (iii) and the fact that the state-action space size is not proportional to the number of slices, addressing the curse of dimensionality.

Although PPO and TD3 allocate all the available PRBs to slices, they fail to satisfy slice SLAs compared to other algorithms. The challenge posed by high dimensionality is apparent in these two algorithms, stemming from the concentration of resource allocation decisions for all slices within a single agent. Additionally, KBRL is not able to properly utilize the available resources in the network in favor of SLA satisfaction. In particular, KBRL gives a significantly poor performance in scenarios with scarce resources (Scenarios 9 and 11). This occurs because KBRL completely decouples the resource allocation of slices from each other, *i.e.* each KBRL learner independently decides the resource allocation for its corresponding slice, disregarding the state of other slices. KBRL also fails to account for long-term performance by focusing only on immediate rewards. Furthermore, although the SLA satisfaction performance of MARS is close to CMARS in scenarios with abundant resources (Scenarios 10 and 12), it gives a really high SLA satisfaction fluctuation in scarce-resources scenarios. This can be attributed to the superiority of the constrained RL approach as explained before. Notably, the advantage of constrained RL methods was less apparent in the previous, implying that constrained RL holds greater significance in more intricate situations, such as large-scale slicing.

Interestingly, during the initial SLA violation period, CMARS utilizes all the resources to improve SLA satisfaction. After the SLA violations ratio reaches an acceptable level, CMARS reduces the PRB allocation while keeping the SLA violations near zero. This behavior is important in online training as it improves the existing policy, avoiding costly explorations that are usual in classic RL methods. It also suggests that SLA violations in the early stages can be mitigated by leveraging pre-trained models instead of training from scratch, as we show later in VI-D.

The training process in this particular class necessitates a substantial number of iterations. Nonetheless, we posit that the extended duration of training should not be regarded as a substantial apprehension for the proposed solution. Remarkably, the training process is executed prior to the actual deployment of the RL model. Consequently, the temporal demands of training do not bear any influence on the performance of the deployed system. Moreover, the model is meticulously designed to be able to adapt to variations in the number of slices without necessitating a retraining process. This confers a noteworthy advantage over alternative approaches, which typically mandate retraining whenever adjustments are made to the number of slices.

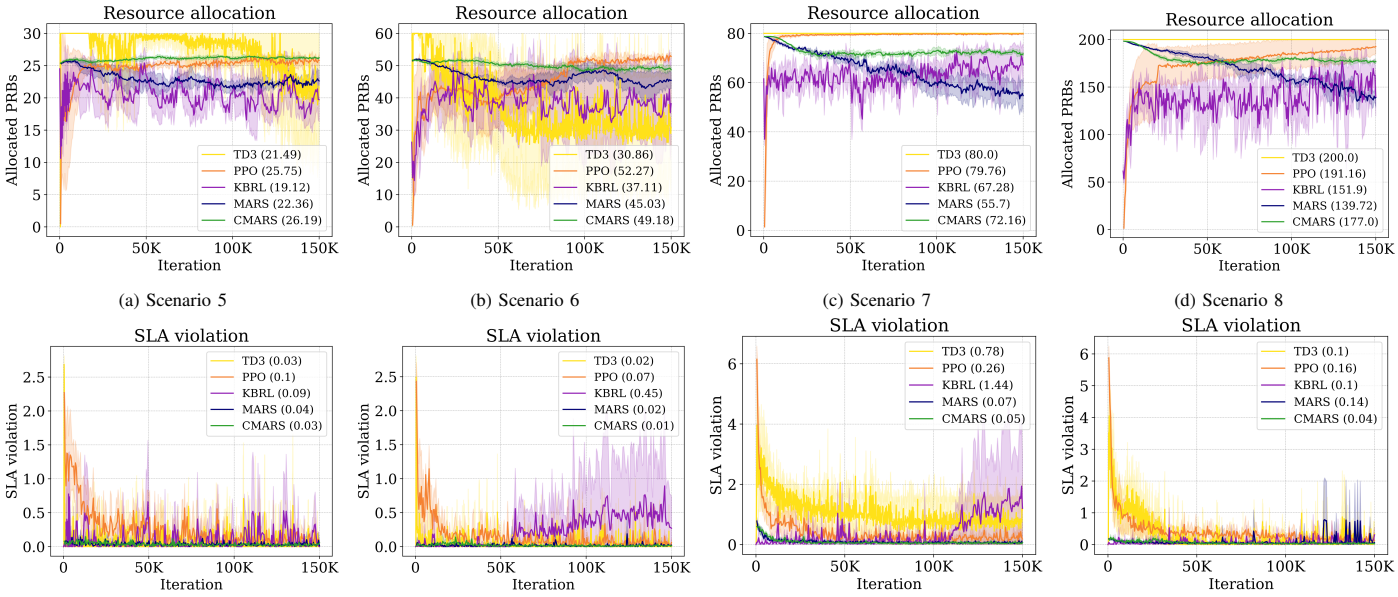
D. Inference with a Dynamic Number of Slices

