PandORA: Automated Design and Comprehensive Evaluation of Deep Reinforcement Learning Agents for Open RAN

Maria Tsampazi, *Graduate Student Member, IEEE*, Salvatore D'Oro, Michele Polese, *Member, IEEE*, Leonardo Bonati, *Member, IEEE*, Gwenael Poitau, Michael Healy, *Member, IEEE*, Mohammad Alavirad, and Tommaso Melodia, *Fellow, IEEE*

***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 intelligent control loops. Recent work has showed how Deep Reinforcement Learning (DRL) is effective in dynamically controlling O-RAN systems. However, how to design these solutions in a way that manages heterogeneous optimization goals and prevents unfair resource allocation is still an open challenge, with the logic within DRL agents often considered as a opaque system. In this paper, we introduce PandORA, a framework to automatically design and train DRL agents for Open RAN applications, package them as xApps and evaluate them in the Colosseum wireless network emulator. We benchmark 23 xApps that embed DRL agents trained using different architectures, reward design, action spaces, and decision-making timescales, and with the ability to hierarchically control different network parameters. We test these agents on the Colosseum testbed under diverse traffic and channel conditions, in static and mobile setups. Our experimental results indicate how suitable fine-tuning of the RAN control timers, as well as proper selection of reward designs and DRL architectures can boost network performance according to the network conditions and demand. Notably, finer decision-making granularities can improve Massive Machine-Type Communications (mMTC)'s performance by $\sim 56\%$ and even increase Enhanced Mobile Broadband (eMBB) Throughput by $\sim 99\%$.**

Received 26 December 2023; revised 7 November 2024; accepted 21 November 2024. Date of publication 25 November 2024; date of current version 6 March 2025. This work was supported in part by Dell Technologies and in part by the U.S. National Science Foundation under Grant CNS-1925601, Grant CNS-2112471, Grant CNS-1923789, and Grant CNS-2120447. Recommended for acceptance by L. Bai. *(Corresponding author: Maria Tsampazi.)*

Maria Tsampazi, Salvatore D'Oro, Michele Polese, Leonardo Bonati, and Tommaso Melodia are with the Institute for the Wireless Internet of Things, Northeastern University, Boston, MA 02115 USA (e-mail: tsampazi.m@northeastern.edu; s.doro@northeastern.edu; m.polese@northeastern.edu; l.bonati@northeastern.edu; t.melodia@northeastern.edu).

Gwenael Poitau, Michael Healy, and Mohammad Alavirad are with the P&O OCTO-Advanced Wireless Technology, Dell Technologies, Round Rock, TX 78682 USA (e-mail: gwenael.poitau@dell.com; mike.healy@dell.com; mohammad.alavirad@dell.com).

Digital Object Identifier 10.1109/TMC.2024.3505781

***Index Terms*—Deep reinforcement learning, network intelligence, O-RAN, open RAN, resource allocation.**

I. Introduction

PROGRAMMABLE, virtualized, and disaggregated architectures are seen as key enablers of NextG cellular networks. Indeed, the flexibility offered through softwarization, virtualization, and open standardized interfaces provides new self-optimization capabilities through Artificial Intelligence (AI). These concepts are at the foundation of the Open 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]. 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. Since they do not require prior knowledge of the underlying network dynamics [2], DRL techniques are usually preferred in the design of such control solutions for the Open RAN [3], [4], [5].

A. Related Work

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. In [7], an O-RAN-compliant RAN platform is introduced for Multi-Radio Access Technology (multi-RAT) environments, enabling network slicing and slice-specific scheduling. The FlexSlice framework in [8] addresses RAN slicing and scheduling with finer control loop granularity for real-time efficiency. In [9], the authors discuss an O-RAN-based framework for predictive Uplink (UL) network slicing, leveraging a Deep Learning-based xApp for dynamic reconfiguration of RAN scheduling for Ultra Reliable and Low Latency Communications (URLLC) services. Similarly, in [10], the authors present a Deep Learning-based approach to develop a RAN slicing xApp. The authors of [11] 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 URLLC slices is presented in [12], while a cooperative multi-agent RL algorithm for Open RAN slicing and radio resource management, taking into account various Service Level Agreement (SLA) constraints, is discussed in [13]. The authors of [4] experimentally evaluate and demonstrate three 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 [14], [15], [16], [17].

DRL has been widely used for resource allocation [18], [19] in the wireless communications and networking field. In [20], the authors propose a QoS-oriented resource allocation scheme for network slicing and throughput maximization in 5th generation (5G) and beyond networks. DeepSlicing in [21] relies on a DRL approach to determine the number of resources required by the User Equipments (UEs) in each slice to ensure their QoS demands are met. In [22], authors present a DRL-based framework for NextG RAN slicing to improve resource utilization and meet the slices' QoS requirements by leveraging feedback generated by the Deep Q-Network (DQN) agents. In [23], the authors discuss a two-level scheduling framework to jointly maximize the Quality of Experience (QoE) and Spectrum Efficiency (SE). Specifically, the higher layer controller manages bandwidth allocation, while the lower level controller is responsible for scheduling the Physical Resource Block (PRB) and power allocation at a smaller timescale, utilizing Multiple Input, Multiple Output (MIMO) antenna technology. In [24] a Multi-Agent DRL (MARL) approach is introduced for joint optimization of UE scheduling and power control in a wireless environment characterized by the coexistence of multiple Transmitters (TXs) with multiple associated UEs. In [25], an energy-efficient RAN slicing scheme utilizing DRL-assisted learning for resource allocation in 5G networks is introduced. Specifically, a combination of Deep Learning and DRL is employed for resource allocation on large and small time-scales, respectively. In [26] a hierarchical DRL-based approach is employed for resource allocation, coupled with a load balancing strategy to enhance energy efficiency while ensuring user fairness from a QoS perspective.

B. Contributions and Outline

The above works clearly show that DRL and 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 which are effective at controlling and optimizing complex Open RAN scenarios—characterized by the coexistence of diverse traffic profiles and potentially conflicting 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 design questions that are key for the success of intelligence in Open RAN systems.

We consider an Open RAN delivering services to URLLC, mMTC and eMBB network slices. Specifically, we use Open-RAN Gym [27]—an open-source framework for Machine Learning (ML) experimentation in O-RAN—to deploy such Open RAN on the Colosseum wireless network emulator [28], and control it through xApps using 23 different DRL agent designs. 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 and DRL architecture, 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, in [29], we conducted the first experimental study that comprehensively evaluated the design choices for DRL-based xApps to provide insights on the design of xApps for NextG Open RANs. In this paper, we extend our previous work [29] and present PandORA, a large-scale evaluation and profiling of DRL agents for Open RAN, leveraging a framework to automate the training of DRL agents and their on-boarding as xApps to be executed in the near-real-time RIC. This all-in-one solution, spanning from the automated end-to-end streamlining of DRL model training, to testing on an Open RAN network, is enabled by the Colosseum testbed. Moreover, we extend the analysis in [29] by considering two new directions that affect DRL-based xApp design. Specifically, (i) we investigate the trade-offs between using a global DRL agent that takes decisions for all slices against the case of training and deploying per-slice dedicated xApps; and (ii) we analyze the effect of RAN control timing (i.e., the temporal granularity we use to compute and enforce new control actions). We explore the aforementioned directions by training a variety of DRL agents, which we then onboard on xApps. We test these agents both under network conditions similar to those encountered in the training dataset (i.e., in-sample experimental evaluation), as well as under previously unseen network conditions not encountered during the training process (i.e., out-of-sample experimental evaluation).

The remainder of this paper is organized as follows. Section II describes the PandORA System Model and Evaluation Framework. Section III presents the different DRL optimization strategies considered in this work, while Section IV details our experimental setup and training methodology, through PandORA. Experimental results are discussed in Sections V, VI and VII. Finally, Section VIII draws our conclusions.

II. The PandORA System Model and Evaluation Framework

Before providing thorough implementation details of the PandORA system design and its operational procedures (Section II-B), we first briefly introduce the O-RAN system model considered in this paper (Section II-A).

A. System Model and Reference Use-Case Scenario

In this work, we consider an Open RAN multi-slice scenario where 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.

We focus on the case where a set of xApps execute in the near-real-time RIC to intelligently control the resource allocation process in a way that satisfies the diverse QoS demands required by each slice. These xApps take these control decisions

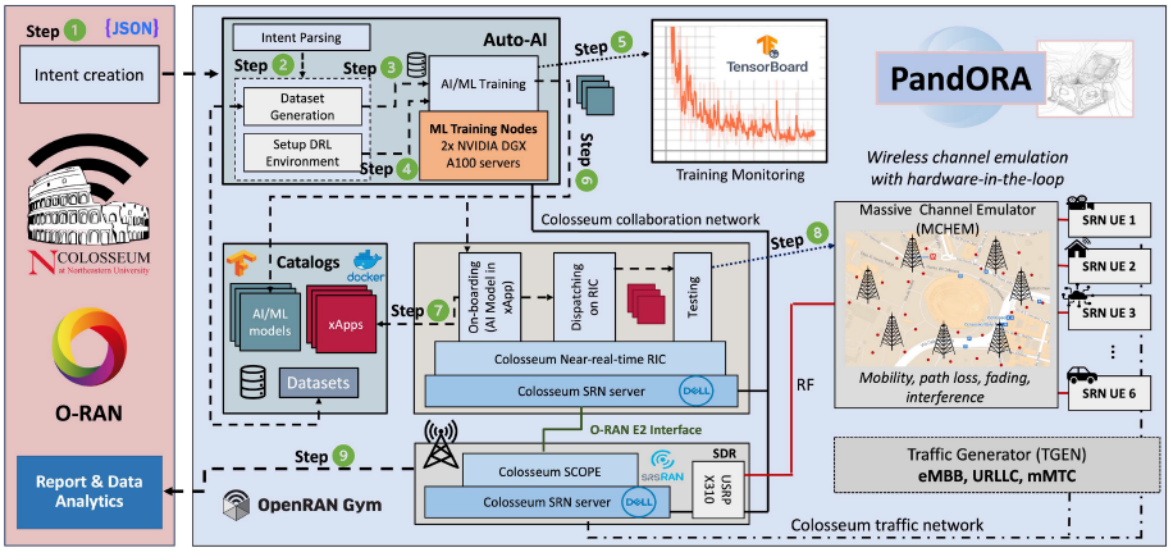


Fig. 1. PandORA framework for intent-driven DRL training, xApp on-boarding, and testing with Open RAN in Colosseum.

by leveraging AI/ML algorithms that can control Base Station (BS) parameters and functionalities such as RAN slicing (i.e., the portion of available PRBs that are allocated to each slice at any given time) and Medium Access Control (MAC) layer scheduling policies (i.e., how such PRBs are internally allocated to users belonging to each slice). It is worth mentioning that xApps can assign a certain scheduler profile (to be selected among Round Robin (RR), Waterfilling (WF) and Proportionally Fair (PF)) to each slice, but do not take Transmission Time Interval (TTI)-level scheduling decisions, which is instead left to the evolved Node Base (eNB). xApps make decisions based on UE traffic demand, load, performance and network conditions that are given by Key Performance Measurements (KPMs) periodically reported by the RAN.

