ColO-RAN: Developing Machine Learning-Based xApps for Open RAN Closed-Loop Control on Programmable Experimental Platforms

Michele Polese, *Member, IEEE*, Leonardo Bonati, *Student Member, IEEE*, Salvatore D'Oro, *Member, IEEE*, Stefano Basagni, *Senior Member, IEEE*, and Tommaso Melodia, *Fellow, IEEE*

Abstract—Cellular networks are undergoing a radical transformation toward disaggregated, fully virtualized, and programmable architectures with increasingly heterogeneous devices and applications. In this context, the open architecture standardized by the O-RAN Alliance enables algorithmic and hardware-independent Radio Access Network (RAN) adaptation through closed-loop control. O-RAN introduces Machine Learning (ML)-based network control and automation algorithms as so-called *xApps* running on RAN Intelligent Controllers . However, in spite of the new opportunities brought about by the Open RAN, advances in ML-based network automation have been slow, mainly because of the unavailability of large-scale datasets and experimental testing infrastructure. This slows down the development and widespread adoption of Deep Reinforcement Learning (DRL) agents on real networks, delaying progress in intelligent and autonomous RAN control. In this paper, we address these challenges by discussing insights and practical solutions for the design, training, testing, and experimental evaluation of DRL-based closed-loop control in the Open RAN. To this end, we introduce ColO-RAN, the first publicly-available large-scale O-RAN testing framework with software-defined radios-in-the-loop. Building on the scale and computational capabilities of the Colosseum wireless network emulator, ColO-RAN enables ML research at scale using O-RAN components, programmable base stations, and a "wireless data factory." Specifically, we design and develop three exemplary xApps for DRL-based control of RAN slicing, scheduling and online model training, and evaluate their performance on a cellular network with 7 softwarized base stations and 42 users. Finally, we showcase the portability of ColO-RAN to different platforms by deploying it on Arena, an indoor programmable testbed. The lessons learned from the ColO-RAN implementation and the extensive results from our first-of-its-kind large-scale evaluation highlight the importance of experimental frameworks for the development of end-to-end intelligent RAN control pipelines, from data analysis to the design and testing of DRL agents. They also provide insights on the challenges and benefits of DRL-based adaptive control, and on the trade-offs associated to training on a live RAN. ColO-RAN and the collected large-scale dataset are publicly available to the research community.

Index Terms—O-RAN, network intelligence, 5G/6G, deep reinforcement learning, Colosseum

1 Introduction

In addition to providing traditional voice and data connectivity services, cellular systems are becoming increasingly pervasive in industrial and agricultural automation, interconnecting millions of sensors, vehicles, airplanes, and drones, and providing the nervous system for a plethora of smart systems [1], [2]. These diverse use cases, however, often come with heterogeneous—possibly orthogonal—network constraints and requirements [3]. For instance, autonomous driving applications require Ultra Reliable and Low Latency Communications (URLLC) to allow vehicles to promptly react to sudden events and changing traffic conditions. Instead, high-quality multimedia content requires high data rates, but can tolerate a higher packet loss and latency. Therefore, the future generations of cellular networks need to be *flexible* and *adaptive* to many different application and user requirements.

To achieve these goals, future Radio Access Networks (RANs) will need to combine three key ingredients [4]: (i) *programmable and virtualized protocol stacks* with clearly defined, open interfaces; (ii) *closed-loop network control*; and (iii) *data-driven modeling and Machine Learning (ML)*. Programmability will allow swift adaptation of the RAN to provide bespoke services able to satisfy the requirements of specific deployments. Closed-loop control will leverage telemetry measurements from the RAN to reconfigure cellular nodes, adapting their behavior to current network conditions and traffic. Last, data-driven modeling will exploit recent developments in ML and Big Data to enable real-time, closed-loop, and dynamic decision-making based, for instance, on Deep Reinforcement Learning (DRL) [5]. These are the very same principles at the core of the Open RAN paradigm, which has recently gained traction as a practical enabler of algorithmic and hardware innovation in future cellular networks [6], [7], [8].

To promote the evolution toward open RAN architectures, 3GPP has standardized disaggregated base stations that are

• *The authors are with the Institute for the Wireless Internet of Things, Northeastern University, Boston, MA 02115 USA. E-mail: {m.polese, l.bonati, s.doro, s.basagni, t.melodia}@northeastern.edu.*

Manuscript received 17 December 2021; revised 13 May 2022; accepted 22 June 2022. Date of publication 4 July 2022; date of current version 31 August 2023. This work was supported in part by the U.S. National Science Foundation under Grants CNS-1923789, CNS-1925601, CNS-2120447, and CNS-2112471, and in part by the U.S. Office of Naval Research under Grant N00014-20-1-2132. (Corresponding author: Michele Polese.)
Digital Object Identifier no. 10.1109/TMC.2022.3188013

split into different functional units: Central Unit (CU), Distributed Unit (DU), and Radio Unit (RU). The O-RAN Alliance, an industry consortium, is standardizing open interfaces that connect the various disaggregated functional units to a common control overlay, the RAN Intelligent Controller (RIC), capable of executing custom control logic via so-called *xApps*. Ultimately, these efforts will render the *monolithic RAN "black-box"* obsolete, favoring *open, programmable and virtualized* solutions that expose status and offer control knobs through standardized interfaces [4].

Intelligent, dynamic network optimization via add-on software xApps is clearly a key enabler for future network automation. However, it also introduces novel practical challenges concerning, for instance, the deployment of data-driven ML control solutions at scale. Domain-specific challenges stem from considering the constraints of standardized RANs, the very nature of the wireless ecosystem and the complex interplay among different elements of the networking stack. These challenges, all yet to be addressed in practical RAN deployments, include:

1) Collecting datasets at scale. Datasets for ML training at scale need to be collected and curated to accurately represent the intrinsic randomness and behavior of real-world RANs.

2) Testing ML-based control at scale. Even if ML algorithms are trained on properly collected data, it is necessary to assess their robustness at scale, especially when considering closed-loop control, to prevent poorly-designed data-driven solutions from causing outages or sub-optimal performance.

3) Designing efficient ML agents with unreliable input and constrained output. In production systems, real-time collection of data from the RAN may be inconsistent (e.g., with varying periodicity) or incomplete (e.g., missing entries), and control actions may be constrained by standard specification.

4) Designing ML agents capable of generalizing. Agents should be able to generalize and adapt to unseen deployment configurations not part of the training set.

5) Selecting meaningful features. Features should be accurately selected to provide a meaningful representation of the network status without incurring into dimensionality issues.

