A Comparative Analysis of Deep Reinforcement Learning-based xApps in O-RAN

Maria Tsampazi*, Salvatore D'Oro*, Michele Polese*, Leonardo Bonati*,
Gwenael Poitau§, Michael Healy§, Tommaso Melodia*
*Institute for the Wireless Internet of Things, Northeastern University, Boston, MA, U.S.A.
E-mail: {tsampazi.m, s.doro, m.polese, l.bonati, melodia}@northeastern.edu
§Dell Technologies, P&O OCTO – Advanced Wireless Technology
E-mail: {gwenael.poitau, mike.healy}@dell.com

GLOBECOM 2023 - 2023 IEEE Global Communications Conference | 979-8-3503-1090-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/GLOBECOM54140.2023.10437367

***Abstract*—The highly heterogeneous ecosystem of Next Generation (NextG) wireless communication systems calls for novel networking paradigms where functionalities and operations can be dynamically and optimally reconfigured in real time to adapt to changing traffic conditions and satisfy stringent and diverse Quality of Service (QoS) demands. Open Radio Access Network (RAN) technologies, and specifically those being standardized by the O-RAN Alliance, make it possible to integrate network intelligence into the once monolithic RAN via intelligent applications, namely, xApps and rApps. These applications enable flexible control of the network resources and functionalities, network management, and orchestration through data-driven control loops. Despite recent work demonstrating the effectiveness of Deep Reinforcement Learning (DRL) in controlling O-RAN systems, how to design these solutions in a way that does not create conflicts and unfair resource allocation policies is still an open challenge. In this paper, we perform a comparative analysis where we dissect the impact of different DRL-based xApp designs on network performance. Specifically, we benchmark 12 different xApps that embed DRL agents trained using different reward functions, with different action spaces and with the ability to hierarchically control different network parameters. We prototype and evaluate these xApps on Colosseum, the world's largest O-RAN-compliant wireless network emulator with hardware-in-the-loop. We share the lessons learned and discuss our experimental results, which demonstrate how certain design choices deliver the highest performance while others might result in a competitive behavior between different classes of traffic with similar objectives.**

***Index Terms*—Open RAN, O-RAN, Resource Allocation, Network Intelligence, Deep Reinforcement Learning.**

I. Introduction

Programmable, virtualized, and disaggregated architectures are seen as key enablers of Next Generation (NextG) cellular networks. Indeed, the flexibility offered through softwarization, virtualization, and open standardized interfaces provides new self-optimization capabilities. These concepts are at the foundation of the Open Radio Access Network (RAN) paradigm, which is being specified by the O-RAN Alliance. Thanks to the RAN Intelligent Controllers (RICs) proposed by O-RAN (i.e., the near- and non-real-time RICs), intelligence can be embedded into the network and leveraged for on-demand closed-loop control of its resources and functionalities [1]. In O-RAN, this is achieved via intelligent applications, called xApps and rApps, which execute on the near- or non-real-time RICs respectively. Through the RICs, these applications interface with the network nodes and implement data-driven closed-loop control based on real-time statistics received from the RAN, thus realizing the vision of resilient, reconfigurable and autonomous networks [1]. Since they do not require prior knowledge of the underlying network dynamics [2], Deep Reinforcement Learning (DRL) techniques are usually preferred in the design of such control solutions for the Open RAN [3]–[5].

Intelligent control in O-RAN through xApps has widely attracted the interest of the research community. For example, [6] proposes the NexRAN xApp to control and balance the throughput of different RAN slices. The authors of [7] develop a Reinforcement Learning (RL) xApp to assign resource blocks to certain users according to their Channel State Information (CSI) and with the goal of maximizing the aggregated data rate of the network. A deep Q-learning-based xApp for controlling slicing policies to minimize latency for Ultra Reliable and Low Latency Communications (URLLC) slices is presented in [8]. The authors of [4] experimentally evaluate and demonstrate several DRL-based xApps under a variety of traffic and channel conditions, and investigate how different action space configurations impact the network performance. Finally, other research efforts focus on coordinating multiple xApps to control different parameters via a combination of federated learning and team learning [9], [10].