B. PandORA Overview and Procedures

The architecture and building blocks of PandORA are illustrated in Fig. 1. These are key to our extensive evaluation campaign, as they streamline the DRL design and xApp evaluation. PandORA leverages automated pipelines that cover the end-to-end lifecycle of AI development for O-RAN systems. Specifically, it embeds the following components and functionalities:

- *Catalogs:* These include AI/ML models trained offline, xApps, and datasets that can be used to train and test data-driven solutions.
- *Intent-driven Auto-AI Engine:* This is a fully-customizable software component that parses training intents defined in JSON format and automatically extracts the necessary data from datasets in the catalog. Once the training process is instantiated, AI solutions are trained based on the intent, and the AI training is configured to match the desired inputs, outputs, and goals. It also offers web-based dashboards (e.g., TensorFlow's TensorBoard [30]) to monitor the training phase.
- *xApp On-boarding Module:* Templates and pipelines are provided to convert trained AI solutions into xApps that are subsequently published to the catalog. These xApps are composed of two interconnected units, namely a service model connector and a data-driven logic unit. The former is responsible for handling the communication with the near-real-time RIC (e.g., extracts live Key Performance Indicators (KPIs) from the RAN nodes or sends control actions, via the E2 termination), while the latter is tasked with embedding trained AI solutions [27].
- *xApp Dispatcher Module:* This module automates the instantiation of xApps in order to facilitate their *testing* process. This is done through scripts that automatically create Docker containers of the xApps to dispatch and deploy them on the near-real-time RIC.
- *Integration with OpenRAN Gym and Colosseum:* PandORA is seamlessly integrated with OpenRAN Gym [27] and Colosseum [28] to enable end-to-end O-RAN-compliant testing of xApps in a reproducible RF environment. Via OpenRAN Gym, PandORA also offers data collection and analytics functionalities that are useful to validate xApp behavior and evaluate their performance under diverse conditions and deployments.

In a nutshell, PandORA users (e.g., telco operators, xApp developers) can generate an intent in JSON format (Step 1in Fig. 1) by specifying the control objective (e.g., maximizing a certain set of KPMs for each slice), desired control parameters and observable inputs (e.g., throughput measurements, Channel Quality Information (CQI), to name a few). PandORA parses the intent (Step 2) and: (i) produces a dataset to be used to train the AI algorithms (Step 3); (ii) configures the AI algorithm to be trained, e.g., a DRL agent based on the Proximal Policy Optimization (PPO) architecture controlling slicing policies to maximize a certain reward function (Step 4); and (iii) trains the AI solution while offering a dashboard to monitor the training phase (Step 5). Once the training phase is complete, the AI/ML models are published to the Catalog, and PandORA integrates with OpenRAN Gym [27] to convert the trained AI solution into an xApp (Step 6), which then gets published to the respective xApp Catalog (Step 7). Once the publishing is complete, the xApp can be tested on Colosseum under one or more Colosseum scenarios (Step 8), and testing data is collected and stored to enable performance evaluation, xApp validation and data analytics (Step 9).

PandORA is developed in Python using Tensorflow libraries [30]. Although it supports the training of any type of AI/ML model, for the scope of this work, we primarily focus on training DRL agents which are the state-of-the-art for intelligent decision making in O-RAN [12], [31], [32], [33].

Following O-RAN specifications and requirements [1], PandORA trains AI solutions offline using an extended and customized version of the `tf_agents` library [34] where the configuration of the training environment, the DRL agents and the datasets are generated at run-time starting from a JSON-based intent file.

A simplified example of a minimal intent document is shown in Listing 1. The example shows how it is possible to specify the slices to be controlled by the agent, the actions that can be controlled, the KPIs for each slice that constitute the observation of the state, and which should be considered to compute the reward. PandORA *intents* use a modular approach where PandORA users can specify slice-specific rewards and set a global reward to combine them. In Listing 1, we illustrate a case where a DRL agent is trained to maximize both the average throughput and number of transmitted packets in the Downlink (DL) for the eMBB slice (Lines 4–9), while maximizing the average number of DL transmitted packets for the mMTC slice (Lines 10–15) and minimizing the maximum DL buffer occupancy for the URLLC slice (Lines 16–21).[1]

These three slice-specific rewards are then combined using a global reward that aims at maximizing the weighted sum of the individual reward contributions for each slice (weights are defined at Line 25).

Users can specify a general control action space (e.g., controlling RAN slicing policies and scheduling profile as shown in Line 23) without the need to specify the explicit values of these control actions. At run-time, PandORA parses the JSON-formatted intent and generates a dataset that fulfills the intent. This dataset includes all necessary KPMs required to compute observations and rewards. Additionally, it generates the proper action space from the available datasets. For instance, PandORA processes these datasets, automatically determining the available scheduling profiles (e.g., RR, WF, and PF as described in Section II-A), along with the RAN slicing policies suitable for the given slices. More advanced PandORA users can also specify the subset of actions that must be considered by the agent. For example, they can force the agent to only consider PF and RR schedulers and exclude WF.

Apart from offering modules to generate custom reward functions, PandORA provides a set of pre-defined reward functions that can be selected and combined to generate a global reward. This includes rewards aimed at maximizing or minimizing specific metrics through average, maximum, minimum, and median values. It enables the prioritization of certain KPMs by configuring weights, as demonstrated in Line 25. PandORA also offers pre-configured slice configurations with tailored rewards and observable KPMs that users can utilize. These configurations serve as default values for each slice, and they can be overrided by the user.

It is also worth noting that intents, slice configurations, rewards, actions, and observation configurations are modular and can be combined to generate new intents. In this way, new intents specific to each slice are created, which are later combined, providing further flexibility to PandORA and its users.

Listing 1. A simplified example of a JSON intent for PandORA

```
1 "{
2 "intent": {
3   "slices": [
4     {
5       "name": "embb",
6       "reward": "MaxAverageReward",
7       "reward_KPIs": ["dl_brate",
        "dl_tx_pkts"],
8       "observation_KPIs": ["dl_buffer",
        "dl_tx_pkts"]
9     },
10    {
11      name": "mmtc",
12      "reward": "MaxAverageReward",
13      "reward_KPIs": ["dl_tx_pkts"],
14      "observation_KPIs": ["dl_brate",
        "dl_tx_pkts"]
15    },
16    {
17      "name": "urllc",
18      "reward": "MaxElemReward",
19      "reward_KPIs": ["dl_buffer"],
20      "observation_KPIs": ["dl_buffer",
        "dl_brate"]
21    }
22  ],
23  "actions": ["scheduling",
    "ran_slicing"],
24  "global_reward_type": "Nested-
    SumWeightedReward",
25  global_reward_weights": [0.5,  0.25,
    -1]
26 }
27 }
```

Via `tf_agents` [34], PandORA provides access to a range of DRL agents, including DQN, PPO, Deep Deterministic Policy Gradient (DDPG), Twin Delayed DDPG (TD3), and various others. Users can utilize PandORA to select the type of agents they want to use, along with specifying hyperparameters. Once the selection is complete, PandORA configures the agents at run-time, adjusting action and observation spaces as well as rewards based on the specified intent. Similarly to intents, users have the flexibility to customize their DRL architecture and hyperparameters or use default values pre-configured in PandORA.

C. DRL Agent Architectures Tested in This Work

To control RAN slicing, and scheduling, we focus on data-driven approaches that rely on RL. In RL, an *agent* iteratively interacts with the *environment* by performing *observations*, in order to learn the optimal control *policy* that maximizes the desired cumulative *reward*. More specifically, the *agent* explores the *environment* and takes *actions* in several environmental *states*, without knowing a priori which actions are more beneficial, and eventually learns the best policy through experience.

[1]Note that the weight associated to the URLLC slice is configured in Line 25 and has negative value. Therefore, the `MaxElemReward` function in Line 18 effectively results in minimizing the maximum buffer occupancy. It is worth mentioning that the eNB does not have direct access to end-to-end latency measurements, therefore we use the buffer occupancy as a proxy for latency [29].

The *reward* is a metric that defines the effectiveness of an *action*, while a sequence of *states*, *actions*, and *rewards* that result in a terminal *state* is called an *episode*. *State Space* S, represents all the possible states of the environment $s \in S$, while *Action Space* A defines all the feasible actions $a \in A$ that can be taken by the *agent*. Finally, in this work, we focus on DRL agents due to their ability to learn directly from experience, without relying on pre-existing models or explicit knowledge of the wireless environment [35], [36], [37] and therefore of its complex and dynamically changing conditions. Moreover, we focus on discrete actions and, for this reason, we limit our analysis to PPO [38] and DQN [39] architectures which are two state-of-the-art algorithms respectively for on-policy and off-policy DRL for discrete action spaces.

PPO has been demonstrated several times to outperform other architectures [4], [11]. It is based on an actor-critic network architectural approach, where 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 Clipped Surrogate Objective function used by PPO is defined in (1)

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(q_t(\theta)\hat{A}_t, \text{clip}\left(q_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t \right) \right], \tag{1}$$

where $\hat{\mathbb{E}}_t$ represents the empirical average; $\hat{A}_t$ is the estimator of the advantage function denoted as A_t, which assesses how well an action performed compared to the expected performance under the current policy; and $q_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$. This ratio represents the probability of taking action a_t at state s_t following the current policy π_θ, divided by the respective probability when following the previous policy (i.e., $\pi_{\theta_{\text{old}}}$). When multiplied by the estimator of the advantage function at time-step t (i.e., $\hat{A}_t$), the unclipped part (1) is obtained, as shown in (2), where Conservative Policy Iteration (CPI) stands for the technique leveraged to avoid large policy updates [40].

$$L^{\text{CPI}}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \mid s_t\right)} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[q_t(\theta)\hat{A}_t \right]. \tag{2}$$

Since maximizing (2) directly would result in large policy updates, by *clipping* the probability ratio, we constrain the surrogate objective function (i.e., (1)) and remove the incentive to move the probability ratio (i.e., $q_t(\theta)$) outside the interval $[1-\epsilon, 1+\epsilon]$. ϵ is a hyperparameter that determines the clip range and is chosen in a way to ensure that the policy update will not be too large, indicating a significant divergence between the old and the current policy. Lastly, we take the minimum of the clipped and non-clipped objectives. Therefore, the final objective serves as a lower pessimistic bound of the unclipped objective. This indicates that we select either the clipped or the non-clipped objective based on $q_t(\theta)$ and the advantage. The final form of the Clipped Surrogate Objective Loss for the actor-critic implementation of PPO is given in (3). It is a combination of the Clipped Surrogate Objective function (i.e., L_t^{CLIP}), the squared-error value loss function (i.e., $L_t^{\text{VF}}(\theta) = (V_\theta(s_t) - V_t^{\text{targ}})^2$) which is the difference between predicted and target cumulative reward estimations, and the entropy bonus (i.e., S''), with the latter ensuring sufficient exploration, while c_1, c_2 are control parameters.

$$L_t^{\text{CLIP+VF+S}}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{\text{CLIP}}(\theta) - c_1 L_t^{\text{VF}}(\theta) + c_2 S''\left[\pi_\theta\right](s_t) \right] \tag{3}$$

In addition, based on a comparative analysis between models with diverse number of layers and neurons, the actor and critic networks we select for this work both consist of 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}.

The **DQN** algorithm leverages Q-learning principles to estimate the Q-function. The Q-function represents the expected discounted cumulative reward obtained by taking action a in state s and then following the policy π. The optimal state-action pair is computed using the Bellman Equation, defined in (4)

$$Q^*(s,a) = \left[r + \gamma \max_{a'} Q^*\left(s', a'\right) \right]. \tag{4}$$

This equation is approximated by a DQN that calculates the cumulative future reward, denoted as r_t at time t, and discounted by a factor $\gamma \in [0,1]$, while s' and a' represent the state and the action taken in the next time-step. In addition, the algorithm uses a replay buffer to store experience and to cope with instabilities caused by correlation between consecutive episodes. The set of experiences is denoted as $D_t = \{e_1, \ldots, e_t\}$, where each $e = (s, a, r, s')$ is a tuple that represents the state, action, reward, and the next state. The system transitions to the next state s' after the agent takes an action a at each time-step t during the training phase, while the experience vector is stored in the replay buffer. In order to reduce the correlation between the Q-function value and the optimal Q^* value, a second Q-Network is employed, namely the target Q-Network. Both the main and target Q-Network share the same structure, while the weights of the latter are periodically updated to match those of the former. During the training phase, the replay buffer is used to randomly select batches of previous experiences. These experiences are then utilized to calculate the DQN weights, denoted as θ_t in (5). This calculation involves minimizing the loss function using Stochastic Gradient Descent (SGD) method. Concludingly, the loss equation leveraged during the Q-Learning update at iteration t is given by (5)

$$L_t\left(\theta_t\right) = \mathbb{E}_{s,a,r,s'} \left[\left(r + \gamma \max_{a'} Q\left(s', a'; \theta_t^-\right) - Q(s, a; \theta_t) \right)^2 \right], \tag{5}$$

where θ_t^- are the weights of the target Q-Network, and γ is the discount factor that prioritizes instantaneous rewards against long-term rewards. Additionally, the ϵ_{DQN}-greedy is employed, where the DRL agent's choice of action depends on the parameter $\epsilon_{\text{DQN}} \in [0,1]$. In detail, with a probability of $1 - \epsilon_{\text{DQN}}$, an action computed by the DQN is selected. Conversely, with a probability of ϵ_{DQN}, the agent selects a random action from the action space A. This strategy effectively balances the exploration of new actions, randomly selected, and the exploitation of the agent's learned knowledge, based on actions selected by the DQN. In our analysis, the latter is implemented as a feed-forward multi-layer Neural Network (NN), consisting of 5 hidden layers of 50 neurons each. The learning rate is set to 10^{-3}, while the discount factor is $\gamma = 0.95$. Finally, the replay buffer memory size is set to 10,000, while $\epsilon_{\text{DQN}} = 0.1$.