In this section, we assess the performance of the trained CMARS policies without further training in scenarios involving a varying number of slices. By training CMARS with a fixed number of slices, we achieve notable computational efficiency when applying it to scenarios with a flexible number of slices. If the model were to undergo training on all different slice count permutations that are possible in deployment scenarios, the training duration and resource requirements would grow exponentially, rendering such an approach impractical.

We also employ previously trained TD3, PPO, and MARS models as baseline in this section. These models are not further trained for a fair comparison with CMARS. However, the KBRL algorithm is particularly allowed for online learning according to the guidelines of the original paper [5]. We explore two variants of KBRL in this section: one where the agents start learning from scratch (KBRL) and another where pre-trained learners offer a warm start (KBRL-PRET).

Both the CMARS and MARS state variables are independent of the number of slices. This independence arises from aggregating the statistics of various slices to be used as the state (refer to section IV-A). In KBRL, each slice has a dedicated learner only to which the corresponding slice's statistics are fed.

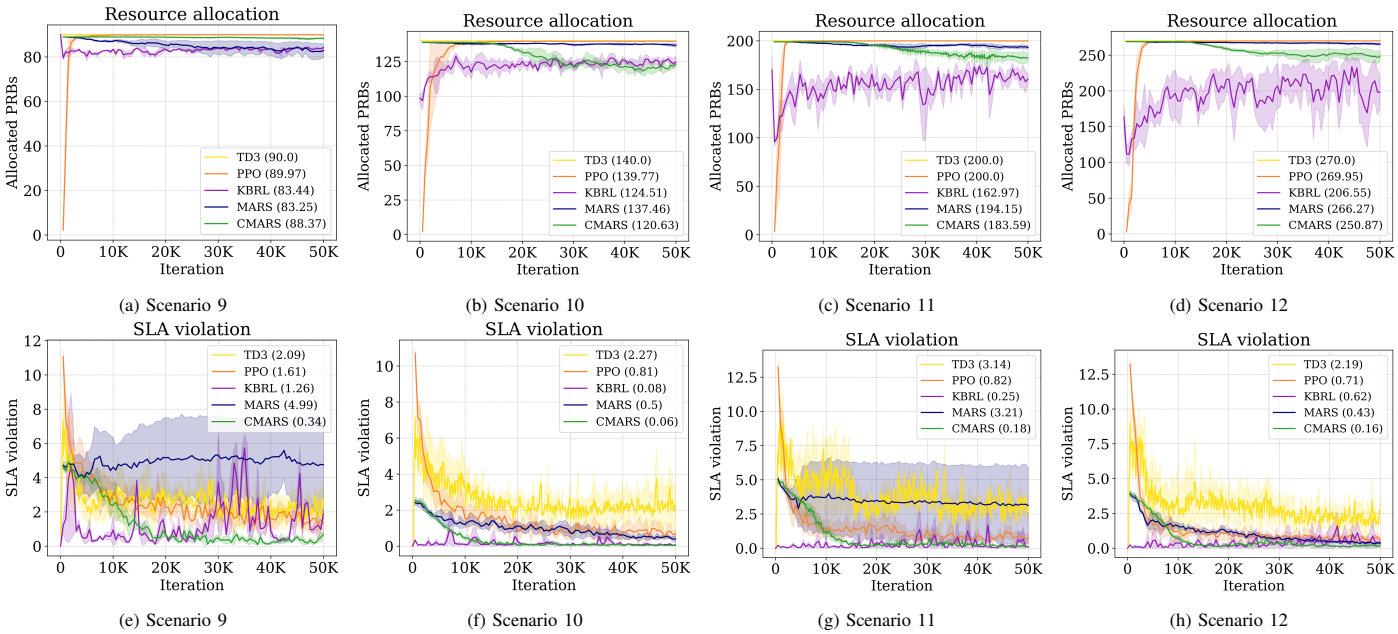
This article has been accepted for publication in IEEE Journal on Selected Areas in Communications. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2023.3336156