Contributions. To address these key challenges, in this paper we describe the design of DRL-based xApps for closed-loop control in O-RAN and their testing with ColO-RAN, a first-of-its-kind softwarized pipeline for large-scale experimental platforms. Based on this experience, we review and discuss key insights in the domain of ML design for O-RAN networks. Our contributions are as follows:

- We introduce ColO-RAN, a first-of-its-kind open, large-scale, experimental O-RAN framework for training and testing ML solutions for next-generation RANs. It combines O-RAN components, a softwarized RAN framework [9], and Colosseum, the world's largest, open, and publicly-available wireless network emulator based on Software-defined Radios (SDRs) [10]. ColO-RAN leverages Colosseum as a *wireless data factory* to generate large-scale datasets for ML training in a variety of Radio Frequency (RF) environments, taking into account propagation and fading characteristics of real-world deployments. The ML models are deployed as xApps on the near-real-time RIC, which connects to RAN nodes through O-RAN-compliant interfaces for data collection and closed-loop control. ColO-RAN is the first platform that enables wireless researchers to deploy ML solutions on a full-stack, fully virtualized O-RAN environment which integrates large-scale data collection and DRL testing capabilities with SDRs. Moreover, the lightweight, containerized implementation of ColO-RAN is easily portable to other experimental platforms. *ColO-RAN and the dataset created for this paper are publicly available to the research community.*[1]
- We develop three xApps for closed-loop control of RAN scheduling and slicing policies, and for the online training of DRL agents on live production environments. We propose an innovative xApp design based on the combination of a unified interface to the near-real-time RIC for data and control messaging, and a data-driven unit with an autoencoder with the DRL agent. This simplifies the design and prototyping of xApps, which share the same interface but are equipped with different intelligent logic. In addition, the combination of the autoencoder and of the DRL agents based on an actor-critic setup with Proximal Policy Optimization (PPO) improves the resilience and robustness to real, imperfect network telemetry, as well as the effectiveness of the policy selection, and the training convergence. We then utilize ColO-RAN to provide insights on the performance of the DRL agents for adaptive RAN control at scale. We train the autoencoders and agents over a 3.4 GB dataset with more than 73 hours of live RAN performance traces, and perform one of the first evaluations of DRL agents autonomously driving a programmable, software-defined RAN with 49 nodes. The results of our large-scale experimental evaluation include new understandings of data analysis, feature selection, and modeling of control actions for DRL agents, and insights on design strategies to train ML algorithms that generalize and operate even with unreliable data.
- We analyze the trade-offs of training of DRL agents on live networks using Colosseum and Arena (a publicly-available indoor testbed for spectrum research [11]) with commercial smartphones. We profile the RAN performance during the DRL exploration phase and after the training, showing how an extra online training step adapts a pre-trained model to deployment-specific parameters, fine-tuning its weights at the cost of a temporary performance degradation in the online exploration phase.
- We discuss and review the lessons learned on multiple levels. First, we consider the *system-level insights* that we gathered while building the ColO-RAN framework and while defining the data and control pipelines that support DRL-based control on a live RAN. Second, we summarize the takeaways on the design of effective *machine-learning-based RAN control*, spanning from the design to the deployment of DRL agents for RAN control. Lessons learned from our work highlight (i) the importance of end-to-end experimental frameworks for the data collection, training, and testing of intelligent RAN control solutions; (ii) the effectiveness of

1. The ColO-RAN source code is available at https://github.com/wineslab/colosseum-near-rt-ric and https://github.com/wineslab/colosseum-scope-e2. The dataset is available at https://github.com/wineslab/colosseum-oran-coloran-dataset.

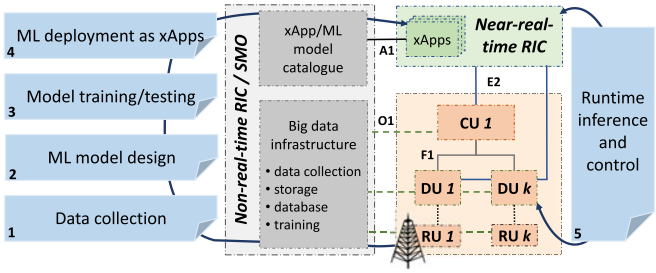


Fig. 1. The O-RAN architecture and the workflow for the design, development and deployment of ML applications in next generation wireless networks.

adaptive control policies over static configurations, even if the latter are optimized; (iii) the impact of different design choices of DRL agents on end-to-end network performance; and (iv) the trade-offs associated to online DRL training in wireless environments.

We believe that these insights and the research infrastructure developed in this work can catalyze, promote and further the deployment of ML-enabled control loops in next generation networks.

The rest of this paper is organized as follows. Section 2 describes the development of ML solutions in O-RAN-based networks. Section 3 introduces ColO-RAN, and Section 4 presents the xApp, DRL agent design, and the data collection campaign for offline training. Large-scale evaluation and lessons learned are discussed in Sections 5 and 6. Section 7 reviews related work. Finally, Section 8 concludes the paper and reviews the main lessons learned.

2 Machine Learning for the Open RAN

The deployment of machine learning models in wireless networks is a multi-step process (Fig. 1). It involves a data collection step, the design of the model, its offline or online training and deployment for runtime inference and control. The O-RAN architecture, also shown in Fig. 1, has been developed to aid the overall deployment process, focusing on open interfaces for data collection and deployment steps. In the following, we describe the O-RAN architecture, and discuss how it facilitates training and deploying ML models in the RAN.

2.1 O-RAN Overview

The O-RAN Alliance, a consortium of academic and industry members, has been pushing forward the concept of an open and programmable cellular ecosystem since its inception in 2018. O-RAN-compliant equipment is based on open and standardized interfaces that enable interoperability of equipment from different vendors and interaction with RAN controllers, which manage the RAN itself. The O-RAN specifications introduce two RICs that perform network control procedures over different time scales, i.e., *near-real-time* and *non-real-time*, respectively [12]. The non-real-time RIC performs operations at time scales larger than 1 s and can involve thousands of devices. Examples include Service Management and Orchestration (SMO), policy management, training and deployment of ML models. The near-real-time RIC, instead, implements tight control loops that span from 10 ms to 1 s, involving hundreds of CUs/DUs. Procedures for load balancing, handover, RAN slicing policies [13] and scheduler configuration are examples of near-real-time RIC operations [14]. The near-real-time RIC can also host third-party applications, i.e., *xApps*. xApps implement control logic through heuristics or data-driven control loops, as well as collect and analyze data from the RAN. At this time, *real-time* loops are left as future work in the O-RAN Alliance specifications [5], [15], [16].

The components of the O-RAN architecture are connected via open and standardized interfaces. The non-real-time RIC uses the O1 interface to collect data in bulk from RAN nodes and to provision services and network functions. The near-real-time RIC connects to CUs and DUs through the E2 interface, which supports different Service Models (SMs), i.e., functionalities like reporting of Key Performance Measurements (KPMs) from RAN nodes and the control of their parameters [17]. The two RICs connect through the A1 interface for the deployment of policies and xApps on the near-real-time RIC.

2.2 ML Pipelines in O-RAN

The O-RAN specifications include guidelines for the management of ML models in cellular networks. Use cases and applications include Quality of Service (QoS) optimization and prediction, traffic steering, handover, and radio fingerprinting [5]. The specifications describe the ML workflow for O-RAN through five steps (Fig. 1): (1) data collection; (2) model design; (3) model training and testing; (4) model deployment as xApp, and (5) runtime inference and control.