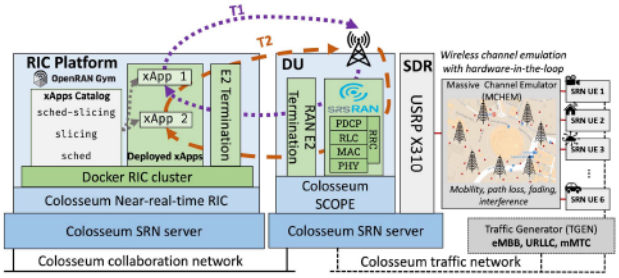


Fig. 2. Reference O-RAN testing architecture with focus on the case of two xApps operating at different time scales, T_i, as described in Section V-B.

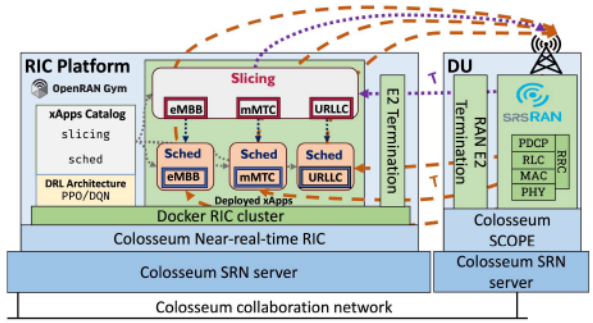


Fig. 3. Reference O-RAN testing architecture with focus on the case of four xApps operating at time scale, T , as described in Section V-C.

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., DL 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. Although the channel state information (e.g., CQI, Modulation and Coding Scheme (MCS)) is not directly fed to the AI/ML agents, it is worth mentioning that its effect on performance is indirectly captured by the KPIs used to feed the DRL agents (e.g., Throughput, Buffer Occupancy, Transmitted Packets), which has been shown to be sufficient to implicitly capturing channel effects on network performance [4].

In the case of the joint-slice optimization, 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 (6)

$$R = \sum_{t=0}^{\infty} \gamma^t \left(\sum_{j=1}^{N} w_j \cdot r_{j,t} \right), \tag{6}$$

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. 2.

In the case of the per-slice optimization, the respective weighted average reward function of the DRL agent is given as follows in (7)

$$R = \sum_{t=0}^{\infty} \gamma^t \left(w_j \cdot r_{j,t} \right), \tag{7}$$

The models that we have designed and trained are deployed as xApps on the near-real-time RIC, as illustrated in Fig. 3.

III. DRL Optimization Strategies

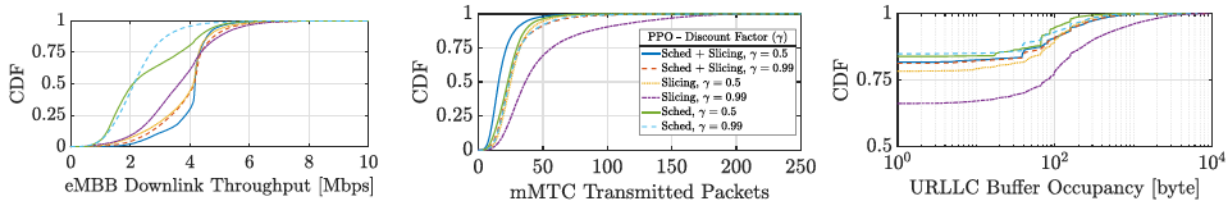
Our goal is to investigate how different design choices affect the effectiveness and decision-making of the DRL-based xApps. For this reason, we consider the following design choices, which resulted in the generation and testing of a total of 23 xApps.

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. 2.

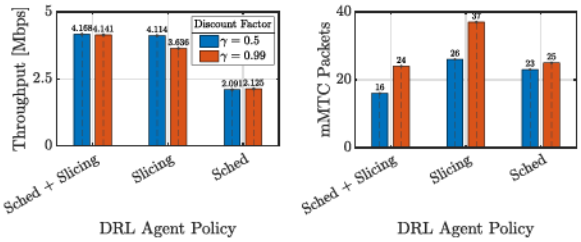
Per-Slice Scheduling Profile Selection: We explore a hierarchical decision-making configuration of four xApps operating simultaneously at the same time granularity and reconfiguring the BS's control parameters. In detail, one slicing xApp simultaneously allocates PRBs to all slices. The remaining three xApps are exclusively assigned one slice each and select a dedicated scheduling profile for each slice. In this way, we aim to examine the effects of multiple xApps operating at the same timescale, and to compare a widely used state-of-the-art off-policy architecture (i.e., the DQN), with its on-policy counterpart (i.e., the PPO algorithm). A practical example is depicted in Fig. 3, while the results of this investigation are presented in Section V-C.

Impact of Reward's Weights: We test different values for the weights w_i of the slices in (6). A different weight configuration affects how DRL agents prioritize each slice. The results of this analysis are reported in Section V-D, where we show how weights significantly impact the overall performance and can result in inefficient control strategies.



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

Fig. 4. Performance evaluation under different action spaces and values of the γ parameter with the PPO DRL Architecture.



(a) eMBB DL Throughput (b) mMTC Packets

Fig. 5. Median values under different action spaces and values of γ with the PPO DRL Architecture.

TABLE I
TRAFFIC PROFILES

Profile Name	eMBB [Mbps]	mMTC [kbps]	URLLC [kbps]
Profile 1	1	30	10
Profile 2	4	44.6	89.3

Effect of RAN Control Timers: Finally, we aim at understanding how different RAN control timers (i.e., different periodicities for RAN telemetry, report and control) will have an impact on the decision-making process and how different control sets will prioritize one slice over the other. Specifically, we look into three control timers namely the KPM log time, the Distributed Unit (DU) Report Timer, and the action update time, i.e., how frequently the actions sent back by the RIC get updated and enforced by the BS. The reported findings of this strategy are provided in Section V-E.

It is reminded that the action set A consists of both scheduling and RAN slicing policies. In our analysis, we consider the cases where an agent can control either scheduling or slicing decisions individually, or control both jointly. The state S is represented by the output of the autoencoder which is used to compress input KPIs collected over the E2 interface and convert them into latent representations. Finally, although in our analysis we evaluate diverse reward designs that consider long-term and short-term goals, and combine diverse target KPIs for each slice, the general form of the reward R considered in all of our DRL agents is defined in (6).

IV. EXPERIMENTAL SETUP AND DRL TRAINING

To experimentally evaluate the DRL agents, we leverage the capabilities of OpenRAN Gym [27], 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 [41] softwarized open-source protocol stack;
- Large-scale data collection, testing and fine-tuning of RAN functionalities through the SCOPE framework [42], 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.

TABLE II
WEIGHT CONFIGURATIONS

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

We deploy OpenRAN Gym on Colosseum [28], a publicly available testbed with 128 Standard Radio Nodes (SRNs), i.e., pairs of Dell PowerEdge R730 servers and NI Universal Software Radio Peripheral (USRP) X310 Software-defined Radios (SDRs). Colosseum enables large-scale experimentation in diverse Radio Frequency (RF) environments and network deployments. The channel emulation 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 (e.g., path loss, fading, attenuation, interference of signals) modeled a priori through ray-tracing software, analytical models, or real-world measurements. Similarly, the Colosseum Traffic Generator (TGEN), built on top of the Multi-Generator (MGEN) TCP/UDP traffic generator [43], emulates different traffic types, 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, we consider two different topologies both located in the urban environment of Rome, Italy [42]. Specifically, **Location 1** concerns UEs uniformly distributed within a 50 m radius from the BS, while in **Location 2**, the corresponding UEs are placed within a 20 m radius from the BS.

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

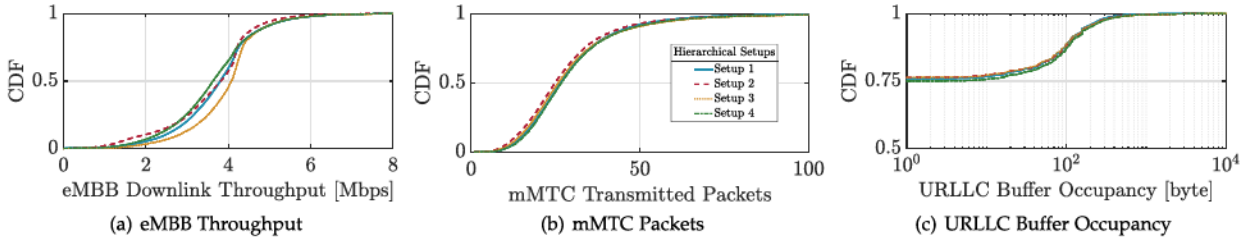


Fig. 6. Performance evaluation under different hierarchical configurations with the PPO DRL Architecture.

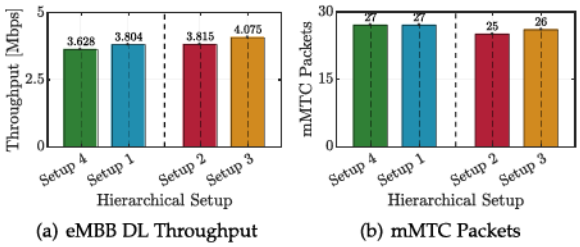


Fig. 7. Median values under different hierarchical configurations with the PPO DRL Architecture.

TABLE III
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`

to each slice. Slice-based traffic is generated following the specifications reported in Table I. Specifically, we consider two different traffic profile configurations (i.e., Profile 1 and Profile 2) with different source bitrates. However, for both profiles, we consider a constant bitrate traffic for eMBB users, while URLLC and mMTC UEs generate traffic based on a Poisson distribution.

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 [44], which do not permit the deployment of untrained data-driven solutions, we train our ML models *offline* on Colosseum's GPU-accelerated environment, which includes two NVIDIA DGX A100 servers with 8 GPUs each. Notably, AI/ML solutions should be trained offline to avoid actions that can potentially lead to performance degradation in the network [1]. Then, the trained DRL-agents are onboarded on xApps inside softwarized containers implemented via Docker and deployed on the ColO-RAN near-real-time RIC.

TABLE IV
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

V. IN-SAMPLE EXPERIMENTAL EVALUATION

In this section, we present the results of an extensive performance evaluation campaign, with more than 38 hours of experiments in Colosseum, 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 DL direction of the communication system. Last, all the experiments of the in-sample experimental evaluation campaign concern **Location 1** which is the configuration used to collect training data, and all the UEs are assumed static. Out-of-sample evaluation, i.e., testing the DRL agents against data collected in a entirely different network deployment, will be the focus of Section VI.