(a) Scenario 5 (b) Scenario 6 (c) Scenario 7 (d) Scenario 8

(e) Scenario 5 (f) Scenario 6 (g) Scenario 7 (h) Scenario 8

Fig. 4: Performance of algorithms in **Class 2** scenarios with a fixed number of slices, showing allocated PRBs and SLA violations. The average of each curve over the last 10k time steps is included in front of each algorithm on the legend box.



(a) Scenario 9 (b) Scenario 10 (c) Scenario 11 (d) Scenario 12

(e) Scenario 9 (f) Scenario 10 (g) Scenario 11 (h) Scenario 12

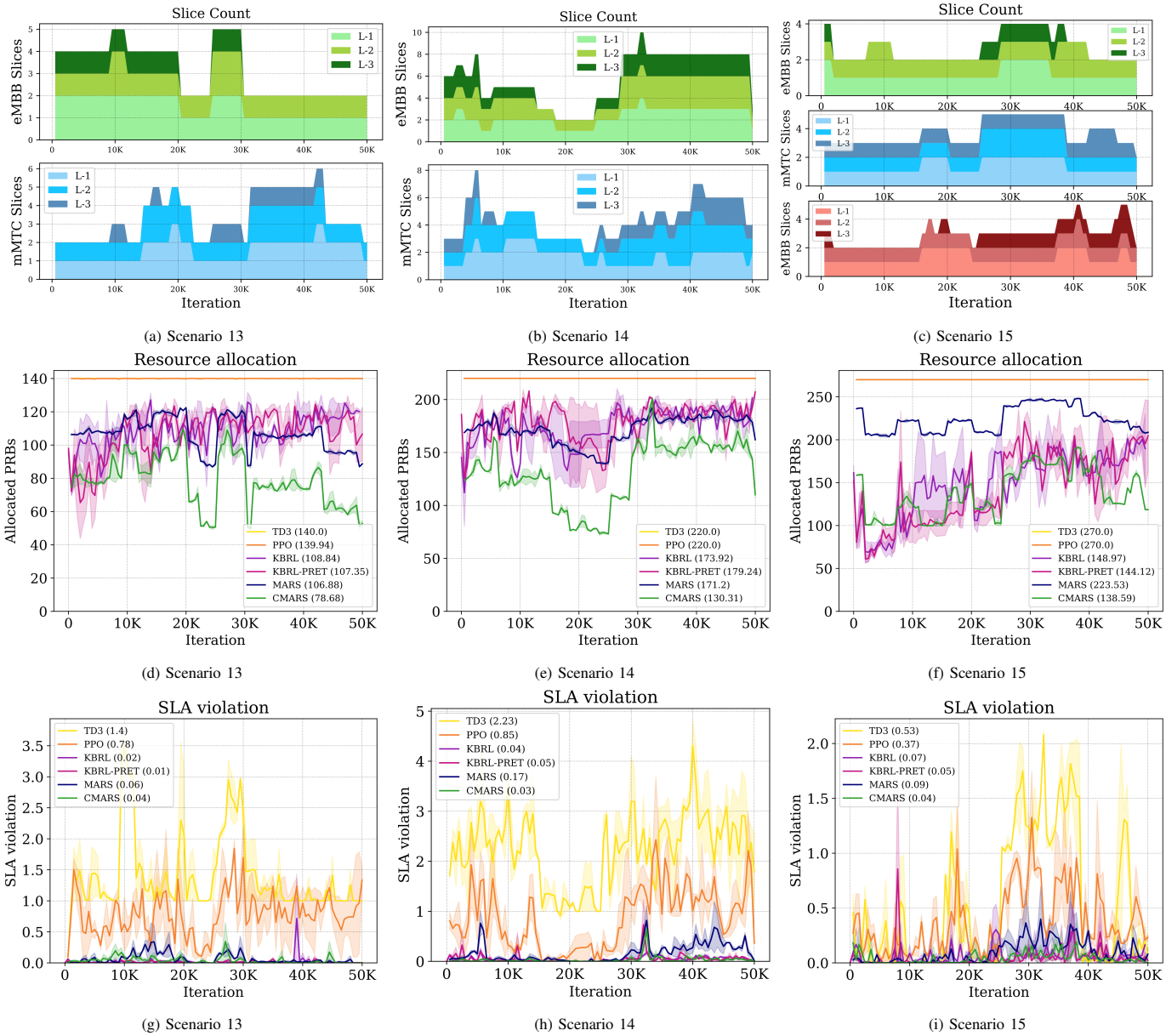
Fig. 5: Performance of algorithms in **Large-scale** scenarios with a fixed number of slices, showing allocated PRBs and SLA violations. The average of each curve over the last 10k time steps is included in front of each algorithm on the legend box.