First, data is collected for different configurations and setups of the RAN (e.g., large/small scale, different traffic, step 1). Data is generated by the RAN nodes, i.e., CUs, DUs and RUs, and streamed to the non-real-time RIC through the O1 interface, where it is organized in large datasets. After enough data has been collected, a ML model is designed (step 2). This entails the following: (i) identifying the RAN parameters to input to the model (e.g., throughput, latency, etc.); (ii) identifying the RAN parameters to control as output (e.g., RAN slicing and scheduling policies); and (iii) the actual ML algorithm implementation. Once the model has been designed and implemented, it is trained and tested on the collected data (step 3). This involves selecting the model hyperparameters (e.g., the depth and number of layers of the neural network) and training the model on a portion of the collected data until a (satisfactory) level of convergence of the model has been reached. After the model has been trained, it is tested on an unseen portion of the collected data to verify that it is able to generalize and react to potentially unforeseen situations. Then, the model is packaged into an xApp ready to run on the near-real-time RIC (step 4). After the xApp has been created, it is deployed on the O-RAN infrastructure. In this phase, the model is first stored in the xApp catalogue of the non-real-time RIC, and then instantiated on demand on the near-real-time RIC, where it is interfaced with the RAN through the E2 interface to perform runtime inference and control based on the current network conditions (step 5).

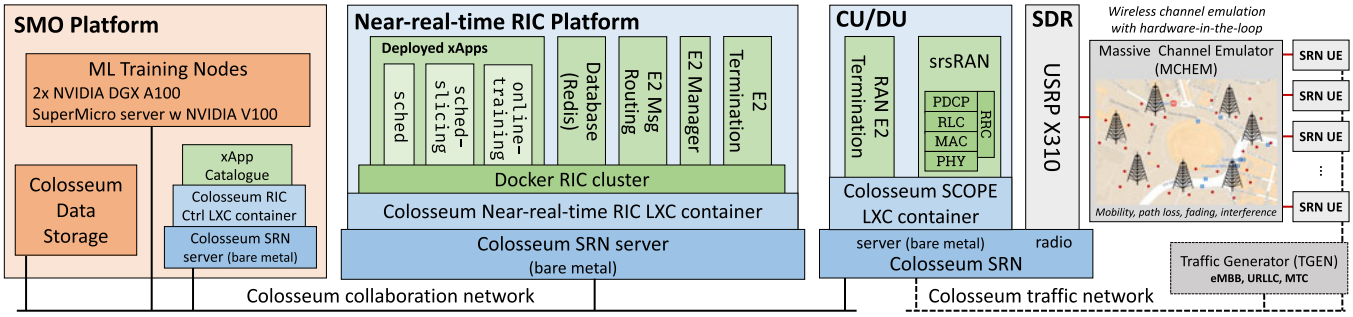


Fig. 2. Integration of the O-RAN infrastructure in Colosseum.

3 ColO-RAN: Enabling Large-Scale ML Research With O-RAN and Colosseum

The ML pipeline described in Section 2.2 involves a number of critical steps whose execution requires joint access to *comprehensive datasets* and *testing facilities at scale*, still largely unavailable to the research community. In fact, even major telecom operators or infrastructure owners might not be able to dedicate (parts of) their extensive commercial networks to training and testing of ML algorithms. This stems from the lack of adequate solutions to separate testing from commercial service and to prevent performance degradation. As a consequence, researchers and innovators are constrained to work with small ad hoc datasets collected in contained lab setups, resulting in solutions that hardly generalize to real-world deployments [18].

To address this limitation, this section introduces ColO-RAN, a large-scale research infrastructure built upon the Colosseum network emulator to train, deploy, and test state-of-the-art wireless ML solutions. We first review the main features of Colosseum and describe its use as a wireless data factory for ColO-RAN (Section 3.1). Then, we introduce the implementation of the ColO-RAN virtualized O-RAN infrastructure on Colosseum (Section 3.2) and of the xApps we designed (Section 4). We finally describe the scenario for data collection that we use to illustrate the usage of ColO-RAN (Section 4.3).

3.1 Colosseum as a Wireless Data Factory

Colosseum is the world's largest wireless network emulator [10]. It was developed by DARPA for the Spectrum Collaboration Challenge and then transitioned to the U.S. National Science Foundation PAWR program to be available for the research community. Colosseum includes 256 USRP X310 SDRs. Half of the SDRs can be controlled by the users, while the other half is part of the Massive Channel Emulator (MCHEM), which uses 64 Virtex-7 FPGAs to emulate wireless channels. MCHEM processes the signals transmitted by radio nodes—called Standard Radio Nodes (SRNs) in Colosseum—through a set of complex-valued finite impulse response filter banks. These filter banks model the propagation characteristics and multi-path scattering of user-defined wireless environments—namely, "scenarios"—as shown in the right part of Fig. 2. Colosseum scenarios can be designed in a variety of different ways. For instance, they can be created from the tap data obtained through field measurement with real radio devices, or from the emulated environment derived from ray-tracing software [10], [19]. In this work, we used scenarios created as part of [9] that feature base stations positioned according to the location of real-world cellular deployments. These locations are derived from the OpenCelliD database [20] by querying it for the cellular deployments in the city of interest, e.g., Rome, Italy in our case. The users are randomly distributed in the surroundings of the base stations. Finally, the channel between any pair of nodes of the network is derived through wireless channel propagation models, and converted into a Colosseum scenario [9], [10]. In this way, MCHEM provides high-fidelity emulation of wireless signals with the same characteristics of those traveling through a real environment. Colosseum also features a user-controlled source Traffic Generator (TGEN), based on MGEN [21], and compute capabilities that make it a full-fledged specialized data center with over 170 high-performance servers.

The combination of programmable software-defined hardware with RF and traffic scenarios uniquely positions Colosseum as a wireless data factory, namely, as a tool that can be used to effectively collect full-stack datasets in heterogeneous and diverse scenarios. With respect to other large testbeds such as the PAWR platforms, Colosseum offers scale and a more controlled and customizable environment that researchers can use to collect data and to test ML algorithms on different RF scenarios and frequencies, without changing the protocol stack or experimental procedures. Compared to a production network, Colosseum is flexible, with programmable radios that can run different software-defined stacks, and the possibility to test closed-loop control without affecting commercial deployments.

3.2 O-RAN-Based Colosseum ML Infrastructure

Besides enabling large-scale data collection, Colosseum also provides a hybrid RF and compute environment for the deployment of ColO-RAN, a complete end-to-end ML infrastructure. ColO-RAN provides researchers with a ready-to-use environment to develop and test ML solutions, following the steps of Fig. 1 (Section 2.2). These include the deployment on a 3GPP-compliant RAN, testing in heterogeneous emulated environments, and an O-RAN-compliant infrastructure. With respect to other open source implementations of the O-RAN infrastructure, ColO-RAN features a more lightweight footprint (e.g., it does not require a full Kubernetes deployment, contrary to the O-RAN Software Community (OSC) RIC), and it can be ported to other testbeds, e.g., Arena [11], with minimal changes, thanks to its virtualized and container-based implementation. As a further contribution, this

platform has been made openly available to the research community.