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. All the reported results were obtained with $\gamma \in \{0.5, 0.99\}$, while the reward's weight configuration is shown in Table II and identified as `Default`.

In Fig. 4, 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 different values of γ. The median of such measurements for the eMBB and mMTC slices is instead reported in Fig. 5. 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 III from best to worst performing.

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

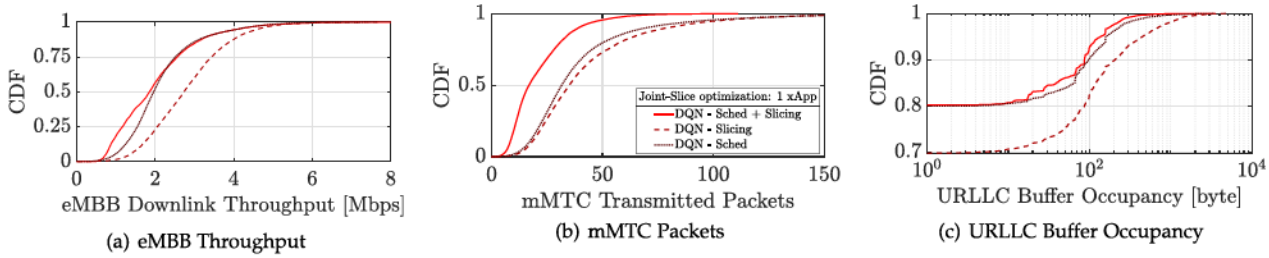


Fig. 8. Performance evaluation under the *Default* weight configuration for different actions spaces and discount factor $\gamma = 0.95$ with the DQN DRL Architecture.

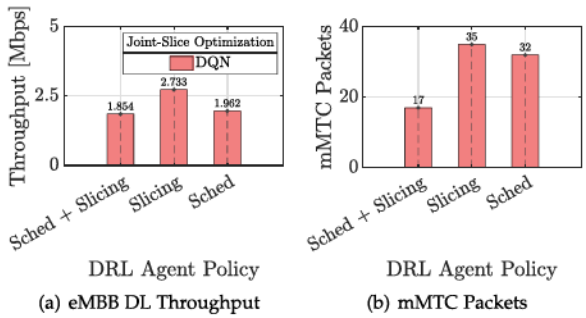


Fig. 9. Median values under the *Default* weight configuration for different actions spaces and discount factor $\gamma = 0.95$ with the DQN DRL Architecture.

TABLE V
RAN CONTROL TIMERS

Control Time	Set 1	Set 2	Set 3
DU Report	1 s	250 ms	100 ms
KPMs Log	250 ms	250 ms	100 ms
Action Update	250 ms	250 ms	100 ms

TABLE VI
FEASIBLE PRB ALLOCATION

eMBB	mMTC	URLLC
30	9	11
30	15	5
36	9	5
24	21	5
24	15	11
18	15	17
18	9	23
18	21	11
12	27	11
12	15	23
12	9	29
6	27	17
6	39	5
6	15	29
6	9	35
36	3	11

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 Fig. 5(a) and (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 III, 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 IV. 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 tens of 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. 6 and 7. From Fig. 6(a), *Setup 3* delivers the best eMBB performance, *Setups* 1 and 2 perform almost equally, while *Setup 4* performs the worst. For mMTC, in Fig. 6(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. 6(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. 7(a), we notice that *Setups* 2 and 3 deliver the highest eMBB throughput.

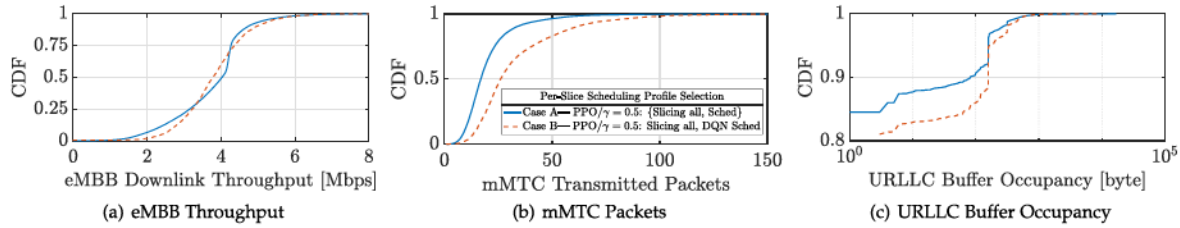


Fig. 10. Performance evaluation with 4 xApps and per-slice scheduling profile selection under PPO and DQN Architectures.

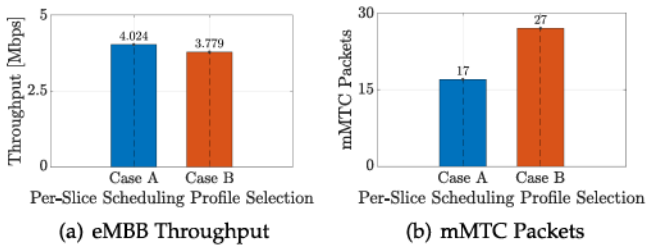


Fig. 11. Median values obtained with a 4-xApp setup and per-slice scheduling profile selection under PPO and DQN Architectures.

In Fig. 7(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 Per-Slice Scheduling Profile Selection

We extend our prior work [29] by evaluating the coexistence of a multitude of xApps delivering services to an Open RAN multi-slice scenario. Specifically, we consider the case where a single xApp distributes the available PRBs to all three slices, while three xApps are each tasked with selecting a dedicated scheduling profile for a single slice. All four xApps operate at the same timescale as given in Set 1 of Table V, where KPMs and actions are both logged and updated every 250 ms respectively, while fresh RAN data are reported by the DU within a granularity of 1 s.

We begin the evaluation by focusing on the DQN algorithm and the case of the joint-slice optimization. All results were produced with agents trained under the `Default` weight configuration of Table II and with a discount factor of $\gamma = 0.95$. The findings presented in Figs. 8 and 9 indicate that in the eMBB slice, DQN underperforms compared to PPO for both $\gamma \in \{0.5, 0.99\}$, with only `Slicing` delivering the highest reported throughput value at 2.733 Mbps, as reported in Fig. 9(a). With respect to mMTC, Figs. 8(b) and 9(b) depict a similar performance to the one shown in Figs. 4(b) and 5(b), when examining the actions separately under their respective discount factor (i.e., $\gamma \in \{0.5, 0.99\}$). In detail, we observe that in both cases, `Slicing` delivers the highest reported number of transmitted packets, followed by `Sched`, and `Sched & Slicing`. It is also shown that the `Sched` xApp, in the case of DQN, performs better in the mMTC compared to the case with PPO, delivering a median value of 32 transmitted packets, which is ~ 28 % more compared to `Sched 0.99` with PPO, as shown in Fig. 5(b). Finally, we observe that slicing is the action that delivers the best performance in both eMBB and mMTC. Indeed, the competitive behavior observed with PPO is mitigated in the case of DQN, as shown in Fig. 9, since the actions that deliver the highest eMBB throughput also deliver the highest number of mMTC transmitted packets. On the URLLC, all configurations once again delivered optimal performance, reporting a median buffer occupancy of 0 B.

Both DRL architectures are model free, i.e., they do not rely on an explicit model of the environment. Instead, the DRL agent learns directly by interacting with the environment and makes decisions through trial and error. PPO employs a trust region method [38], [45], that helps in achieving more stable training, compared to DQN which can suffer from divergence and slow convergence [46]. The trust region ensures that the old and current policy do not deviate significantly, which helps mitigate divergence issues. Actor and critic networks calculate the state-value, with the critic network reducing variance in action value estimates. This reduction in variance makes policy updates more reliable and exploration safer [47]. On the contrary, DQN does not implement trust regions and uses the target network to achieve stability during training [35]. In environments affected by noisy observations and stochasticity, the aforementioned features, combined with the model's ability to explicitly model the policy's probability distribution over actions, make PPO capable of adapting more efficiently to varying conditions. This ability to generalize better across tasks and environments can be achieved with DQN but might require more fine-tuning and adjustments when transitioning to new tasks. Furthermore, DQN relies on epsilon-greedy exploration, which can be less efficient, especially in complex environments [48], [49]. PPO, being an on-policy algorithm, manages to optimize the current policy directly based on the most recent experiences. On the other hand, DQN, an off-policy algorithm, estimates the value of actions independently of the current policy. On-policy methods often work better when the policy is changing during training, especially in stochastic environments. Therefore, in the latter case where rapid policy changes are observed, off-policy methods may struggle to deal successfully with them, and they may require more extensive replay buffers or further fine-tuning to handle noisy environments effectively [50], [51], [52].

Lastly, PPO has been shown to work more efficiently in problems with high-dimensional state and action spaces, as demonstrated in [53], owing to its policy-based and trust region approaches. In our work, the state and action spaces are high-dimensional. In detail, a total of 43 actions are considered when resource allocation is performed jointly for all slices. For scheduling, the size of the action space is 27, while for slicing, it is 16. The former size encompasses all possible combinations

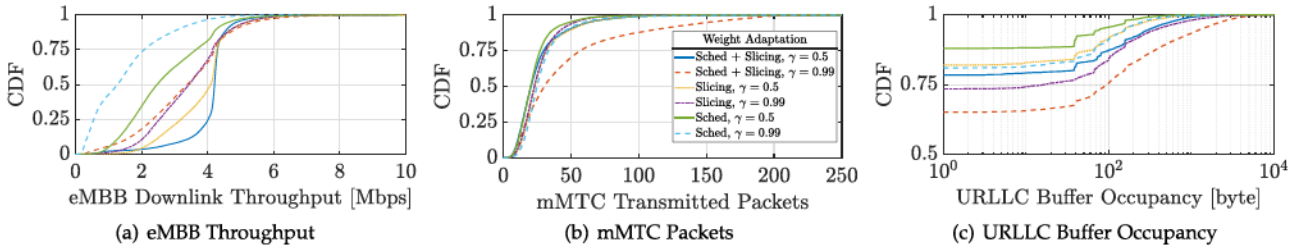


Fig. 12. Performance evaluation under the *Alternative* weight configuration for different actions spaces and discount factors with the PPO DRL Architecture.

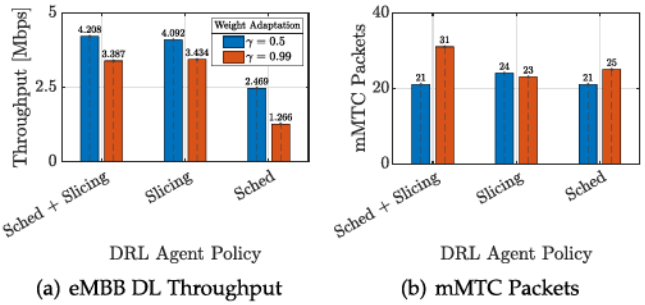


Fig. 13. Median values under the *Alternative* weight configuration for different actions spaces and discount factors with the PPO DRL Architecture.

of scheduling policies for all three slices (i.e., RR, WF, and PF), while the combination for PRB allocation is given in Table VI for a total of 50 PRBs. Finally, the observation of the state, obtained through srsRAN, consists of 10 independent measurements of 3 KPIs, i.e., DL buffer occupancy, the DL throughput and the number of transmitted packets on the DL.

From the previous discussion, we have seen that DQN performs poorly in the high-dimensional state and action spaces, such as the ones considered in this work. Therefore, we test its effectiveness in a reduced action space where the DQN agent only controls scheduling decisions. Specifically, we consider the case of one `Slicing` xApp that serves all slices simultaneously, while the remaining 3 xApps are dedicated to each slice and control the scheduling profile for the corresponding slice only. For instance, one xApp selects a scheduling profile for the eMBB slice, another one focuses on the mMTC slice, and the last one controls the URLLC slice only. This configuration allows for both testing and mixing different DRL architectures, such as PPO and DQN. It also enables experimentation with a setup in which the scheduling selection action follows the PRB allocation. Therefore, the preceding xApps will reconfigure a network where the action space is altered after the changes made by the `Slicing` xApp.