In these multi-agent RL approaches, when a new slice joins the network, we introduce the corresponding agent/learner to the system. Conversely, when a slice departs, its corresponding agent/learner is removed. For single-agent algorithms, we consider the network's maximum potential number of slices and train a model accordingly. This approach involves concatenating the statistics of different slices into the agent's state variable. When a slice is absent from the system, the relevant positions in the state variable are populated with zeros.

Fig, 6a, 6b, and 6c show the number of slices present in the system per slice type and SLA level. Slices of mMTC and eMBB types arrive in the system following a Poisson distribution with a rate of 1 slice over 4000, 1500, and 1000 steps in scenarios 13, 14, and 15, respectively. The corresponding slices' lifetime duration follows an exponential distribution with an average of $10K$, $15K$, and $30K$ steps. Moreover, uRLLC slices are considered in scenario 15 where they join the network with a rate of 1 over 3500 steps and stay alive with an average of $15K$ steps.

CMARS demonstrates superior performance compared to the baselines in terms of both total PRB allocation and ensuring SLA satisfaction in Fig. 6. PPO and TD3 allocate almost a fixed

This article has been accepted for publication in IEEE Journal on Selected Areas in Communications. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2023.3336156



(a) Scenario 13 (b) Scenario 14 (c) Scenario 15

(d) Scenario 13 (e) Scenario 14 (f) Scenario 15

(g) Scenario 13 (h) Scenario 14 (i) Scenario 15

Fig. 6: Slice count per type and SLA level along with the performance of algorithms in scenarios with a dynamic number of slices, showing allocated PRBs and SLA violations. On the legend box, we included the average of the corresponding curve over the entire 50k time steps in front of each algorithm.

number of PRBs regardless of changes in the number of slices, consuming almost all available PRBs in the system. However, we observe that CMARS, MARS, KBRL, and KBRL-PRET allocate varying numbers of PRBs correlated with the total number of slices in the system. For example, when the number of slices increases in step $27,000$ of scenario 13 (6b), CMARS adapts itself by allocating more PRBs to serve the new slices (Fig. 6e). As such, we conclude that single-agent RL methods are not suitable for accommodating flexibility in the number of slices.