A. Contributions and Outline

The above works clearly show that DRL and Artificial Intelligence (AI), are catalysts in the design and development of intelligent control solutions for the Open RAN. However, despite early results showing their success and effectiveness, designing DRL agents for complex Open RAN scenarios—characterized by the coexistence of diverse traffic profiles and potentially conflicting Quality of Service (QoS) demands—is still an open challenge that, as we describe below, we aim at addressing in this paper. Specifically, our goal is to go beyond merely using AI, and specifically DRL, in a black-box manner. Instead, we try to address some fundamental questions that are key for the success of intelligence in Open RAN systems.

We consider an Open RAN delivering services to URLLC, Massive Machine-Type Communications (mMTC) and Enhanced Mobile Broadband (eMBB) network slices. Specifi-

This article is based upon work partially supported by Dell Technologies and by the U.S. National Science Foundation under grants CNS-1925601, CNS-2112471, CNS-1923789 and CNS-2120447.

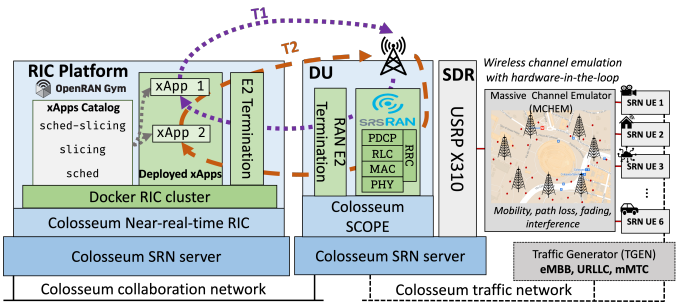


Fig. 1. Reference O-RAN testing architecture with focus on the case of two xApps operating at different time scales, T_i, as described in Section V-B.

cally, we use OpenRAN Gym [11]—an open-source framework for Machine Learning (ML) experimentation in O-RAN—to deploy such Open RAN on the Colosseum wireless network emulator [12], and control it through 12 DRL xApps. These xApps have been trained to perform slice-based resource allocation (i.e., scheduling profile selection and RAN slicing control) and to meet the diverse requirements of each slice. We investigate the trade-off between long-term and short-term rewards, we discuss and compare different design choices of action set space, hierarchical decision-making policies and action-taking timescales. Finally, we show how these choices greatly impact network performance and affect each slice differently.

To the best of our knowledge, this is the first experimental study that provides a comprehensive evaluation of the design choices for DRL-based xApps. We hope that our findings and insights might help in designing xApps for NextG Open RANs.

The remainder of this paper is organized as follows. Section II describes our system model and data-driven optimization framework. Section III presents the different DRL optimization strategies considered in this work, while Section IV details our experimental setup and training methodology. Experimental results are discussed in Section V. Finally, Section VI draws our conclusions and presents some future work directions.

II. System Model and Data-Driven Optimization Framework

In this work, we consider an Open RAN multi-slice scenario where User Equipments (UEs) generate traffic with diverse profiles and QoS demands. Without loss of generality, we assume that traffic generated by UEs can be classified into eMBB, URLLC, or mMTC slices.

To satisfy the diverse QoS demands required by each slice and intelligently control the resource allocation process, we leverage RL. Specifically, as shown in Fig. 1, we leverage xApps embedding DRL agents that are tasked with reconfiguring control parameters of the Base Station (BS) such as RAN slicing (i.e., the portion of available Physical Resource Blocks (PRBs) that are allocated to each slice at any given time) and Medium Access Control (MAC) layer scheduling policies (in our case, by selecting a dedicated scheduler profile to each slice among Round Robin (RR), Waterfilling (WF) and Proportionally Fair (PF)). xApps make decisions based on the UEs traffic demand, load, performance and network conditions that are given by Key Performance Measurements (KPMs) periodically reported by RAN.

A. DRL Agent Architecture

We focus on DRL agents that implement the Proximal Policy Optimization (PPO) algorithm, a state-of-the-art on-policy DRL algorithm based on an actor-critic network architectural approach. Specifically, the actor and critic network "work" cooperatively to learn a policy that selects actions that deliver the highest reward possible for each state. While the actor's task is to take actions based on current network states, the critic's target is to evaluate actions taken by the actor network and provide feedback that reflects how effective the action taken by the actor is. In this way, the critic helps the actor in taking actions that lead to the highest rewards for each given state.