The software, compute and networking components of our end-to-end infrastructure are shown in Fig. 2. The SMO (left) features three compute nodes to train large ML models, 64 Terabyte of storage for models and datasets, and the xApp catalogue. The near-real-time RIC (Fig. 2, center) provides E2 connectivity to the RAN and support for multiple xApps interacting with the base stations. It is implemented as a standalone Linux Container (LXC) that can be deployed on a Colosseum SRN.[2] It includes multiple Docker containers for the *E2 termination* and *manager*, the *E2 message routing* to handle messages internal to the RIC, a *Redis database*, which keeps a record of the nodes connected to the RIC, and the *xApps* (Section 4). The implementation of the near-real-time RIC is based on the Bronze release of the OSC [22]. The OSC near-real-time RIC was adapted into a minimal version, which does not require a Kubernetes cluster, and can fit in a lightweight LXC container. We also extended the OSC codebase to support concurrent connections from multiple base stations and xApps, and to provide improved support for encoding, decoding and routing of control messages.

The near-real-time RIC connects to the RAN base stations through the E2 interface (Fig. 2, right). The base stations leverage a joint implementation of the 3GPP DUs and CUs. These nodes run the publicly available SCOPE framework [9] co-located with srsRAN [23]. Specifically, SCOPE can tune certain capabilities of the srsRAN base stations at run-time, thus modifying their configuration. Examples of this are the amount of resources, expressed as number of Physical Resource Blocks (PRBs), to allocate to each slice, or the scheduling policy that each base stations should adopt on each slice. Additionally, we extended SCOPE to access relevant network KPMs and forward them to the near-real-time RIC through a custom, standard-compliant E2 termination. The latter extends the capabilities of the E2 termination in [24], making it possible to reconfigure base stations directly from the near-real-time RIC and to perform an automatic and periodic data reporting and collection.[3] The E2 termination allows the setup procedure and registration of the base stations with the near-real-time RIC. Our implementation also features two custom SMs (as discussed next) for trigger-based or periodic reporting, and control events in the base stations. This effectively enables data-driven real-time control loops between the base stations and the xApps. The RAN supports network slicing with 3 slices for different QoS: (i) Enhanced Mobile Broadband (eMBB), representing users requesting video traffic; (ii) Machine-type Communications (MTC) for sensing applications; and (iii) URLLC for latency-constrained applications. Hence, we characterize our slices based on the type of traffic requested by its users, rather than based on a feature of the RAN itself. Slicing is implemented in the SCOPE framework by applying PRB masks during the scheduling process, and it is possible to control the number of PRBs for each slice [9]. As slices represent a specific type of service the operators agree to provide to their subscribers (e.g., as part of Service Level Agreements (SLAs)), they are pre-instantiated on the base stations, and users are statically assigned to one of such slices based on the purchased service level. For each slice, the base stations can adopt 3 different scheduling policies independently of that of the other slices, namely, the Round Robin (RR), the Waterfilling (WF), and the Proportional Fair (PF) scheduling policies. These policies were selected as they represent popular scheduling strategies in wireless deployments [25]. It is worth noticing that we do not directly control the scheduling of the users (i.e., which specific user should be scheduled), for which we would need tighter control loops implemented directly at the base stations [15]. Instead, our control involves the type of scheduling policy run by the base stations for each slice of the network. Finally, the base stations connect to the RF frontends (USRPs X310) that perform signal transmission and reception.

4 xApp Design for DRL-Based Control

The xApps deployed on the near-real-time RIC are the heart of the O-RAN-based RAN control loops. We developed three xApps to evaluate the impact of different ML strategies for closed-loop RAN control (Table 1). Each xApp can receive data and control RAN nodes with two custom SMs, which resemble the O-RAN KPM and RAN control SMs [17]. The control actions available to the xApps are the selection of the slicing policy (the number of PRB allocated to each slice) and of the scheduling policy (which scheduler is used for each slice).

The xApps have been developed by extending the OSC basic xApp framework [26], and include two components (Fig. 3). The first is the interface to the RIC, which implements the SM and performs ASN.1 encoding/decoding of RAN data and control. The second is the ML infrastructure itself, which includes one or more autoencoders and DRL agents. For these, we used TensorFlow 2.4 [27] and the TF-Agents library [28], and we used the neural network architectures described in Section 4.2.

4.1 DRL Agent Design

Agent Architecture. The DRL agents considered in this paper have been trained using the PPO algorithm [29]. PPO is a well-established on-policy DRL architecture that uses an actor-critic configuration where the *actor* network takes actions according to current network state, and the value network (or *critic*) scores the actions taken by the actor network by observing the reward obtained when taking an action in a specific state of the environment. By leveraging this architecture, the PPO algorithm decouples the action taking process from the evaluation of achieved rewards. This is extremely important to ensure that the actor network can learn an unbiased policy (i.e., a mapping between state and actions) where the actor network selects an action because it is effective in the long run and not only because it occasionally results in high instantaneous rewards that are instead inefficient in the majority of cases.

It is worth mentioning that the actor-critic setup is also important because PPO is an on-policy architecture, which means that the training procedure uses a memory buffer that contains data that is collected by using actions that are taken with the most current version of the actor network. If compared to off-policy algorithms (such as Deep Q-Networks

2. https://github.com/wineslab/colosseum-near-rt-ric
3. https://github.com/wineslab/colosseum-scope-e2

TABLE 1
Catalogue of the Three Developed Xapps

xApp	Functionality	Input (Observation)	Output (Action)	ML Models	Utility (Reward)
`sched-slicing`	Single-DRL-agent for joint slicing and scheduling control	Rate, buffer size, PHY TBs (DL)	PRB and scheduling policy for each slice	DRL-base, DRL-reduced-actions, DRL-no-autoencoder	Maximize rate for eMBB, PHY TBs for MTC, minimize buffer size for URLLC
`sched`	Multi-DRL-agent per-slice scheduling policy selection	Rate, buffer size, PRB ratio (DL)	Scheduling policy for each slice	DRL-sched	Maximize rate for eMBB and MTC, PRB ratio for URLLC
`online-training`	Train DRL agents with online exploration	Rate, buffer size, PHY TBs (DL)	Training action (PRB and scheduling)	Trained online by the xApp itself	Based on specific training goals

(DQNs)), which use a memory buffer that store experience collected at any time by the DRL agent, PPO only uses data that is fresh and does not contain experiences from the past, meaning that the memory buffer is emptied every time the actor network is updated during the training phase. This approach is usually slower then others, but together with the actor-critic setup it has been shown to be one of the most efficient and reliable DRL architectures in the literature [29].

Pre-Processing Observations via Autoencoders. One of the main causes of slow training of DRL agents is the use of observations with high dimensionality that result in actor and critic networks with many parameters and large state space. Indeed, the RAN produces an extremely large amount of data which not always provide meaningful insights on the actual state of the system due to redundant information and outliers. To reduce the size of the observation fed to the DRL agent, mitigate outliers and provide a high-quality yet high-level representation of the state of the system, we resort to autoencoders, as also shown in Fig. 3. Specifically, before being fed to the DRL agents, the data produced by the RAN is processed by the encoding portion of an autoencoder for dimensionality reduction (whose impact on DRL-based control is investigated in Section 5.2).

Although autoencoders might have several implementations according to the specific applications, autoencoders for dimensionality reduction have an hourglass architecture with an encoder and a decoder components. The former produces a lower dimension representation of the input data (i.e., latent representation) which - if trained properly - can be accurately reconstructed by the decoder portion of the autoencoder with negligible error. The decoder is the specular image of the encoder and the goal of this architecture is to create a reduced version of the input data that contains only relevant information, yet it is accurate enough to be able to reconstruct the original data without any loss. To further reduce the complexity of the DRL agents, we perform feature selection on the metrics that are observed by the agents (see Section 5 for more details).