In Figs. 10 and 11, we present the results obtained from testing a total of 4 xApps that perform resource allocation. For the purposes of the experiments we consider two cases, namely *Case A* and *Case B*. In both cases, the `Slicing` xApp embeds a DRL agent trained with PPO under the discount factor, $\gamma = 0.5$. In *Case A*, the remaining `Sched` xApps also embed PPO agents trained with $\gamma = 0.5$. In *Case B*, the xApps embed DQN agents with a discount factor of $\gamma = 0.95$, as described in Section II-C. In Fig. 10, we observe that *Case A* performs better for the eMBB slice. Indeed, the results reported in Fig. 11 indicate that *Case A* achieves a median eMBB throughput value of 4.024 Mbps, which is $\sim 6.5\%$ higher than that of *Case B*. Regarding the mMTC slice, the median for the corresponding KPM metric was measured at 27 transmitted packets, representing an increase of $\sim 60\%$ compared to *Case A*. For URLLC, both cases delivered optimal performance with a median buffer occupancy of 0 B. We notice that *Case A* performs similarly to the case of jointly performing slicing and scheduling control with $\gamma = 0.5$, as shown in Figs. 4 and 5, by delivering one additional mMTC packet, as depicted in Fig. 11. With regards to *Case B*, the obtained results closely match those reported with Hierarchical Control under Setup 1in Fig. 7. Precisely, as depicted in Fig. 11(b), the two configurations achieve identical performance on the mMTC slice, by achieving the same median value of transmitted packets (i.e., 27 packets). For the eMBB slice, the setup with 4 xApps (i.e., *Case B*) exhibits a minimal drop of $\sim 0.66\%$ in the median throughput value compared to the performance achieved with Setup 1. Therefore, *Case B* provides a more flexible and heterogeneous setup where different DRL agents can coexist, without a significant performance degradation. Furthermore, the DQN can still be leveraged in those cases where agents independently optimize the performance of each slice thanks to the reduced action space.

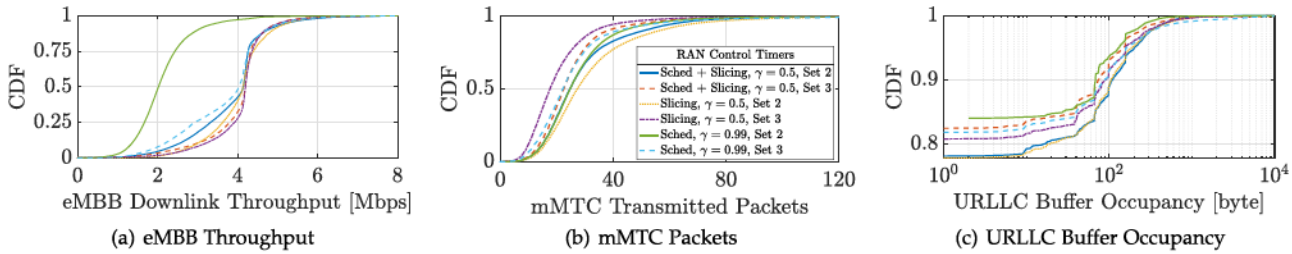
TABLE VII
WEIGHT DESIGN

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

D. Impact of Weight Configuration

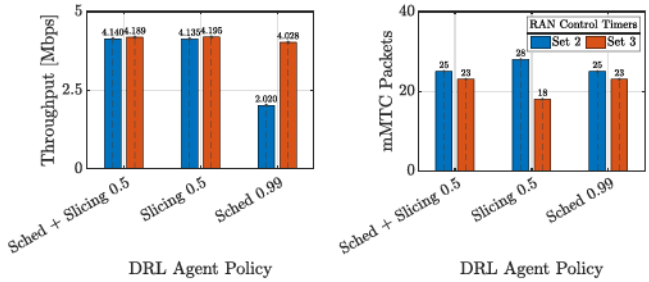
In this study, we consider different weight configurations to compute the cumulative average reward function in (6). The considered configurations are reported in Table II. The *Alternative* weight configuration is computed by using the weights in Table VII, where $A, B, C, \alpha_{eMBB}, \beta_{mMTC}$, 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, 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.



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

Fig. 14. Performance evaluation under different action spaces, values of the γ parameter and sets of RAN control timers with the PPO DRL Architecture.



(a) eMBB DL Throughput (b) mMTC Packets

Fig. 15. Median values under different action spaces and RAN control timers with the PPO DRL Architecture.

TABLE VIII
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 IX
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

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 II. 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× larger than the *Default* configuration.

Results for the *Alternative* configuration are reported in Figs. 12 and 13. In Fig. 12(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. 12(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`.

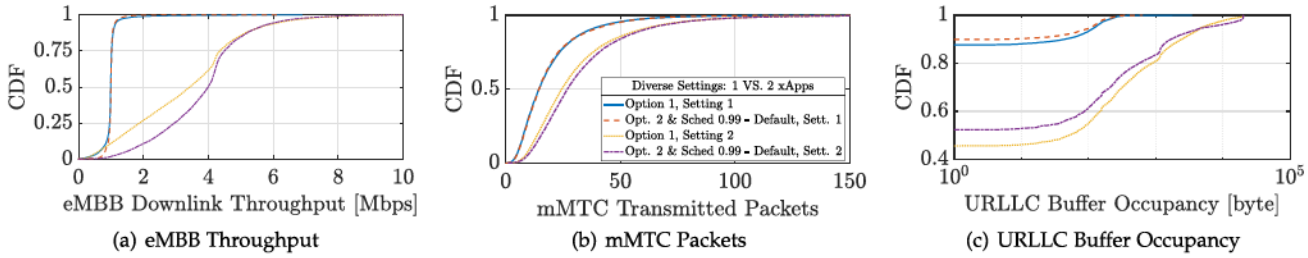
Fig. 13(a) and (b), confirm that controlling scheduling alone does not improve performance in general. Similarly to our previous analysis, a high eMBB performance (i.e., `Sched & Slicing 0.5`) results in a degraded mMTC performance. However, if compared with the `Default`, the `Alternative` weight configuration achieves a 31.25% increase for mMTC, with the same equally good URLLC performance and a 1% throughput increase for eMBB users.

In Table VIII we summarize the design options that have delivered a good overall performance so far. Table IX 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.

E. Impact of RAN Control Timers

We look into the case where we control three different RAN control timers, as reported in Table V, with the focus primarily steered onto Sets 2 and 3. In the latter cases, the KPM collection granularity (i.e., KPMs Log time) matches both the Action Update and the KPM report granularities on the BS's DU. In detail, KPMs are directly streamed into the RIC, upon collected, with an action being sent back through the E2 termination. This is in contrast with the case of Set 1, where the DU report time is equal to 1s, and hence some of the freshly captured data may not be reported. The findings of this evaluation are presented in Figs. 14 and 15, and all configurations concern DRL agents trained under the `Default` weight configuration presented in Table II.

In Fig. 14(a), it is observed that all configurations result in a comparable high performance, delivering a median throughput value of ∼4 Mbps (Fig. 15(a)) with the only exception of `Sched 0.99` under Set. 2 which delivers poor throughput performance. The aforementioned configuration yields a performance similar to the one reported in Fig. 5(a), where `Sched 0.99` is evaluated under the time granularities of Set. 1. Indeed, this combination achieves the same number of transmitted packets (i.e., 25 packets) with a negligible drop on the eMBB throughput, which is now reported at ∼2.02 Mbps, a value approximately 5% less than the 2.125 Mpbs reported with Set 1. Additionally, the `Sched 0.99` xApp, when tested under Set 3, attains the maximum reported throughput value for controlling scheduling using a single xApp, (reaching 4.028 Mbps, a ∼99% increase compared to 2.02Mbps achieved with Set 2). The latter is



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

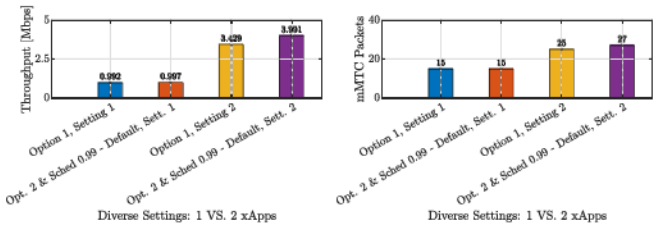
Fig. 16. Performance evaluation under mobility with Settings 1 and 2 of Table X and the PPO DRL Architecture.

achieved while transmitting 23 packets, which is two packets fewer than the maximum number of transmitted packets ever recorded when controlling scheduling alone, and with PPO-implemented agents. Based on these findings, scheduling should be controlled using small time granularities (i.e., $\sim$100 ms) to ensure high eMBB throughput, while guaranteeing no performance degradation on the mMTC slice. Furthermore, the results illustrated in Figs. 14(b) and 15(b) indicate that both scheduling and slicing, as well as scheduling under Sets 2 and 3, can achieve the same performance on the mMTC. However, `Sched & Slicing 0.5` delivers higher eMBB throughput, and hence, is preferred compared to `Sched 0.99`. In consistency with the reported findings so far, and when focusing on the three action profiles individually, we observe mMTC's and eMBB's competitive behavior (e.g., they heavily compete to gain radio resources). This trend is clearly illustrated in Figs. 14(a), 14(b), 15(a), and 15(b), where `Slicing 0.5` under Set 3 achieves the highest throughput value at 4.195 Mbps, but the lowest number of transmitted packets (i.e., 18 packets). The reported results in Fig. 15(a) and (b) also indicate that Set 3 primarily boosts eMBB's performance, compared to Set 2 which is a better fit for mMTC UEs. Finally, in Fig. 14(c), we note that all configurations for the investigated set of granularities deliver identical performance on the URLLC slice, and although not shown in the figures, they consistently maintain a median buffer occupancy of 0 B to ensure low latency. It is also observed that both `Sched & Slicing 0.5` and `Slicing 0.5` under Set 2, perform slightly better on the aforementioned slice, compared to `Sched 0.99` under Set 2, which performs slightly worse.

VI. Out-of-Sample Experimental Evaluation

In the Out-of-Sample experimental performance evaluation, we focus on the case of **Location 2**, and we test the framework's performance in static and mobile scenarios under two different traffic loads. Recall that our agents have been trained using data from **Location 1** only. Information regarding the traffic profiles and the UE mobility can be found in Tables I and X.

In Figs. 16 and 17, we present the results collected when testing the framework under mobile UE conditions and diverse traffic profiles for Settings 1 and 2 as defined in Table X. We juxtapose the configuration that delivered the highest eMBB throughput (i.e., 4.208 Mbps), among those tested in this work, namely Option 1: `Sched & Slicing 0.5 - Alternative` with the case of two xApps which jointly optimize the performance of all slices, namely Option 2: `Slicing 0.5`, and `Sched 0.99`. Finally, the set of granularities which are considered for this evaluation are given by Set 1 of Table V.



(a) eMBB Throughput (b) mMTC Packets

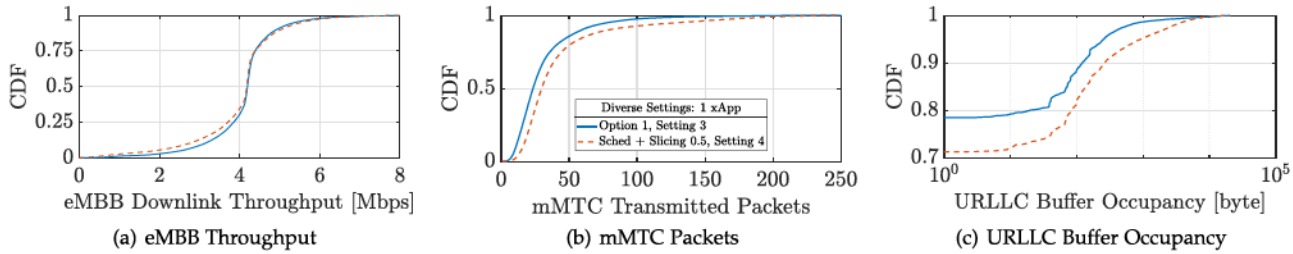
Fig. 17. Median values obtained under mobility with Settings 1 and 2 of Table X and the PPO DRL Architecture.