The reason we focus on PPO is that it has been demonstrated several times to outperform other architectures [4], [7]. Actor and critic networks are fully-connected neural networks with 3 layers of 30 neurons each. The hyperbolic tangent serves as the activation function while the learning rate is set to 10^{-3}. We follow the same approach as in [4], where the input (e.g., KPMs) of the DRL agent is first processed by the encoding part of an autoencoder for dimensionality reduction. This also synthetically reduces the state space and makes training more efficient in terms of time and generalization. In detail, the autoencoder converts an input matrix of $K = 10$ individual measurements of $M = 3$ KPM metrics (i.e., downlink throughput, buffer occupancy, and number of transmitted packets) into a single M-dimensional vector. The Rectified Linear Unit (ReLU) activation function and four fully-connected layers of 256, 128, 32 and 3 neurons are also used in the encoder.

The cumulative average reward function of the DRL agent is designed to jointly satisfy the QoS demand of the three slices with respect to their KPM requirements. For instance, eMBB users aim to maximize throughput, while mMTC users aim at maximizing the number of transmitted packets. Finally, the goal of URLLC users is to deliver packets with minimum latency. Since the base station cannot measure end-to-end application-layer latency (which is instead measured at the receiver side), we measure latency in terms of number of bytes in the transmission buffer, the smaller the buffer, the smaller the latency. The reward is formulated as the weighted sum in Eq. (1)

$$R = \sum_{t=0}^{\infty} \gamma^t \left(\sum_{j=1}^{N} w_j \cdot r_{j,t} \right), \tag{1}$$

where t represents the training step, and $N = 3$ is the total number of slices, w_j represents the weight associated to slice j, considered for reward maximization in the three corresponding slices. Finally, γ is the discount factor and $r_{j,t}$ describes the slice-specific reward obtained at each training step t. In our case, $r_{j,t}$ represents the average value of the KPM measured by all users of slice j at time t (e.g., throughput for the eMBB slice). Note that the weight w_j for the URLLC slice is negative to model the minimization of the buffer occupancy. The models that we have designed and trained are deployed as xApps on the near-real-time RIC, as illustrated in Fig. 1.

III. DRL Optimization Strategies

We investigate how different design choices affect the effectiveness and decision-making of the DRL-based xApps. We consider the following design choices, for a total of 12 xApps.

Short-term vs. Long-term Rewards. We train DRL agents with different values of the discount factor γ. The PPO discount factor weights instantaneous rewards against long-term rewards. A higher value prioritizes long-term rewards, while a lower γ prioritizes short-term rewards. Results of this exploration are provided in Section V-A.

Hierarchical Decision-Making. We investigate the case of two xApps configuring different parameters in parallel but at different timescales. In this way, we investigate how multiple xApps with different optimization goals and operating timescales impact the overall network performance. The findings of this investigation are provided in Section V-B, and a practical example is illustrated in Fig. 1.

Impact of Reward's Weights. Finally, we test different values for the weights w_i of the slices in Eq. (1). A different weight configuration affects how DRL agents prioritize each slice. The results of this analysis are reported in Section V-C, where we show how weights significantly impact the overall performance and can result in inefficient control strategies.

IV. Experimental Setup and DRL Training

To experimentally evaluate the DRL agents, we leverage the capabilities of OpenRAN Gym [11], an open-source experimental toolbox for end-to-end design, implementation, and testing of AI/ML applications in O-RAN. It features:

- End-to-end RAN and core network deployments though the srsRAN [13] softwarized open-source protocol stack;
- Large-scale data collection, testing and fine-tuning of RAN functionalities through the SCOPE framework [14], which adds open Application Programming Interfaces (APIs) to srsRAN for the control of slicing and scheduling functionalities, as well as for KPMs collection;
- An O-RAN-compliant control architecture to execute AI/ML-based xApps via the ColO-RAN near-real-time RIC [4]. The E2 interface between RAN and the RIC and its Service Models (SMs) [1] manage streaming of KPMs from the RAN and control actions from the xApps.

We deploy OpenRAN Gym on Colosseum [12], a publicly available testbed with 128 Standard Radio Nodes (SRNs), i.e., pairs of Dell PowerEdge R730 servers and NI USRP X310 Software-defined Radios (SDRs). Colosseum enables large-scale experimentation in diverse Radio Frequency (RF) environments and network deployments. This is done through the Massive Channel Emulator (MCHEM) component, which leverages Field Programmable Gate Array (FPGA)-based Finite Impulse Response (FIR) filters to reproduce different conditions of the wireless environment modeled a priori through ray-tracing software, analytical models, or real-world measurements. The channel conditions that can be emulated in this way include path loss, fading, attenuation, mobility and interference of signals. Similarly, the Colosseum Traffic Generator (TGEN), built on top of the Multi-Generator (MGEN) TCP/UDP traffic generator [15], emulates different traffic profiles (e.g., multimedia content), demand, and distributions (e.g., Poisson, periodic).