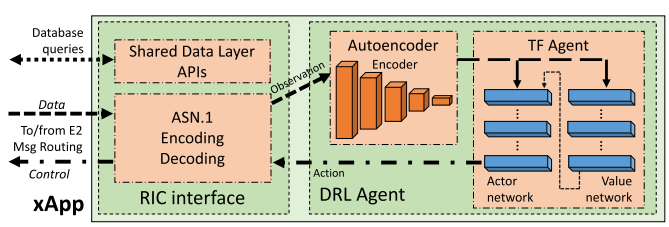


Fig. 3. Structure of a ColO-RAN xApp.

Observations, Actions, and Rewards for Each Agent. As mentioned before, although our xApps share the same high-level architecture shown in Fig. 3, each xApp embeds a different DRL agent whose configuration varies according to the specific goal of the xApp. Specifically, each DRL agent observes different metrics of the RAN, takes diverse actions and aims at maximizing different rewards. The configurations considered in this paper are summarized in Table 1 and discussed in the following:

- **`sched-slicing` xApp:** this xApp is designed to simultaneously select slicing and scheduling policies for a single base station and all slices (eMBB, MTC, and URLLC). For this xApp we trained three DRL models: a baseline model (DRL-base) able to select any feasible actions (i.e., slicing and scheduling policies), an agent that explores a reduced set of actions (DRL-reduced-actions) and an agent where input data is not processed by the autoencoder but is fed directly to the agent (DRL-no-autoencoder). In this case, the reward of the DRL agent is configured to jointly maximize the rate of the eMBB slice, maximize the number of transmitted packets of the MTC slice, and minimize the buffer size (i.e., a proxy for the data transmission latency) of the URLLC slice.
- **`sched` xApp:** this xApp includes three DRL agents that act in parallel and are responsible for selecting the scheduling policy for each individual slice (DRL-slice). Each agent has been trained using slice-specific rewards. Specifically, the eMBB and MTC agents aim at maximizing the data rate of the controlled slice, while the URLLC agents aims at maximizing the ratio between the number of PRBs being requested by each user and how many are effectively allocated by the scheduler (i.e., the higher the ratio, the faster a user is served, and the lower the latency).
- **`online-training` xApp:** this xApp represents a variation of the above two xApps where the embedded DRL agents are fine-tuned in an online fashion by updating the pre-trained weights according to live data from the RAN by performing exploration steps on the online RAN infrastructure itself. While this is not recommended by O-RAN [5], it specializes the trained model to the specific deployment.

It is worth noticing that while agents for the eMBB and MTC slices respectively optimize the user throughput and transmitted packets—which can be measured at the base stations directly—the URLLC slice aims at maintaining the transmission buffer queues of the base stations as low as possible, or transmit as many URLLC packets as possible. These metrics are a proxy for the latency of the service provided to the users in practice, for which we do not have a direct measure at base station protocol stack.

TABLE 2
Configuration Parameters for the Considered Scenario

Parameter	Value
Number of nodes	$N_{\text{BS}} = 7$, $N_{\text{UE}} = 42$
RF parameters	DL carrier $f_d = 0.98$ GHz, UL carrier $f_u = 1.02$ GHz, bandwidth $B = 10$ MHz (50 PRBs)
Schedulers	RR, WF, PF
Slices	eMBB, MTC, URLLC (2 UEs/BS/slice)
Traffic profiles	Slice-based: 4 Mbit/s/UE for eMBB, 44.6 kbit/s/UE for MTC, 89.3 kbit/s/UE URLLC Uniform: 1.5 Mbit/s/UE for eMBB, MTC, URLLC

4.2 Training the DRL Agents

DRL agents are trained on the dataset described in Section 4.3, where at each training episode we select RAN data from different base stations to remove dependence on a specific wireless environment (Section 6) and facilitate generalization.

Following O-RAN specifications, training is performed offline on the dataset. In our case, this is achieved by randomly selecting instances in which the network reaches the state s_1 that results from the combination of the previous state s_0 and the action to explore a_0.

In our experiments, the actor and critic networks of all DRL agents have been implemented as two fully-connected neural networks with 5 layers with 30 neurons each and an hyperbolic tangent activation function. The encoder consists of 4 fully-connected layers with 256, 128, 32 and 3 neurons and a rectified linear activation function. Moreover, the input size of the autoencoder is a matrix of size (10,3), where 3 represents the input KPMs relevant to the specific DRL agent (as specified in Table 1) and 10 represents the number of independent measurements of such KPMs. For all models, the learning rate is set to 0.001.

With respect to the `online-training` xApp, we leverage TensorFlow `CheckPoint` objects to save and restore a pre-trained model for multiple consecutive rounds of training. In this way, the training services in the xApp can restore an agent trained on an offline dataset using it as starting point for the online, live training on the RAN. We discuss the trade-offs involved in this operation in Section 6.

4.3 Large-Scale Data Collection for ColO-RAN

To train the DRL agents for the ColO-RAN xApps we performed large-scale data collection experiments on Colosseum. The parameters for the scenario are summarized in Table 2.

The large-scale RF scenario mimics a real-world cellular deployment in downtown Rome, Italy, with the positions of the base stations derived from the OpenCelliD database [20]. We instantiated a softwarized cellular network with 7 base stations through the SCOPE framework. Each base station operates on a 10 MHz channel (50 PRBs) which can be dynamically assigned to the 3 slices (i.e., eMBB, MTC, URLLC). Additionally, we considered two different TGEN traffic scenarios: slice-based traffic and uniform traffic. In slice-based traffic, users are distributed among different traffic profiles (4 Mbit/s constant bitrate traffic to eMBB users, and 44.6 kbit/s and 89.3 kbit/s Poisson traffic to MTC and URLLC, respectively). The uniform traffic is configured with 1.5 Mbit/s for all users. The training of the DRL agents on the offline dataset has been performed with slice-based traffic. Finally, the base stations serve a total of 42 users equally divided among the 3 slices.

In our data collection campaign, we gathered 3.4 GB of data, for a total of more than 73 hours of experiments. In each experiment, the base stations periodically report RAN KPMs to the non-real-time RIC. These include metrics such as throughput, buffer queues, number of PHY Transport Blocks (TBs) and PRBs. The complete dataset features more than 30 metrics that can be used for RAN analysis and ML training.[4]

5 DRL-Based xApp Evaluation

Learning strategies for RAN control are coded as xApps on ColO-RAN. This section presents their comparative performance evaluation. Feature selection based on RAN KPMs is described in Section 5.1. The experimental comparison of the different DRL models is reported in Section 5.2.