TABLE X
ML Agent & Network Condition Settings

Setting ID	UE Speed [m/s]	Traffic Load	RAN Control Timers	Weight Configuration
1	3	Profile 1	Set 1	`Alternative/Default`
2	3	Profile 2	Set 1	`Alternative/Default`
3	0	Profile 2	Set 1	`Alternative`
4	0	Profile 2	Set 2	`Default`

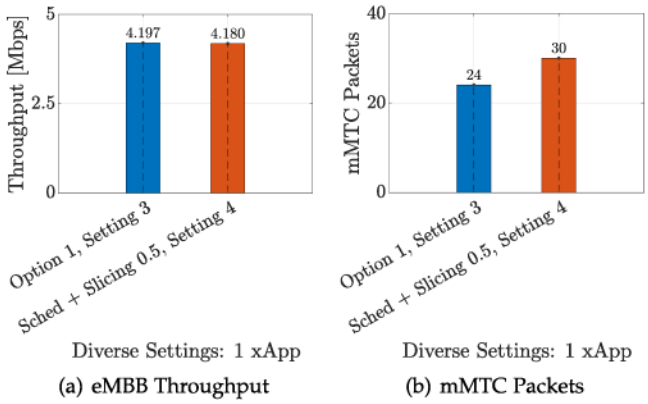
In Fig. 16(a), (b), and (c), we observe that under the traffic conditions of Setting 1, both configurations deliver almost identical performance. This is clearly depicted in Fig. 17(a) and (b), where the mentioned xApps achieve the same median values for eMBB throughput and number of mMTC transmitted packets. However, when the two configurations are tested under Setting 2 and specifically under Traffic Profile 2, a slight performance degradation is observed in the case of joint-slice optimization with 1 xApp. In detail, Figs. 16(a) and 17(a) indicate that the performance of eMBB is optimized when using 2 xApps, resulting in $\sim$ 16 % more throughput compared to the case with a single xApp. In the mMTC slice, both configurations deliver similar performance, as shown in Fig. 16(b). However, when focusing on the former setup, it results in delivering two more packets as shown in Fig. 17(b). On the URLLC, it is noted that most of the configurations under both Settings resulted in a median buffer occupancy of 0 B. However, in the case of joint optimization with 1 xApp, this value was reported at 52 B. Hence, in case of UE mobility, `Sched & Slicing 0.5 - Alternative` underperforms compared to joint optimization with `Slicing 0.5`, and `Sched 0.99`. Based on the reported findings, the simultaneous operation of two xApps can more successfully tackle possible UE disconnections caused by mobile conditions, and therefore enhance network performance.

In Figs. 18 and 19 we include the results collected when testing the optimization framework under Settings 3 and 4 in **Location 2**. In this evaluation, we once again compare Option 1: `Sched & Slicing 0.5 - Alternative` with



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

Fig. 18. Performance evaluation focusing on the case of joint-slice optimization with a single xApp and the PPO DRL Architecture under Settings 3 and 4 of Table X.



(a) eMBB Throughput (b) mMTC Packets

Fig. 19. Median values obtained when focusing on the case of joint-slice optimization with a single xApp and the PPO DRL Architecture under Settings 3 and 4 of Table X.

`Sched & Slicing 0.5 - Default`. It is noted that the main difference between these two setups pertains to the fact that the former is evaluated under granularity Set 1, while the latter under Set 2. This comparison allows us to assess how collecting and reporting KPMs as well as updating actions at smaller timescales (i.e., 250 ms) will impact the effectiveness of the optimization framework, under the traffic load defined in Profile 2. Additionally, the latter xApp is chosen due to the fact that, among all configurations evaluated in Section V-E in terms of RAN control timers, it has managed to provide the most significant performance boost in the network. In detail, when tested under Set 2, it has improved the performance of mMTC by ~ 56 %, achieving a median value of 25 packets, as shown in Fig. 15(b). This is in contrast to when it was evaluated under Set 1, where it delivered a median value of 16 packets (Fig. 5(b)). With regards to the eMBB slice, it still manages to maintain good overall performance, delivering a median throughput value of 4.140 Mbps (Fig. 15(a)).

The results reported in Figs. 18(a) and 19(a) indicate that both configurations deliver similar performance in the eMBB slice, with Option 1: `Sched & Slicing 0.5 - Alternative` performing slightly better. However, on the mMTC slice, `Sched & Slicing 0.5 - Default` performs better, as depicted in Fig. 18(b), by achieving a median value of 30 transmitted packets (Fig. 19(b)). URLLC UEs, once again, achieve zero latency (i.e., buffers are emptied), and, as a result, the respective figures are omitted.

TABLE XI
DRL REWARD DESIGN CATALOG FOR `Sched & Slicing 0.5` WITH PPO IN LOCATION 1 AND SET 1 OF RAN CONTROL TIMERS

Reward Design ID		eMBB	mMTC	URLLC
I	*Weight Config.*	1	1	1
	Slice Reward	Max. PRB Ratio	Max. PRB Ratio	Max. PRB Ratio
II	*Weight Config.*	72.0440333	0.5	0.00005
	Slice Reward	Max. DL Throughput	Max. PRB Ratio	Min. DL Buffer Occupancy
III	*Weight Config.*	72.0440333	0.229357798	0.00005
	Slice Reward	Max. DL Throughput	Max. DL TX Packets	Min. DL Buffer Occupancy

VII. BROADER EVALUATION OF PANDORA

The goal of this section is to provide additional insights into PandORA by: (i) considering a broader set of evaluation metrics aimed at shedding light on resource utilization and UE satisfaction; and (ii) testing across scenarios with a larger number of UEs. Specifically, in the latter case, we examine the performance of our xApps in a scenario where we add additional UEs to the the mMTC slice. In line with the previous sections, we will evaluate PandORA in setups previously seen during the training process (i.e., in-sample experimental evaluation in Location 1), as well as unseen conditions (i.e., out-of-sample experimental evaluation in Location 2). All subsequent results concern joint optimization with `Sched & Slicing 0.5` and are tested using Sets 1 and 2 of RAN control timers from Table V and the Traffic Profiles listed in Table I, both in static and mobile scenarios (e.g., 3 m/s).

It is noted that so far we have considered three main reward functions for the respective slices (i.e., maximization of DL eMBB throughput and mMTC TX Packets, and minimization of DL buffer occupancy in the URLLC). In order to demonstrate PandORA's capabilities as well as to compare the reward functions evaluated so far, we additionally craft two new reward designs, which are defined in Table XI. In *Reward Design I*, all slices are given the same priority (i.e., they are all assigned a weight of 1). The reward for all slices is the maximization of the PRB Ratio, defined as $\text{PRB Ratio} = \frac{\text{Sum of Granted PRBs}}{\text{Sum of Requested PRBs}}$, where $\text{PRB Ratio} \in [0, 1]$, that represents the amount of the allocated PRBs per slice. For *Reward Design II*, we consider the maximization of the eMBB throughput and the minimization of the URLLC buffer occupancy, with their weights defined in Table II under the `Default` weight configuration. For the mMTC slice, we consider the maximization of the PRB Ratio and the slice's weight is set to 0.5. Lastly, *Reward Design III* pertains to the joint scheduling and slicing optimization discussed and evaluated in Section V-A and shown in Table XI. The results of this experimental evaluation are illustrated in Figs. 20, 21, 22 and were collected under Traffic Profile 2 (i.e., 4 Mbps eMBB

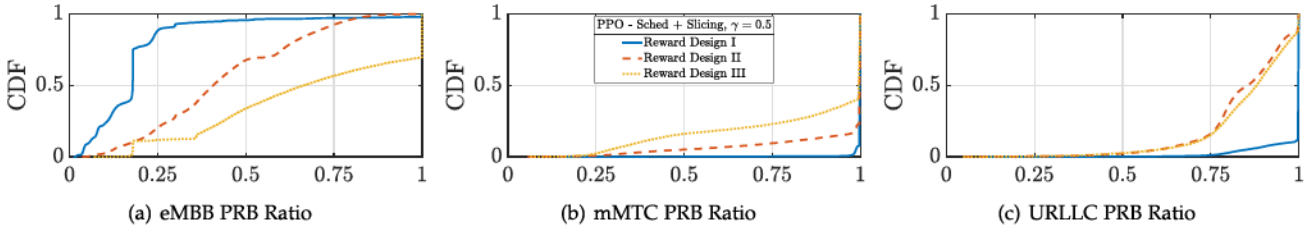


Fig. 20. UE Satisfaction expressed in the form of PRB ratio for the Reward Designs of Table XI.

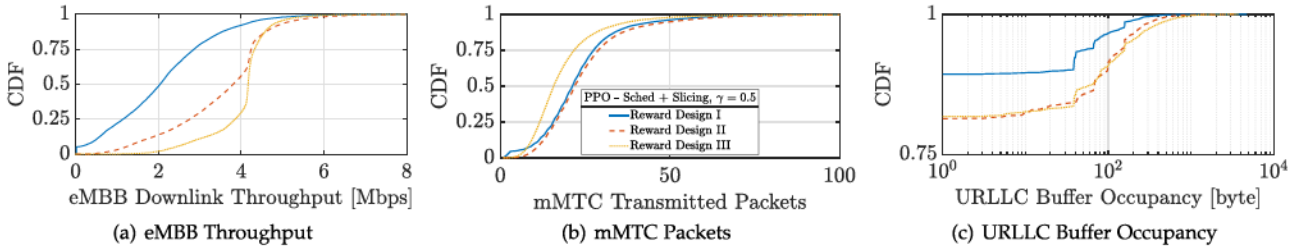


Fig. 21. Impact of the Reward Designs from Table XI on DL eMBB Throughput, DL mMTC and TX Packets.

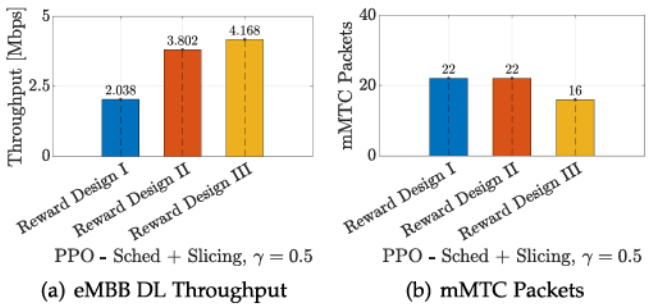


Fig. 22. Impact of the Reward Designs from Table XI on the median values of DL eMBB throughput, DL mMTC TX Packets, and DL URLLC buffer occupancy.