We deploy a 3GPP-compliant cellular network with one base station and 6 UEs uniformly distributed across 3 different slices. These are: (i) eMBB that concerns high traffic modeling of high-quality multimedia content and streaming applications; (ii) URLLC for time-critical applications, such as autonomous driving in Vehicle-to-everything (V2X) scenarios; and (iii) mMTC for Internet of Things (IoT) devices with low data rate requirements but with high need for consistent information exchange. In terms of physical deployment, UEs are uniformly distributed within a 20 m radius from the BS, in the urban environment of Rome, Italy [14].

The bandwidth of the BS is set to 10 MHz (i.e., 50 PRBs) and is divided among the 3 slices, with 2 users statically assigned to each slice. Slice-based traffic is created with the following specifications: eMBB users request 4 Mbps constant bitrate, while URLLC and mMTC UEs generate 89.3 kbps and 44.6 kbps Poisson traffic, respectively.

To train the DRL agents, we used the publicly available dataset described in [4]. This dataset contains about 8 GB of KPMs collected by using OpenRAN Gym and the Colosseum network emulator over 89 hours of experiments, and concerns setups with up to 7 base stations and 42 UEs belonging to different QoS classes, and served with heterogeneous scheduling policies. Each DRL model evaluated in the following sections takes as input RAN KPMs such as throughput, buffer occupancy, number of PRBs, and outputs resource allocation policies (e.g., RAN slicing and/or scheduling) for RAN control.

Abiding by the O-RAN specifications, we train our ML model offline on Colosseum's GPU-accelerated environment, which includes two NVIDIA DGX A100 servers with 8 GPUs each. Trained DRL-agents are onboarded on xApps inside softwarized containers implemented via Docker and deployed on the ColO-RAN near-real-time RIC.

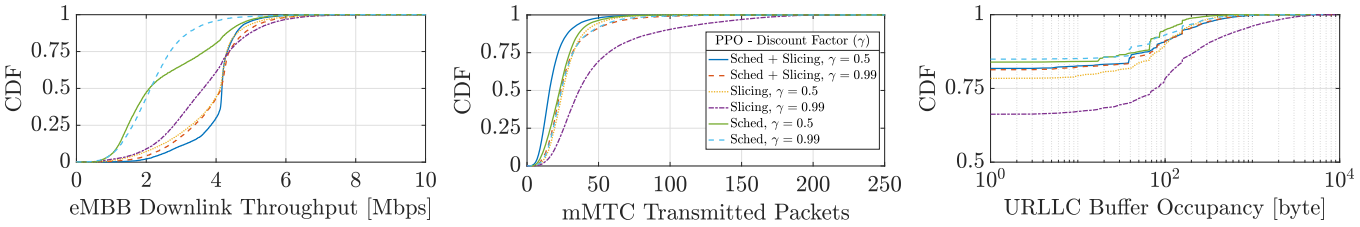
V. Experimental Evaluation

In this section, we present the results of an extensive performance evaluation campaign, with more than 16 hours of experiments, to profile the impact of the strategies discussed in Section III. These results were produced by taking the median as the most representative statistical value of a dataset, and averaged over multiple repetitions of experiments in the Downlink (DL) direction of the communication system.

A. Impact of Discount Factor on the Action Space

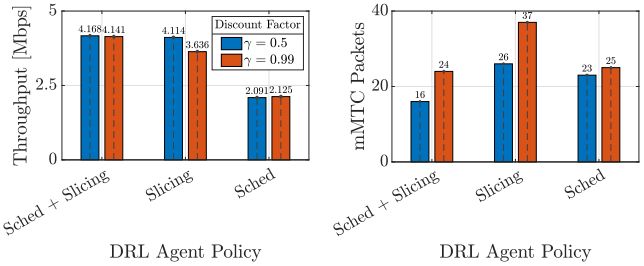
We explore how RAN slicing, MAC scheduling, and joint slicing and scheduling control are affected by training procedures that favor short-term against long-term rewards. Due to space limitations, we only report results obtained with $\gamma \in \{0.5, 0.99\}$. The reward's weight configuration used in this study is shown in Table I and identified as `Default`.