While all multi-agent algorithms demonstrate close performance in satisfying a high ratio of SLAs, CMARS outperforms the others by a narrow margin of 5% (refer to Fig. 6g, 6h, and 6i). On the other hand, CMARS exhibits superior resource utilization efficiency, outperforming the baselines by an average of 27% in scenarios 13 and 14, and by 6% in scenario 15.

KBRL suffers from over-allocation of resources although it trains online in these dynamic slicing scenarios. While utilizing pre-trained learners in KBRL-PRET slightly improves performance, it still fails in competing with CMARS in resource efficiency. Furthermore, MARS tries to avoid SLA violations by over-allocating resources since any SLA violation will cause huge negative rewards (as defined in its reward function). We posit that MARS's resource inefficiency is rooted in its challenge of balancing the intrinsic trade-off between increasing resource utilization and minimizing SLA violations as it relies on unconstrained RL.

VII. Conclusion

In this paper, we proposed a multi-agent RL-based RAN slicing approach for the efficient distribution of radio resources

among slices in scenarios with variable and large numbers of slices. Our approach involves experience reuse among agents that sequentially take action based on aggregated statistics of slices. We also employed a constrained RL algorithm to explicitly consider slice service requirements. Simulation results showed that CMARS outperforms state-of-the-art RL methods in SLA satisfaction by an average of 50% while consuming only 9% more resources in large-scale slicing scenarios. Furthermore, CMARS demonstrates a negligible level of SLA violations, on average 8% fewer than state-of-the-art methods, but with an average resource consumption that is 19% lower in scenarios with a dynamic number of slices.

Future directions for this work include adding an intelligent admission control module, as performance guarantees may not be possible for an unexpectedly large number of slices given finite resources. Furthermore, the current version of CMARS uses a simple aggregation function for slice metrics, but advanced embedding techniques such as variable input size auto-encoders can be employed to improve aggregation effectiveness.

REFERENCES

[1] I. Afolabi, T. Taleb *et al.*, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2429–2453, 2018.

[2] H. Zhou, M. Elsayed *et al.*, "RAN resource slicing in 5G using multi-agent correlated Q-learning," in *32nd International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2021, pp. 1179–1184.

[3] R. Li, C. Wang *et al.*, "The LSTM-based advantage actor-critic learning for resource management in network slicing with user mobility," *IEEE Communications Letters*, vol. 24, no. 9, pp. 2005–2009, 2020.

[4] T. Wang and S. Wang, "Online convex optimization for efficient and robust inter-slice radio resource management," *IEEE Transactions on Communications*, vol. 69, no. 9, pp. 6050–6062, 2021.

[5] J. J. Alcaraz, F. Losilla *et al.*, "Model-based reinforcement learning with kernels for resource allocation in RAN slices," *IEEE Transactions on Wireless Communications*, 2022.

[6] ORAN Alliance, "O-RAN: Towards an Open and Smart RAN," White Paper, October 2018.

[7] Y. Liu, J. Ding *et al.*, "A constrained reinforcement learning based approach for network slicing," in *IEEE 28th International Conference on Network Protocols (ICNP)*, 2020, pp. 1–6.

[8] A. M. Nagib, H. Abou-Zeid *et al.*, "Transfer learning-based accelerated deep reinforcement learning for 5G RAN slicing," in *IEEE 46th Conference on Local Computer Networks (LCN)*, 2021, pp. 249–256.

[9] M. Polese, L. Bonati *et al.*, "ColO-RAN: Developing machine learning-based xApps for open RAN closed-loop control on programmable experimental platforms," *IEEE Transactions on Mobile Computing*, 2022.

[10] Q. Liu, N. Choi *et al.*, "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.

[11] Y. Abiko, T. Saito *et al.*, "Flexible resource block allocation to multiple slices for radio access network slicing using deep reinforcement learning," *IEEE Access*, vol. 8, pp. 68 183–68 198, 2020.

[12] J. Mei, X. Wang *et al.*, "Intelligent radio access network slicing for service provisioning in 6G: A hierarchical deep reinforcement learning approach," *Transactions on Communications*, 2021.