TABLE XII
XAPP CATALOG FOR `Sched & Slicing 0.5` WITH THE PPO DRL ARCHITECTURE

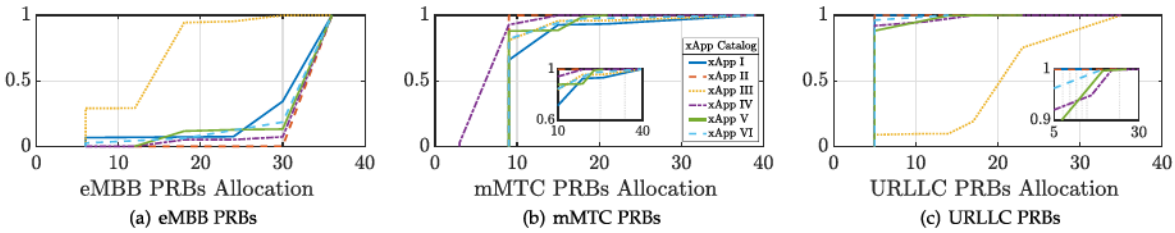
xApp ID		eMBB	mMTC	URLLC	Testing Conditions
I	*Weight Config.*	72.0440333	1.5	0.00005	*mobility, RAN Control Timer Set 1, Location 2*
	Slice Reward	Max. DL Throughput	Min. DL TX Packets	Min. DL Buffer Occupancy	
	Traffic	1 Mbps	30 kbps	10 kbps	
II	*Weight Config.*	72.0440333	0.229357798	0.00005	*RAN Control Timer Set 2, Location 2*
	Slice Reward	Max. DL Throughput	Max. DL TX Packets	Min. DL Buffer Occupancy	
	Traffic	4 Mbps	44.6 kbps	89.3 kbps	
III	*Weight Config.*	1	1	1	*RAN Control Timer Set 1, Location 1*
	Slice Reward	Max. PRB Ratio	Max. PRB Ratio	Max. PRB Ratio	
	Traffic	4 Mbps	44.6 kbps	89.3 kbps	
IV	*Weight Config.*	72.0440333	0.5	0.00005	*RAN Control Timer Set 1, Location 1*
	Slice Reward	Max. DL Throughput	Max. PRB Ratio	Min. DL Buffer Occupancy	
	Traffic	4 Mbps	44.6 kbps	89.3 kbps	
V	*Weight Config.*	72.0440333	0.229357798	0.00005	*RAN Control Timer Set 1, Location 1*
	Slice Reward	Max. DL Throughput	Max. DL TX Packets	Min. DL Buffer Occupancy	
	Traffic	4 Mbps	44.6 kbps	89.3 kbps	
VI	*Weight Config.*	72.0440333	1.5	0.00005	*RAN Control Timer Set 1, Location 1*
	Slice Reward	Max. DL Throughput	Max. DL TX Packets	Min. DL Buffer Occupancy	
	Traffic	4 Mbps	44.6 kbps	89.3 kbps	

throughput, 44.6 kbps mMTC throughput and 89.3 kbps URLLC throughput), in Location 1 (i.e, 50 m radius from the BS), and Set 1 of RAN control timers (see Table V), for a total of 6 UEs, uniformly distributed across the slices.

In Fig. 20, we show the PRB Ratio for the three Reward Designs defined in Table XI. We observe that with *Reward Design I*, eMBB UEs experience lower levels of *satisfaction* (Fig. 20(a)), in terms of the number of allocated PRBs. On the contrary, both on the mMTC and URLLC slices (Fig. 20(b) and (c)), *Reward Design I* results in significant UE satisfaction. The latter is due to the fact that both mMTC and URLLC have lower traffic requirements to satisfy (i.e., 44.6 kbps and 89.3 kbps throughput, respectively), compared to eMBB (i.e., 4 Mbps throughput). With *Reward Design II*, we observe an improvement in UE satisfaction levels on the eMBB, meaning that UEs are granted the resources they request, along with equally good performance in mMTC and URLLC. The results with *Reward Design III* indicate that this combination of weight configuration (i.e., tailored to prioritize one slice over the other) and slice-specific rewards effectively result in good overall performance without penalizing one slice over the other. Notably, *Reward Design III* is the configuration under which eMBB UEs experience more satisfaction. Finally, we can observe the competitive behavior among eMBB and mMTC. Indeed, the reward designs that delivers a higher UE satisfaction for eMBB (Fig. 20(a)) also delivers lower satisfaction for mMTC (Fig. 20(b)).

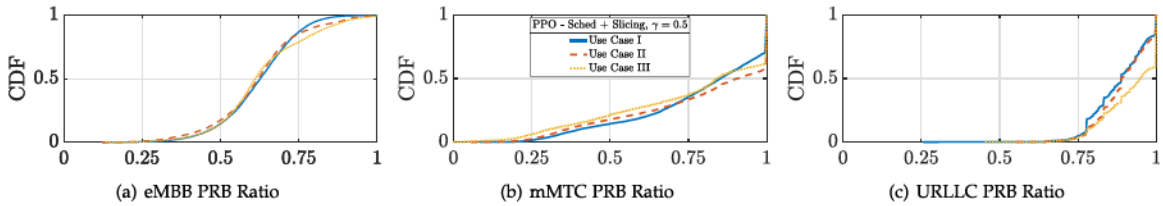
In Figs. 21 and 22, we observe the impact of the Reward Designs explored in Fig. 20 on the three KPIs of interest, namely DL eMBB throughput, mMTC TX packets, and URLLC buffer occupancy. In detail, Figs. 20(a), 21(a) and 22(a) demonstrate a similar trend, since a higher value of PRB Ratio yields in higher throughput. A similar trend is observed in Figs. 20(b), 21(b) and 22(b) for the mMTC slice, with *Reward Designs I & II* demonstrating similar performance (i.e., a median value of 22 TX packets as illustrated in Fig. 22(b)). These results indicate that the maximization of the PRB Ratio poses as ideal reward for the aforementioned slice but with a penalty on the eMBB. Specifically, even though *Reward Designs I & II* result in $\sim$ 37.5% improvement on the performance of the mMTC, they result in lower eMBB throughput values as depicted in Fig. 22(a). On the URLLC, all *Reward Designs* result in empty buffers, with *Reward Design I* slightly resulting in higher performance, as seen in Fig. 21(c).

In Fig. 23, we include the distribution of actions in terms of selected PRBs for the xApp Catalog presented in Table XII. The performance evaluation results indicate that agents can make decisions resulting in diverse action distributions (summarized



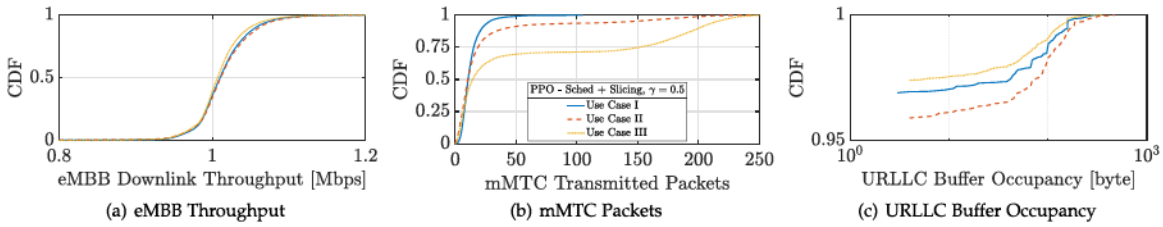
(a) eMBB PRBs (b) mMTC PRBs (c) URLLC PRBs

Fig. 23. Selection of PRB Actions for the three slices from the xApp Catalog of Table XII.



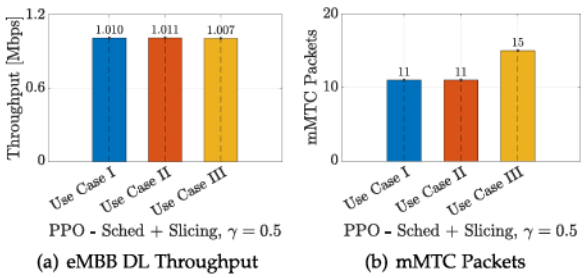
(a) eMBB PRB Ratio (b) mMTC PRB Ratio (c) URLLC PRB Ratio

Fig. 24. Resource Utilization and UE Satisfaction for a network deployment described by three different Use Cases.



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

Fig. 25. Impact of the network deployment described by three different Use Cases on DL eMBB Throughput, DL mMTC TX Packets, and DL URLLC Buffer Occupancy.



(a) eMBB DL Throughput (b) mMTC Packets

Fig. 26. Impact of the network deployment described by three Use Cases on the median values of DL eMBB Throughput and DL mMTC TX Packets.

in Table VI), which is due to varying rewards and design choices, impacting the performance achieved by the system.

We now consider the case where we increase the number of UEs allocated to the mMTC slice as illustrated by *Use Cases II & III*. Specifically, *Use Case II* involves a total of 7 UEs, while in *Use Case III*, the setup comprises a total of 8 UEs, with 3 and 4 UEs allocated to the mMTC slice, respectively. *Use Case I*, corresponds to a setup of 6 UEs in total, and all deployments have been tested under Traffic Profile 1 of Table I and Set 1 of RAN control timers of Table V in Location 2 (i.e., 20 m radius from the BS). Finally, the xApp under evaluation corresponds to *Reward Design III* of Table XI.

Figs. 24, 25 and 26 demonstrate that even when the number of UEs increases, xApps trained using PandORA still deliver good performance, with all *Use Cases* resulting in good resource utilization (Fig. 24) for the three slices. Similarly, the improved UE satisfaction is reflected on the eMBB DL throughput (Fig. 25(a)), with all *Use Cases* enjoying a median throughput value of $\sim$1 Mbps (Fig. 26(a)). For the mMTC slice, and specifically in Fig. 25(b), we observe that as the number of UEs increase, the number of generated TX packets also increases, with *Use Case III* reporting the highest median value of 15 DL TX packets (Fig. 26(b)). In line with the results presented in the previous evaluation sections, the performance on the URLLC remains high (i.e., empty buffers), with *Use Case III* performing slightly better as seen in Figs. 24(c) and 25(c).

VIII. Conclusion

In this paper, we presented PandORA, a comprehensive evaluation with insights on how to design DRL agents for RAN control. PandORA leverages a framework to streamline and automate DRL agent training and xApp on-boarding for extensive evaluation and testing of DRL agents with Open RAN in Colosseum. We investigated the impact of DRL design 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 23 xApps trained using DRL agents with different actions spaces, architectures, reward design and decision-making timescales. Additionally, we tested the DRL agents under various traffic and mobility conditions, considering different hierarchical setups and time granularities in locations both

observed and unseen during the training phase. 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, and possibly different architectures. Additionally, we have explored how the fine-tuning of RAN control timers can boost the performance of various xApps tasked with altering different action spaces. Based on the reported findings, the PandORA evaluation shows that DRL agents adapt well to network conditions encountered during the AI/ML training phase as well as unforeseen conditions, while ensuring high system resource utilization.

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 Commun. Surv. Tut.*, vol. 25, no. 2, pp. 1376–1411, Second Quarter 2023.

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: 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," *Comput. Netw.*, vol. 206, 2022, Art. no. 108682.

[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 Trans. Mobile Comput.*, vol. 22, no. 10, pp. 5787–5800, Oct. 2023.

[5] S. D'Oro, L. Bonati, M. Polese, and T. Melodia, "OrchestRAN: Network automation through orchestrated intelligence in the open RAN," in *Proc. IEEE Conf. Comput. Commun.*, 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 *Proc. 15th ACM Workshop Wirel. Netw. Testbeds Exp. Eval. CHaracterization*, 2022, pp. 17–23.

[7] A. Kak, V.-Q. Pham, H.-T. Thieu, and N. Choi, "HexRAN: A programmable multi-RAT platform for network slicing in the open RAN ecosystem," 2023, *arXiv:2304.12560*.

[8] C.-C. Chen, C.-Y. Chang, and N. Nikaein, "Flexslice: Flexible and real-time programmable RAN slicing framework," in *Proc. IEEE Glob. Commun. Conf.*, Kuala Lumpur, 2023, pp. 3807–3812.

[9] R. Wiebusch, N. A. Wagner, D. Overbeck, F. Kurtz, and C. Wietfeld, "Towards open 6G: Experimental O-RAN framework for predictive uplink slicing," in *Proc. IEEE Int. Conf. Commun.*, IEEE, 2023, pp. 4834–4839.

[10] S.-P. Yeh, S. Bhattacharya, R. Sharma, and H. Moustafa, "Deep learning for intelligent and automated network slicing in 5G Open RAN (ORAN) deployment," *IEEE Open J. Commun. Soc.*, vol. 5, pp. 64–70, 2024.

[11] M. Kouchaki and V. Marojevic, "Actor-critic network for O-RAN resource allocation: XApp design, deployment, and analysis," in *Proc. IEEE Globecom Workshops*, 2022, pp. 968–973.