In Fig. 2, we report the Cumulative Distribution Function (CDF) of individual KPMs for each slice and for different xApps trained to control different sets of actions and using



(a) eMBB Throughput (b) mMTC Packets (c) URLLC Buffer Occupancy

Fig. 2. Performance evaluation under different action spaces and values of the γ parameter.



(a) eMBB DL Throughput (b) mMTC Packets

Fig. 3. Median values under different action spaces and values of γ.

TABLE I
WEIGHT CONFIGURATIONS

Weights	eMBB	mMTC	URLLC
`Default`	72.0440333	0.229357798	0.00005
`Alternative`	72.0440333	1.5	0.00005

different values of γ. The median of such measurements for the eMBB and mMTC slices is instead reported in Fig. 3. The median for the URLLC slice is not reported, as this value is zero in all configurations. The best performing configurations for the eMBB and mMTC slices are instead listed in numerical order in Table II from best to worst performing.

TABLE II
PER-SLICE TOP PERFORMING XAPPS UNDER THE DEFAULT WEIGHT CONFIGURATION

	eMBB	mMTC
1)	`Sched & Slicing 0.5`	`Slicing 0.99`
2)	`Sched & Slicing 0.99`	`Slicing 0.5`
3)	`Slicing 0.5`	`Sched 0.99`
4)	`Slicing 0.99`	`Sched & Slicing 0.99`
5)	`Sched 0.99`	`Sched 0.5`
6)	`Sched 0.5`	`Sched & Slicing 0.5`

Our results show that `Sched & Slicing` and `Slicing 0.5` favor eMBB the most, with `Sched & Slicing 0.5` being the best configuration among the ones considered. Moreover, slicing is essential to ensure high throughput values (the four top-performing xApps for eMBB include slicing as a control action). We also notice that prioritizing immediate rewards (i.e., $\gamma = 0.5$) results in higher throughput values if compared to xApps embedding agents trained to maximize long-term rewards. This design option, when combined with a bigger action space (e.g., scheduling & slicing) ultimately yields a higher throughput.

For the mMTC slice, the `Slicing 0.99` xApp always yields the best performance. However, we notice that `Sched & Slicing 0.5`, which is the best-performing xApp for eMBB, yields the worst performance for mMTC. Although a larger action space and a short-term reward design is ideal for eMBB (e.g., `Sched & Slicing 0.5`), we notice that this performance gain comes at the expense of the mMTC slice. Indeed, in Figs. 3(a) and 3(b), we observe that the higher the eMBB performance, the lower the mMTC's. This is clearly illustrated when we compare the "best" per-slice policies, respectively `Sched & Slicing 0.5` (eMBB) and `Slicing 0.99` (mMTC). The former delivers the highest reported eMBB throughput (4.168 Mbps) but the lowest number of mMTC packets (16 packets), while the latter delivers the highest number of mMTC packets (37 packets) and one of the lowest measured eMBB throughput values (i.e., 3.636 Mbps).

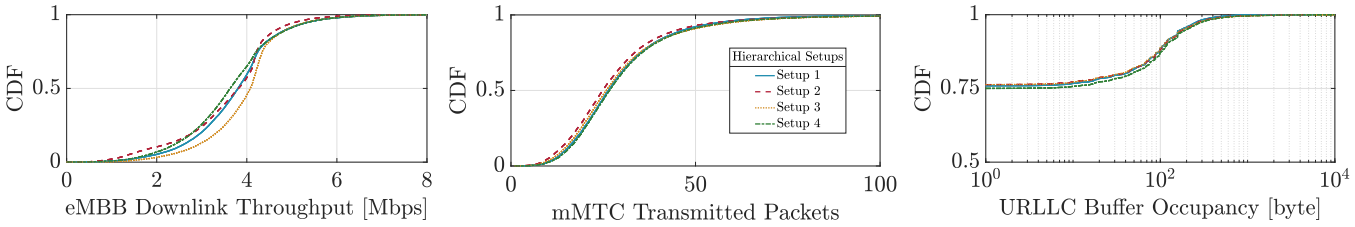
Hence, eMBB-mMTC slices indicate a competitive behavior, since we cannot optimally satisfy both of them without loss in their respective rewards, as they compete for the amount of packets required for transmission. Our results show that, in general, controlling scheduling only is not ideal as it strongly penalizes eMBB performance with a modest improvement in terms of number of transmitted mMTC packets.