[13] M. Zangooei, N. Saha *et al.*, "Reinforcement learning for radio resource management in ran slicing: A survey," *IEEE Communications Magazine*, pp. 1–7, 2022.

[14] Y. Hua, R. Li *et al.*, "GAN-powered deep distributional reinforcement learning for resource management in network slicing," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 334–349, 2019.

[15] O.-R. W. G. 1, "Slicing Architecture Technical Specification 9.0," March 2023.

[16] ——, "O-RAN Use Cases Analysis Report 10.0," March 2023.

[17] S. Gu, J. G. Kuba *et al.*, "Multi-agent constrained policy optimisation," *arXiv preprint arXiv:2110.02793*, 2021.

[18] X. Foukas, G. Patounas *et al.*, "Network slicing in 5G: Survey and challenges," *IEEE communications magazine*, vol. 55, no. 5, pp. 94–100, 2017.

[19] 3rd Generation Partnership Project (3GPP), "Telecommunication management: Study on management and orchestration of network slicing for next generation network," 2018.

[20] A. De Domenico, Y.-F. Liu *et al.*, "Optimal virtual network function deployment for 5G network slicing in a hybrid cloud infrastructure," *IEEE Transactions on Wireless Communications*, vol. 19, no. 12, pp. 7942–7956, 2020.

[21] M. Golkarifard, C. F. Chiasserini *et al.*, "Dynamic VNF placement, resource allocation and traffic routing in 5G," *Computer Networks*, vol. 188, p. 107830, 2021.

[22] J. S. P. Roig, D. M. Gutierrez-Estevez *et al.*, "Management and orchestration of virtual network functions via deep reinforcement learning," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 304–317, 2019.

[23] H. Halabian, "Distributed resource allocation optimization in 5G virtualized networks," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 627–642, 2019.

[24] B. Han, V. Sciancalepore *et al.*, "Multiservice-based network slicing orchestration with impatient tenants," *IEEE Transactions on Wireless Communications*, vol. 19, no. 7, pp. 5010–5024, 2020.

[25] J. Zheng, P. Caballero *et al.*, "Statistical multiplexing and traffic shaping games for network slicing," *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, pp. 2528–2541, 2018.

[26] P. Caballero, A. Banchs *et al.*, "Network slicing games: Enabling customization in multi-tenant mobile networks," *IEEE/ACM Transactions on Networking*, vol. 27, no. 2, pp. 662–675, 2019.

[27] S. D'Oro, F. Restuccia *et al.*, "The slice is served: Enforcing radio access network slicing in virtualized 5G systems," in *IEEE Conference on Computer Communications (INFOCOM)*, 2019, pp. 442–450.

[28] M. Zambianco and G. Verticale, "Interference minimization in 5G physical-layer network slicing," *IEEE Transactions on Communications*, vol. 68, no. 7, pp. 4554–4564, 2020.

[29] A. T. Z. Kasgari, W. Saad *et al.*, "Experienced deep reinforcement learning with generative adversarial networks (GANs) for model-free ultra reliable low latency communication," *IEEE Transactions on Communications*, 2020.

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

[31] R. Ferrus, O. Sallent *et al.*, "On 5g radio access network slicing: Radio interface protocol features and configuration," *IEEE Communications Magazine*, vol. 56, no. 5, pp. 184–192, 2018.

[32] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[33] K. Zhang, Z. Yang *et al.*, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," *Handbook of Reinforcement Learning and Control*, pp. 321–384, 2021.

[34] A. Oroojlooy and D. Hajinezhad, "A review of cooperative multi-agent deep reinforcement learning," *Applied Intelligence*, pp. 1–46, 2022.

[35] A. Wong, T. Bäck *et al.*, "Deep multiagent reinforcement learning: Challenges and directions," *Artificial Intelligence Review*, pp. 1–34, 2022.

[36] W. Fu, C. Yu *et al.*, "Revisiting some common practices in cooperative multi-agent reinforcement learning," in *39th International Conference on Machine Learning (PMLR)*, 2022, pp. 6863–6877.