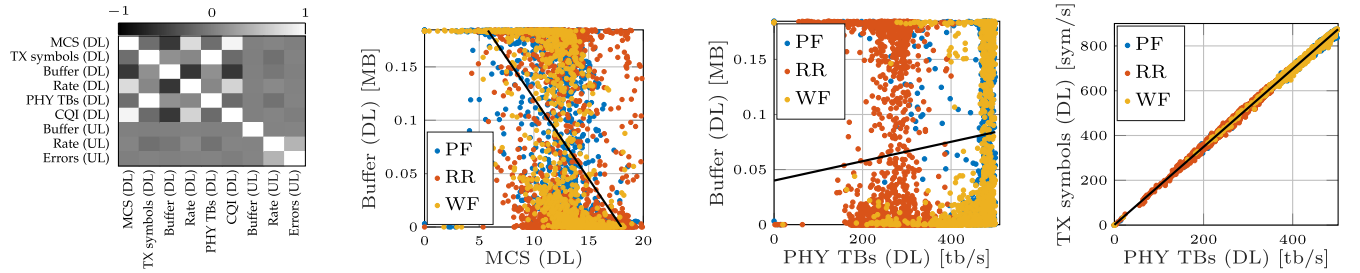
5.1 RAN KPM and Feature Selection

O-RAN is the first architecture to introduce a standardized way to extract telemetry and data from the RAN to drive closed-loop control. However, O-RAN does not indicate which KPMs should be considered for the design of ML algorithms. The O-RAN E2SM KPM specifications [17] allow the generation of more than 400 possible KPMs, listed in [30], [31]. More vendor-specific KPMs may also be reported on E2. These KPMs range from physical layer metrics to base station monitoring statistics. Therefore, the bulk set of data may not be useful to represent the network state for a specific problem. Additionally, reporting or collecting all the metrics via the E2 or O1 interfaces introduces a high overhead, and a highly dimensional input may lead to suboptimal performance for ML-driven xApps [32].

Therefore, a key step in the design process of ML-driven xApps is the selection of the features that should be reported for RAN closed-loop control. In this context, the availability of large-scale, heterogeneous datasets and wireless data factories is key to enable feature selection based on a combined expert- and data-driven approach. To better illustrate this, in Figs. 4 and 5 we report a correlation analysis for several metrics collected in the dataset described in Section 4.3. The correlation analysis helps us identify the KPMs that provide a meaningful description of the network state with minimal redundancy.

Correlation Analysis. Fig. 4a shows the correlation matrix of 9 among the 30 UE-specific metrics in the dataset for the eMBB slice. While downlink and uplink metrics exhibit a low correlation, most downlink KPMs positively or negatively correlate with each other (the same holds for uplink KPMs). For example, the downlink Modulation and Coding Scheme (MCS) and buffer occupancy have a negative correlation (-0.56). This can also be seen in the scatter plot of Fig. 4b: as the MCS increases, it is less likely to have a high buffer occupancy, and vice versa. Similarly, the number of TBs and symbols in downlink have a strong positive correlation (0.998), as

4. The dataset is available at https://github.com/wineslab/colosseum-oran-coloran-dataset.



(a) Correlation matrix. (b) MCS vs. buffer size. (c) Number of PHY TB vs. buffer size. (d) Number of PHY TB vs. TX symbols.

Fig. 4. Correlation analysis for the eMBB slice with 36 PRBs and the slice-based traffic profile. The solid line is the linear regression fit of the data.

also shown in Fig. 4d. Two downlink metrics that do not correlate well, instead, are the number of TBs and the buffer occupancy. Indeed, the amount of data transmitted in each TB varies with the MCS and therefore cannot be used as indicator of how much the buffer will empty after each transmission. Additionally, as shown in Fig. 4c, the three scheduling policies have a different quantitative behavior, but they all show a low correlation.

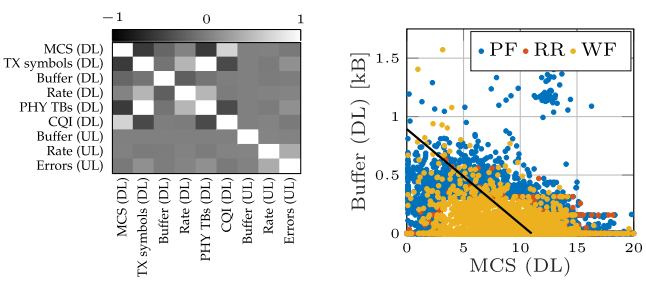
eMBB versus URLLC. The correlation among metrics also depends on the RAN configuration and slice traffic profile. This can be seen by comparing Fig. 4, which analyzes the eMBB slice with 36 PRBs, and Fig. 5, which uses telemetry for the URLLC slice with 11 PRBs. With the slice-based traffic, the URLLC users receive data at a rate that is an order of magnitude smaller than that of the eMBB users. As a consequence, the load on the URLLC slice (represented by the buffer occupancy of Fig. 5b) is lower, and the buffer is quickly drained even with lower MCSs. Consequently, the correlation among the buffer occupancy and the MCS (-0.2) is lower with respect to the eMBB slice. This further makes the case for collecting datasets that are truly representative of a wireless RAN deployment, including heterogeneous traffic and diverse applications.

Summary. Figs. 4 and 5 provide insights on which metrics can be used to describe the RAN status. Since the number of downlink symbols and TBs, or the MCS and the buffer occupancy for the eMBB slice are highly correlated, using them to represent the state of the network only increases the dimensionality of the state without introducing additional information. Conversely, the buffer occupancy and the number of TBs enrich the representation with low redundancy. Therefore, the DRL agents for the xApps in this paper consider as input metrics the number of TBs, the buffer occupancy (or the ratio of PRB granted and requested, which has a high correlation with the buffer status), and the downlink rate.

5.2 Comparing Different DRL-Based RAN Control Strategies

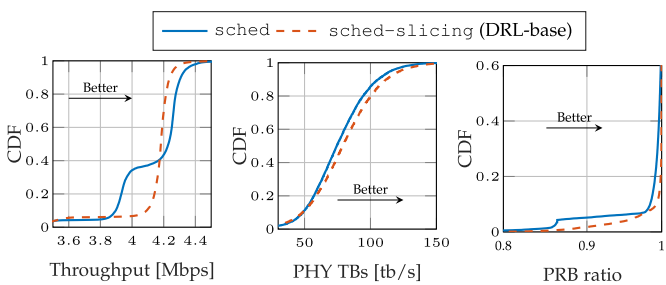
Once the input metrics have been selected, the next step in the design of ML applications involves the selection of the proper modeling strategy [5]. In this paper, we consider ML models for sequential decision making, and thus focus on DRL algorithms.