B. Impact of Hierarchical Decision-Making

In this analysis, we evaluate the effectiveness of making disjoint decisions to control scheduling and slicing policies. We select the best performing single-action xApps from Table II, i.e., `Slicing 0.5` and `Sched 0.99`, and we compare their execution at different timescales. The former, provides a good balance in terms of eMBB throughput (~ 4 Mbps) and number of mMTC packets, while the latter, provides the best performance for the mMTC slice. With this design choice, we expect to maintain high performance for both eMBB and mMTC.

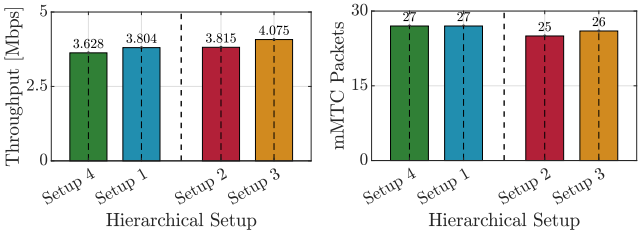
We consider four setups, summarized in Table III. Each entry describes how frequently the BS reports KPMs to the RIC. For instance, in Setup 1, the xApp for slicing control receives data from the BS every 1 s, while the scheduling agent receives the respective metrics every 10 s. Despite taking into account RAN telemetry reported every $1, 5$ or 10 s, the DRL decision-making process and the enforcement of a control policy on the BS occur within a granularity of sub-milliseconds, and hence the intelligent control loops are still in compliance with the timescale requirements of the near-real-time RIC.

Results of this analysis are presented in Figs. 4 and 5. From Fig. 4(a), *Setup 3* delivers the best eMBB performance, *Setups* 1 and 2 perform almost equally, while *Setup 4* performs the



(a) eMBB Throughput (b) mMTC Packets (c) URLLC Buffer Occupancy

Fig. 4. Performance evaluation under different hierarchical configurations.



(a) eMBB DL Throughput (b) mMTC Packets

Fig. 5. Median values under different hierarchical configurations.

TABLE III
HIERARCHICAL REPORTING SETUP

Setup ID	`Slicing 0.5`	`Sched 0.99`
1	1 s	10 s
2	1 s	5 s
3	10 s	1 s
4	5 s	1 s

worst. For mMTC, in Fig. 4(b) we notice that all combinations perform similarly and deliver approximately 26 packets, with *Setup 1* and *Setup 4* delivering an additional packet. From Fig. 4(c), we notice that all setups deliver the same performance for the URLLC slice and, despite not being reported in the figures, they all yield a median buffer occupancy of 0 byte, i.e., they maintain an empty buffer to ensure low latency values. In Fig. 5(a), we notice that *Setups* 2 and 3 deliver the highest eMBB throughput. In Fig. 5(b), instead, we notice that *Setups* 1 and 4 deliver the highest number of transmitted mMTC packets.

Our findings on hierarchical control verify eMBB's and mMTC's competitive behavior for individual reward maximization. Our results show that the rewards of eMBB and mMTC slices are competing with one another, as the best configuration for eMBB corresponds to the worst configuration for mMTC, and vice versa. Among all considered configurations, *Setup 3* offers the best trade-off, as it delivers the highest throughput at the expense of a single mMTC packet less being transmitted.

C. Impact of Weight Configuration

In this study, we consider different weight configurations to compute the cumulative average reward function in Eq. (1). The considered configurations are reported in Table I. The *Alternative* weight configuration is computed by using the weights in Table IV, where $A, B, C, \alpha_{eMBB}, \beta_{mMTC}$, and γ_{URLLC} are used to both scale and prioritize certain slices. Specifically, A, B, C are used to scale the individual weights according to statistical information of corresponding KPMs. For example,

TABLE IV
WEIGHT DESIGN

w_{eMBB}	w_{mMTC}	w_{URLLC}
$\alpha_{eMBB} \cdot \frac{1}{A}$	$\beta_{mMTC} \cdot \frac{1}{B}$	$\gamma_{URLLC} \cdot \left(-\frac{1}{C}\right)$

A, B, C can represent either the average, minimum or maximum values reported KPM per slice so as to scale the weight according to the dynamic range of the corresponding KPM. Similarly, $\alpha_{eMBB}, \beta_{mMTC}$, and γ_{URLLC} can be used to give priority to one slice or the other.