[37] J. G. Kuba, R. Chen *et al.*, "Trust region policy optimisation in multi-agent reinforcement learning," *arXiv preprint arXiv:2109.11251*, 2021.

[38] S. Gronauer and K. Diepold, "Multi-agent deep reinforcement learning: a survey," *Artificial Intelligence Review*, 2022.

[39] P. Henderson, R. Islam *et al.*, "Deep reinforcement learning that matters," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[40] 3GPP. Open ran. [Online]. Available: https://www.3gpp.org/news-events/3gpp-news/open-ran

[41] M. Polese, L. Bonati *et al.*, "Understanding o-ran: Architecture, interfaces, algorithms, security, and research challenges. corr abs/2202.01032 (2022)," 2022.

[42] O.-R. W. G. 3, "O-ran near-rt ran intelligent controller nearrt ric architecture 2.00," March 2021.

[43] ——, "O-RAN E2 Service Model (E2SM) KPM 3.0," March 2023.

[44] ——, "O-RAN E2 Service Model (E2SM), RAN Control 1.03," October 2022.

[45] O.-R. W. G. 2, "O-RAN AI/ML workflow description and requirements 1.03," October 2021.
[46] ——, "O-RAN Non-RT RIC Architecture 2.01," October 2022.
[47] P. Li, J. Thomas *et al.*, "Rlops: Development life-cycle of reinforcement learning aided open ran," *IEEE Access*, 2022.
[48] A. Lacava, M. Polese *et al.*, "Programmable and customized intelligence for traffic steering in 5g networks using open ran architectures," *IEEE Transactions on Mobile Computing*, 2023.
[49] J. J. Alcaraz, "Network slicing environment," https://github.com/jjalcaraz-upct/network-slicing/, 2022.
[50] 3rd Generation Partnership Project (3GPP), "Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Frequency (RF) system scenarios," 2018.
[51] ——, "E-UTRA Base Station (BS) radio transmission and receptio," 2022.
[52] T. Hu, Q. Liao *et al.*, "Inter-cell slicing resource partitioning via coordinated multi-agent deep reinforcement learning," *arXiv preprint arXiv:2202.12833*, 2022.
[53] A. R. e. a. A. Hill, "Stable baselines," https://github.com/hilla/stable-baselines, 2018.

Mohammad Zangooei is a Ph.D. student at the David R. Cheriton School of Computer Science at the University of Waterloo. He received his Bachelor's degree in Electrical Engineering from the Sharif University of Technology, Tehran, Iran. His research interests revolve around next-generation mobile networks, artificial intelligence, and programmable data planes.

Morteza Golkarifard received his B.Sc., M.Sc., and Ph.D. degrees in computer engineering from the Sharif University of Technology. He is currently a post-doctoral fellow at the David R. Cheriton School of Computer Science at the University of Waterloo. His research interests include 5G networks, NFV, and SDN.

Mohamed Rouili is a Ph.D. student at the David R. Cheriton School of Computer Science at the University of Waterloo. Previously he completed his Bachelor's and Master's degrees in Computer Science at the University of Tebessa, Algeria. His research interests revolve around network systems and protocols, 5G cellular networks, and machine learning.

Niloy Saha is a Ph.D. student at the University of Waterloo. He received his Master's degree in Computer Science from the Indian Institute of Technology, Kharagpur, India. His research interests are focused on building next-generation mobile networks and intelligent algorithms for their orchestration and management.

Raouf Boutaba received the M.Sc. and Ph.D. degrees in computer science from Sorbonne University in 1990 and 1994, respectively. He is currently a University Chair Professor and the Director of the David R. Cheriton School of Computer Science at the University of Waterloo (Canada). He is the founding Editor-in-Chief of the IEEE Transactions on Network and Service Management (2007-2010) and served as the Editor-in-Chief of the IEEE Journal on Selected Areas in Communications (2018-2021). He is a fellow of the IEEE, the Engineering Institute of Canada, the Canadian Academy of Engineering, and the Royal Society of Canada.