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

[13] M. Zangooei, M. Golkarifard, M. Rouili, N. Saha, and R. Boutaba, "Flexible RAN slicing in open RAN with constrained multi-agent reinforcement learning," *IEEE J. Sel. Areas Commun.*, vol. 42, no. 2, pp. 280–294, Feb. 2024.

[14] 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, 2022, Art. no. 5375.

[15] H. Zhang, H. Zhou, and M. Erol-Kantarci, "Team learning-based resource allocation for open radio access network (O-RAN)," in *Proc. IEEE Int. Conf. Commun.*, IEEE, 2022, pp. 4938–4943.

[16] H. Zhang, H. Zhou, and M. Erol Kantarci, "Federated deep reinforcement learning for resource allocation in O-RAN slicing," in *Proc. IEEE Glob. Commun. Conf.*, 2022, pp. 958–963.

[17] F. Rezazadeh, L. Zanzi, F. Devoti, H. Chergui, X. Costa-Pérez, and C. Verikoukis, "On the specialization of FDRL agents for scalable and distributed 6G RAN slicing orchestration," *IEEE Trans. Veh. Technol*, vol. 72, no. 3, pp. 3473–3487, Mar. 2023.

[18] N. C. Luong et al., "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Commun. Surv. Tut.*, vol. 21, no. 4, pp. 3133–3174, Fourth Quarter 2019.

[19] A. Alwarafy, M. Abdallah, B. S. Ciftler, A. Al-Fuqaha, and M. Hamdi, "Deep reinforcement learning for radio resource allocation and management in next generation heterogeneous wireless networks: A survey," 2021, *arXiv:2106.00574*.

[20] K. Suh, S. Kim, Y. Ahn, S. Kim, H. Ju, and B. Shim, "Deep reinforcement learning-based network slicing for beyond 5G," *IEEE Access*, vol. 10, pp. 7384–7395, 2022.

[21] Q. Liu, T. Han, N. Zhang, and Y. Wang, "DeepSlicing: Deep reinforcement learning assisted resource allocation for network slicing," in *Proc. 2020 IEEE Glob. Commun. Conf.*, IEEE, 2020, pp. 1–6.

[22] G. Sun, Z. T. Gebrekidan, G. O. Boateng, D. Ayepah-Mensah, and W. Jiang, "Dynamic reservation and deep reinforcement learning based autonomous resource slicing for virtualized radio access networks," *IEEE Access*, vol. 7, pp. 45 758–45 772, 2019.

[23] D. Yan, B. K. Ng, W. Ke, and C.-T. Lam, "Deep reinforcement learning based resource allocation for network slicing with massive MIMO," *IEEE Access*, vol. 11, pp. 75899–75911, 2023.

[24] N. Naderializadeh, J. J. Sydir, M. Simsek, and H. Nikopour, "Resource management in wireless networks via multi-agent deep reinforcement learning," *IEEE Trans. Wireless Commun.*, vol. 20, no. 6, pp. 3507–3523, Jun. 2021.

[25] Y. Azimi, S. Yousefi, H. Kalbkhani, and T. Kunz, "Energy-efficient deep reinforcement learning assisted resource allocation for 5G-RAN slicing," *IEEE Trans. Veh. Technol*, vol. 71, no. 1, pp. 856–871, Jan. 2022.

[26] A. M. Rahimi, A. Ziaeddini, and S. Gonglee, "A novel approach to efficient resource allocation in load-balanced cellular networks using hierarchical DRL," *J. Ambient Intell. Humanized Comput.*, vol. 13, no. 5, pp. 2887–2901, 2022.

[27] 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," *Comput. Netw.*, vol. 220, 2023, Art. no. 109502.

[28] L. Bonati et al., "Colosseum: Large-scale wireless experimentation through hardware-in-the-loop network emulation," in *Proc. IEEE Int. Symp. Dynamic Spectr. Access Netw.*, 2021, pp. 105–113.

[29] M. Tsampazi et al., "A comparative analysis of deep reinforcement learning-based xApps in O-RAN," in *Proc. IEEE Glob. Commun. Conf.*, 2023, pp. 1638–1643.

[30] M. Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[31] Q. Wang et al., "Resource allocation based on radio intelligence controller for open RAN towards 6G," *IEEE Access*, vol. 11, pp. 97909–97919, 2023.

[32] C. Fiandrino, L. Bonati, S. D'Oro, M. Polese, T. Melodia, and J. Widmer, "EXPLORA: AI/ML EXPLainability for the open RAN," *Proc. ACM Netw.*, vol. 1, no. CoNEXT3, pp. 1–26, 2023.

[33] B. Brik, K. Boutiba, and A. Ksentini, "Deep learning for B5G open radio access network: Evolution, survey, case studies, and challenges," *IEEE Open J. Commun. Soc.*, vol. 3, pp. 228–250, 2022.

[34] S. Guadarrama et al., "TF-Agents: A library for Reinforcement Learning in TensorFlow," 2018. Accessed: Jun. 25, 2019. [Online]. Available: https://github.com/tensorflow/agents

[35] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[36] A. Kaloxylos, A. Gavras, D. Camps, M. Ghoraishi, and H. Hrasnica, "AI and ML–enablers for beyond 5G networks," 2021. [Online]. Available: https://5g-ppp.eu/wp-content/uploads/2021/05/AI-MLforNetworks-v1-0.pdf

[37] J. Yang, X. You, G. Wu, M. M. Hassan, A. Almogren, and J. Guna, "Application of reinforcement learning in UAV cluster task scheduling," *Future Gener. Comput. Syst.*, vol. 95, pp. 140–148, 2019.

[38] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv: 1707.06347*.

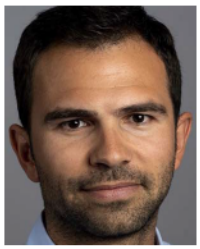
[39] V. Mnih et al., "Playing Atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.

[40] S. Kakade and J. Langford, "Approximately optimal approximate reinforcement learning," in *Proc. 19th Int. Conf. Mach. Learn.*, 2002, pp. 267–274.

[41] 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 *Proc. 10th ACM Int. Workshop Wirel. Netw. Testbeds Exp. Eval. Characterization*, 2016, pp. 25–32.

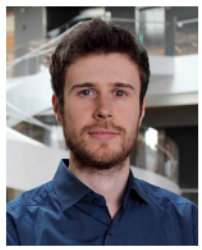
[42] L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "SCOPE: An open and softwarized prototyping platform for NextG systems," in *Proc. 19th Annu. Int. Conf. Mobile Syst. Appl. Serv.*, 2021, pp. 415–426.
[43] U.S. Naval Research Laboratory, "Multi-generator (MGEN) network test tool," 2019. [Online]. Available: https://www.nrl.navy.mil/itd/ncs/products/mgen
[44] O.-RAN Alliance, "O-RAN working group 2: O-RAN AI/ML workflow description and requirements 1.03," O-RAN Working Group 2 Tech. Specification, vol. V01.03, 2021. [Online]. Available: https://www.o-ran.org/blog/o-ran-alliance-introduces-48-new-specifications-released-since-july-2021
[45] M. Kim, J.-S. Kim, M.-S. Choi, and J.-H. Park, "Adaptive discount factor for deep reinforcement learning in continuing tasks with uncertainty," *Sensors*, vol. 22, no. 19, 2022, Art. no. 7266.
[46] C. Zhang et al., "Minibatch recursive least squares Q-learning," *Comput. Intell. Neurosci.*, vol. 2021, 2021, Art. no. 5370281.
[47] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.
[48] M. Hessel et al., "Rainbow: Combining improvements in deep reinforcement learning," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 3215–3222.
[49] M. Fortunato et al., "Noisy networks for exploration," 2017, *arXiv:1706.10295*.
[50] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2015, pp. 1889–1897.
[51] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2019, pp. 2052–2062.
[52] Y. Xiao et al., "NFVdeep: Adaptive online service function chain deployment with deep reinforcement learning," in *Proc. IEEE/ACM Int. Symp. Qual. Serv.*, 2019, pp. 1–10.
[53] R. Kozlica, S. Wegenkittl, and S. Hiränder, "Deep Q-learning versus proximal policy optimization: Performance comparison in a material sorting task," in *Proc. IEEE 32nd Int. Symp. Ind. Electron.*, 2023, pp. 1–6.

Maria Tsampazi (Graduate Student Member, IEEE) received the MEng degree in ECE from National Technical University of Athens, Greece, in 2021. She is currently working toward the PhD degree in electrical engineering with the Institute for the Wireless Internet of Things. Her research lies on NextG networks and intelligent resource allocation in Open RAN. She has received academic awards sponsored by the U.S. National Science Foundation, the IEEE Communications Society, and Northeastern University, and is a 2024 recipient of the National Spectrum Consortium Women in Spectrum Scholarship. She has previously collaborated with both government and industry, including organizations, such as the U.S. Department of Transportation and Dell Technologies.

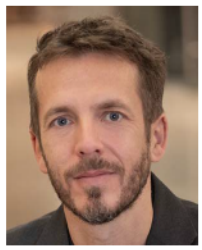


Salvatore D'Oro received the PhD degree from the University of Catania and is an area editor of Elsevier Computer Communications journal. He is a research assistant professor with Northeastern University. He serves on the TPC of IEEE INFOCOM, IEEE CCNC & ICC and IFIP Networking. He is one of the contributors to OpenRAN Gym for AI/ML applications in the Open RAN. His research interests include optimization, AI & network slicing for NextG Open RANs.

Michele Polese (Member, IEEE) received the PhD degree from the University of Padova. He is a research assistant professor with Northeastern University. His research is funded by the US NSF, OUSD, NTIA, and the O-RAN ALLIANCE. He holds several best paper awards and the '22 Mario Gerla Award for Research in Computer Science. He is an associate technical editor for the *IEEE Communications Magazine*, and a guest editor in JSAC's Special Issue on Open RAN.



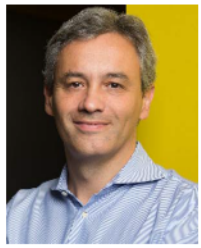
Leonardo Bonati (Member, IEEE) received the PhD degree in computer engineering from Northeastern University in 2022. He is an associate research scientist with the Institute for the Wireless Internet of Things, Northeastern University. His research focuses on softwarized approaches for the Open RAN of next generation of cellular networks, on O-RAN-managed networks, and on network automation and orchestration.



Gwenael Poitau has 20 years of experience developing wireless communication systems for the telecom and defense markets. He has worked for Cavium/Marvell, and has served as head of Engineering with Ultra Electronics. He co-founded the Resilient Machine Learning Institute in collaboration with École de Technologie Supérieure. He is a Wireless AI Technology director with Dell Technologies.

Michael Healy (Member, IEEE) is a technical leader in Dell Technologies. He has 14 issued patents and has contributed to revenue streams, such as the Cisco DOCSIS 3.0, SMPTE, and PCI-SIG. At IBM he led an IBM POWER-ISA CPU Vector Processor silicon project. At AMD, he designed CPU Processors and GPUs for the computing industry. At Cisco, he delivered products to the DOCSIS Cable Telecom industry.

Mohammad Alavirad is a principal technical staff with Dell Technologies. He has worked for Ranovus, Viavi, and Ciena and has served as a guest editor for *IEEE Photonics Technology Letters*, *Quantum Electronics*, and *Applied Physics Letters*. His research lies on Open RAN, and the development of B5G/6G cellular architectures for mmWave Communication systems.



Tommaso Melodia (Fellow, IEEE) is the director of the Institute for the Wireless Internet of Things and research director for the Platforms for Advanced Wireless Research. He has served as associate editor of *IEEE Transactions on Wireless Communications*, *Transactions on Mobile Computing* and *Elsevier Computer Networks*, and as TPC Chair for IEEE INFOCOM, general chair for IEEE SECON, ACM Nanocom, and WUWnet. He is an ACM senior member.