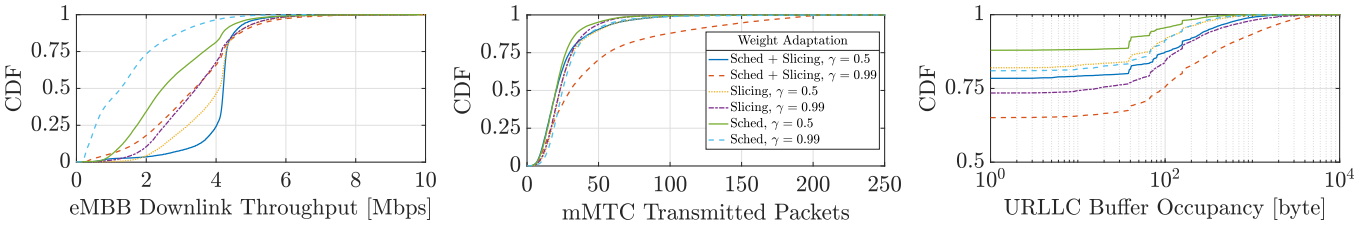
We set $\alpha_{eMBB} = 1000$, $\beta_{mMTC} = 456$ and $\gamma_{URLLC} = 1$. As a reference for A, B and C, we choose the historically maximum reported KPM values for each slice, i.e., $A = 13.88$ Mbps, $B = 304$, and $C = 20186$ byte.

Based on these steps, we derive their respective weights w_{eMBB}, w_{mMTC}, w_{URLLC}. For example, the weight of mMTC can be computed as $w_{mMTC} = \beta_{mMTC} \cdot \frac{1}{B} = 456/304 = 1.5$, as reported in the *Alternative* configuration in Table I. The goal of comparing the two *Default* and *Alternative* weight configurations is to explore and understand the dynamics between mMTC and eMBB and the overall impact on the network performance. Specifically, since previous results have shown that the mMTC can be penalized by the eMBB slice, with the *Alternative configuration* we aim at giving the former a weight that is $6\times$ larger than the *Default* configuration.

Results for the *Alternative* configuration are reported in Figs. 6 and 7. In Fig. 6(a), `Sched & Slicing 0.5` delivers the best eMBB performance. Similarly to the results presented in Section V-A, `Scheduling & Slicing 0.5` and `Slicing 0.5` are the best choices, with short-term reward design being ideal for eMBB. In Fig. 6(b), the *Alternative* weight configuration results in `Scheduling & Slicing 0.99` being the best mMTC choice and long-term rewards are better for mMTC users. For URLLC, all policies perform well, with `Scheduling & Slicing 0.5` performing slightly better compared to `Scheduling & Slicing 0.99`.

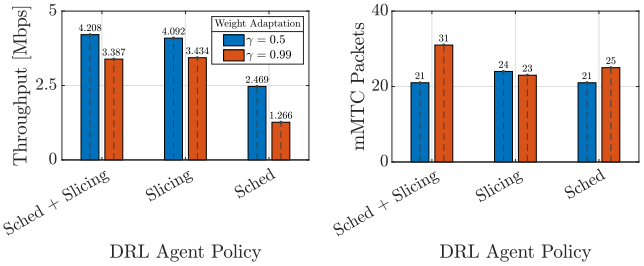
Figs. 7(a) and 7(b), confirm that controlling scheduling alone does not improve performance in general. Similarly to our previous analysis, a high eMBB performance (i.e., `Sched & Slicing 0.5`) results in a degraded mMTC performance. However, if compared with the `Default`, the `Alternative` weight configuration achieves a 31.25% increase for mMTC, with the same equally good URLLC performance and a 1% throughput increase for eMBB users.

In Table V we summarize the design options that deliver



(a) eMBB Throughput (b) mMTC Packets (c) URLLC Buffer Occupancy

Fig. 6. Performance evaluation under the *Alternative* weight configuration for different actions spaces and discount factors.



(a) eMBB DL Throughput (b) mMTC Packets

Fig. 7. Median values under the *Alternative* weight configuration for different actions spaces and discount factors.

good overall performance. Table VI indicates eMBB and mMTC's dynamic and competitive relation. Option 2 brings balance, in terms of throughput and transmitted packets, Option 1 favors eMBB, and Option 4 boosts mMTC but with a significant decrease in the QoS of the eMBB slice.