Control Policy Selection. In this context, it is clearly crucial to properly select the control knobs, i.e., the RAN parameters that need to be controlled and adapted automatically, and the action space, i.e., the support on which these parameters can change. To this end, Fig. 6 compares the performance for the `sched` and `sched-slicing` xApps, which perform different control actions. The first assumes a fixed slicing profile and includes three DRL agents that select the scheduling policy for each slice, while the second jointly controls the slicing (i.e., number of PRBs allocated to each slice) and scheduling policies with a single DRL agent. For this comparison, the slicing profile for the `sched` xApp evaluation matches the configuration that is chosen most often by the `sched-slicing` agent, and the source traffic is slice-based. The Cumulative Distribution Functions (CDFs) of Fig. 6 show that the joint control of slicing and scheduling improves the relevant metric for each slice, with the most significant improvements in the PRB ratio and in the throughput for the users below the 40th percentile. This shows that there exist edge cases in which adapting the slicing profile further improves the



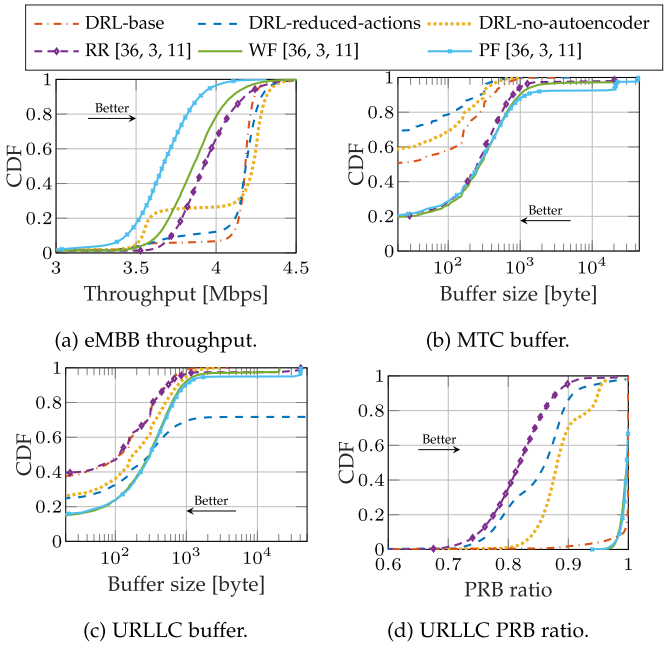
(a) Correlation matrix. (b) MCS vs. buffer size in downlink.

Fig. 5. Correlation analysis for the URLLC slice with 11 PRBs and the slice-based traffic profile. The solid line is the linear regression fit of the data.



(a) eMBB throughput. (b) MTC PHY TBs. (c) URLLC PRB ratio.

Fig. 6. Comparison between the `sched` and `sched-slicing` xApps, with the slice-based traffic profile. The slicing for the `sched` xApp is fixed and based on the configuration chosen with highest probability by the `sched-slicing` xApp (36 PRBs for eMBB, 3 for MTC, 11 for URLLC).



(a) eMBB throughput. (b) MTC buffer.

(c) URLLC buffer. (d) URLLC PRB ratio.

Fig. 7. Comparison between the different models of the `sched-slicing` xApp and baselines without DRL-based adaptation. For the latter, the performance is based on the slicing configuration chosen with highest probability by the best-performing DRL agent, and the three scheduler policies.

network performance with respect to adaptive schedulers with a static slice configuration, even if the fixed slicing configuration is the one that is chosen most often by the `sched-slicing` xApp.

DRL Agent Design. To further elaborate on the capabilities of `sched-slicing`, in Fig. 7 we compare results for different configurations of the DRL agent of the xApp, as well as for a static baseline without slicing or scheduling adaptation, using the slice-based traffic. The slicing profile for the static baseline is the one chosen most often by the `sched-slicing` xApp. The results of Fig. 7 further highlight the performance improvement introduced by adaptive, closed-loop control, with the DRL-driven control outperforming all baselines.

Additionally, this comparison spotlights the importance of careful selection of the action space for the DRL agents. By constraining or expanding the action space that the DRL agents can explore, the xApp designer can bias the selected policies. Consider the DRL-base and DRL-reduced-actions agents (see Table 1), whose difference is in the set of actions that the DRL agent can explore. Notably, the DRL-reduced-actions agent lacks the action that results in the policy chosen most often by the DRL-base agent. Compared to the most common action chosen by the DRL-reduced-actions agent (36 PRB for eMBB, 9 for MTC, 5 for URLLC), the most likely policy of DRL-base agent favors the URLLC over the MTC slice (11 versus 3 PRBs). This is reflected in the performance metrics for the different slices. Notably, DRL-reduced-actions fails to maintain a small buffer and high PRB ratio for the URLLC slice (Figs. 7c and 7d), but achieves the smallest buffer occupancy for the MTC traffic.

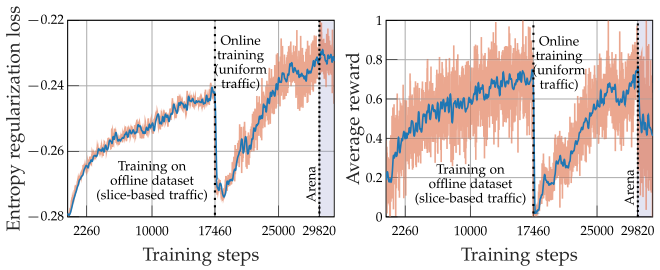
Autoencoder. Finally, the results of Fig. 7 show the benefit of using an autoencoder, as the DRL-base and DRL-reduced-actions agents generally outperform the DRL-no-autoencoder agent. The autoencoder decreases the dimensionality of the input for the DRL agent, improving the mapping between the network state and the actions. Specifically, the autoencoder used in this paper reduces a matrix of $T = 10$ input vectors with $N = 3$ metrics each to a single N-dimensional vector. Second, it improves the performance with online inference on real RAN data. One of the issues of operating ML algorithms on live RAN telemetry is that some entries may be reported inconsistently or may be missing altogether. To address this, we train the autoencoder simulating the presence of a random number of zero entries in the training dataset. This allows the network to be able to meaningfully represent the state even if the input tensor is not fully populated with RAN data.

Control Loop Performance. ColO-RAN is able to perform control loops compliant with the OSC specifications, i.e., between 10ms and 1s. As an example, the average round-trip-time from when the base stations transmit the KPM reports to the RIC to when they receive the control actions computed by the DRL-based xApps equals to 114.27ms, with a variance of 2.653ms.

6 Online Training for DRL-Driven xApps

The last set of results presents an analysis of the trade-offs associated with training DRL agents on a live network in an online fashion. These include the evaluation of the time required for convergence, the impact of the exploration process on the RAN performance, and the benefits involved with this procedure. To do this, we load on the `online-training` xApp a model pre-trained on the offline dataset with the slice-based traffic profile. The same model is used in the DRL-reduced-actions agent. We deploy the `online-training` xApp on a ColO-RAN base station and further continue the training with online exploration, using the uniform traffic profile (with the same constant bitrate traffic for each user). Additionally, we leverage the containerized nature of ColO-RAN to deploy it on Arena [11], a publicly available indoor testbed, and perform training with one SDR base station and three smartphones.

Convergence. Figs. 8 and 9 show how quickly the pre-trained agent adapts to the new environment. Fig. 8a reports the entropy regularization loss as a function of the training step of the agent. This metric correlates with the convergence of the training process: the smaller the absolute value of the entropy, the more likely the agent has converged to a set of actions that maximize the reward in the long run [33]. We stop the training when this metric (and the average reward, Fig. 8b) plateaus, i.e., at step 17460 for the offline training and step 29820 for the online training on Colosseum. The loss remains stable when transitioning from the Colosseum to the Arena online training, while it increases (in absolute value) when switching traffic profile at step 17460. This shows that the agent generalizes better across different channel conditions than source traffic profiles. The same trend is observed for the average reward (Fig. 8b), although the transition from Colosseum to Arena halves the reward (as this configuration features 3 instead of 6 users for each base station). While Colosseum online training requires 30% fewer steps than the initial offline training, it also comes with a higher wall-clock time as offline exploration allows the instantiation of multiple parallel learning environments.