TABLE V
DESIGN OPTIONS CATALOG

Option 1	`Sched & Slicing 0.5 - Alternative`
Option 2	`Slicing 0.5 - Default`
Option 3	`Hierarchical Control - Setup 1`
Option 4	`Slicing 0.99 - Default`

TABLE VI
DESIGN OPTIONS

	eMBB [Mbps]	**mMTC** [packet]	**URLLC** [byte]
Option 1	4.208	21	0
Option 2	4.114	26	0
Option 3	3.804	27	0
Option 4	3.636	37	0

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we investigated the impact of DRL design choices on the performance of an Open RAN system controlled by xApps embedding DRL agents that make decisions in near-real-time to compute efficient slicing and scheduling control policies. We benchmarked 12 xApps trained using DRL agents with different actions spaces, reward design and decision-making timescales. Our experimental results show that network slices with similar objectives (e.g., maximizing throughput and number of transmitted packets) might result in a competitive behavior that can be mitigated using proper weight and reward configurations. Our results point the need for either a) slice-specific xApp design, or b) joint optimization on the premise of xApp conflict avoidance. Part of our current and future work focuses on these directions, with additional testing under diverse Channel Quality Information (CQI) conditions, mobility patterns and dynamically changing traffic load. Optimal weight design with respect to the size of the action space is also part of our ongoing work.

REFERENCES

[1] 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.

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

[3] X. Wang, J. D. Thomas, R. J. Piechocki, S. Kapoor, R. Santos-Rodríguez, and A. Parekh, "Self-play learning strategies for resource assignment in Open-RAN networks," *Computer Networks*, vol. 206, p. 108682, 2022.

[4] 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*, 2022.

[5] S. D'Oro, L. Bonati, M. Polese, and T. Melodia, "OrchestRAN: Network automation through orchestrated intelligence in the open RAN," in *IEEE Conference on Computer Communications*, 2022, pp. 270–279.

[6] D. Johnson, D. Maas, and J. Van Der Merwe, "NexRAN: Closed-loop RAN slicing in POWDER-A top-to-bottom open-source open-RAN use case," in *Proceedings of the 15th ACM Workshop on Wireless Network Testbeds, Experimental evaluation & CHaracterization*, 2022, pp. 17–23.

[7] M. Kouchaki and V. Marojevic, "Actor-Critic Network for O-RAN Resource Allocation: xApp Design, Deployment, and Analysis," in *IEEE Globecom Workshops (GC Wkshps)*, 2022, pp. 968–973.

[8] A. Filali, B. Nour, S. Cherkaoui, and A. Kobbane, "Communication and computation O-RAN resource slicing for URLLC services using deep reinforcement learning," *IEEE Communications Standards Magazine*, vol. 7, no. 1, pp. 66–73, 2023.

[9] P. Iturria Rivera, H. Zhang, H. Zhou, S. Mollahasani, and M. Erol Kantarci, "Multi-agent team learning in virtualized open radio access networks (o-ran)," *Sensors*, vol. 22, p. 5375, 07 2022.

[10] H. Zhang, H. Zhou, and M. Erol-Kantarci, "Federated deep reinforcement learning for resource allocation in O-RAN slicing," in *IEEE Global Communications Conference (GLOBECOM)*, 2022, pp. 958–963.

[11] L. Bonati, M. Polese, S. D'Oro, S. Basagni, and T. Melodia, "OpenRAN Gym: AI/ML development, data collection, and testing for O-RAN on PAWR platforms," *Computer Networks*, vol. 220, p. 109502, 2023.

[12] L. Bonati, P. Johari, M. Polese, S. D'Oro, S. Mohanti, M. Tehrani-Moayyed, D. Villa, S. Shrivastava, C. Tassie, K. Yoder *et al.*, "Colosseum: Large-scale Wireless Experimentation through Hardware-in-the-Loop Network Emulation," in *IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, 2021, pp. 105–113.

[13] I. Gomez-Miguelez, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano, and D. J. Leith, "srsLTE: An Open-Source Platform for LTE Evolution and Experimentation," in *Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization*, 2016, pp. 25–32.

[14] L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "SCOPE: An Open and Softwarized Prototyping Platform for NextG Systems," in *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, 2021, pp. 415–426.

[15] U.S. Naval Research Laboratory, "Multi-Generator (MGEN) Network Test Tool". https://www.nrl.navy.mil/itd/ncs/products/mgen. 2019.