(a) Entropy regularization loss. (b) Reward.

Fig. 8. Metrics for the training on the offline dataset and the online training on Colosseum and Arena. The Arena configuration uses LTE band 7. Notice that the Arena deployment considers three users per base station, contrary to the six users per base station of Colosseum, thus the absolute average reward decreases.

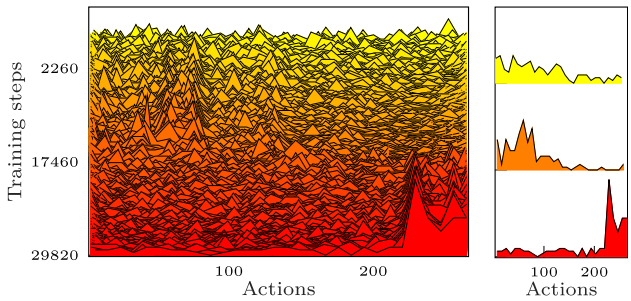


Fig. 9. Distribution of the actions during the training on the offline dataset and the online training on Colosseum. The offline training stops at step 17460.

Because of this, the Colosseum DGX supports the simultaneous exploration of 45 network configurations. Instead, online training can explore one configuration at a time, leading to a higher wall-clock time.

Fig. 9 reports the evolution of the distribution of the actions chosen by the DRL agent for the Colosseum offline and online training. Three histograms for steps 2,260, 17,460 (end of offline training) and 29,820 (end of online training) are also highlighted in the plot on the right. During training, the distribution of the actions evolves from uniform (in yellow) to more skewed, multi-modal distributions at the end of the offline training (in orange) and online training (in red). Additionally, when the training on the new environment begins, the absolute value of the entropy regularization loss increases (Fig. 8a), and, correspondingly, the distribution starts to change, until convergence to a new set of actions is reached again.

Impact of Online Training on RAN Performance. Achieving convergence with a limited number of steps is particularly important for online training, as the performance of the RAN may be negatively affected during the training process. Fig. 10 reports the CDF for the user throughput during training and after, when the agent trained online is deployed on the `sched-slicing` xApp. The performance worsens when comparing the initial training step, which corresponds to the agent still using the actions learned during offline training, with an intermediate step, in which it is exploring random actions. Once the agent identifies the policies that maximize the reward in the new environment (in this case, with the uniform source traffic profile), the throughput improves. The best performance, however, is achieved with the trained agent, which does not perform any exploration. Fig. 11

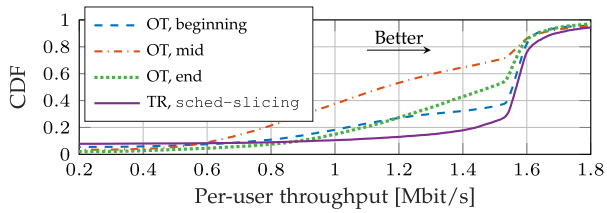


Fig. 10. CDF of the throughput for the eMBB slice during the online training (OT) and with the trained agent (TR) with the uniform traffic profile.

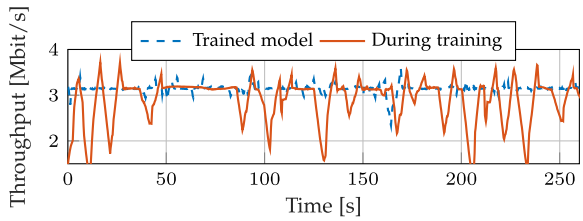


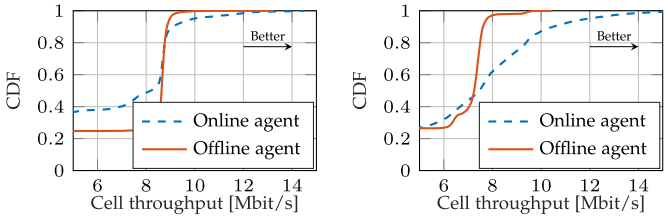
Fig. 11. eMBB slice throughput during training and with the trained model.

further elaborates on this by showing how the online training process increases the throughput variability for the two eMBB users. Therefore, performing online training on a production RAN may be something a telecom operator cannot afford, as it may temporarily lead to disservices or reduced quality of service for the end users. In this sense, testbeds such as Colosseum can be an invaluable tool for two reasons. First, they provide the infrastructure to test pre-trained ML algorithms—and ColO-RAN enables any RAN developer to quickly onboard and test their xApps in a standardized O-RAN platform. Second, they allow online training without affecting the performance of production environments.

Adaptability. The main benefit of an online training phase is to allow the pre-trained agent to adapt to updates in the environment that are not part of the training dataset. In this case, the agent trained by the `online-training` xApp adapts to a new configuration in the slice traffic, i.e., the uniform traffic profile. Fig. 12 compares the cell throughout for the agent before/after the online training, with the slice-based (Fig. 12a) and the uniform traffic (Fig. 12b). Notably, the online agent achieves a throughput comparable with that of the agent trained on the offline dataset with slice-based traffic, showing that—despite the additional training steps—it is still capable of selecting proper actions for this traffic profile. This can also be seen in Fig. 13, which shows that the action selected most often grants the most PRBs to the eMBB slice (whose users have a traffic one order of magnitude higher than MTC and URLLC).

The online agent, however, outperforms the offline-trained agent with the uniform traffic profile, with a gap of 2Mbit/s in the 80th percentile, demonstrating the effectiveness of the online training to adapt to the updated traffic. The action profile also changes when comparing slice-based and uniform traffic, with a preference toward more balanced PRB allocations.

Summary. These results show how online training can help pre-trained models evolve and meet the demands of the specific environment in which they are deployed, at the cost, however, of reduced RAN performance during training. This makes the case for further research in this area, to



(a) Slice-based source traffic. (b) Uniform source traffic.

Fig. 12. Throughput comparison between the offline- and online-trained models with two source traffic patterns. The offline agent is the DRL-base for the `sched-slicing` xApp.

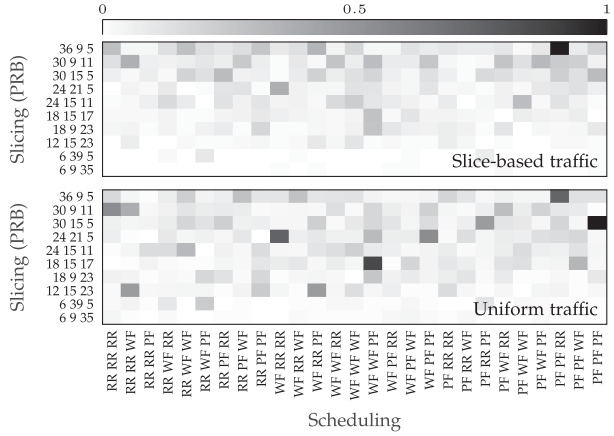


Fig. 13. Probability of selecting a slicing/scheduling combination for the online-trained agent with two different source traffic patterns. For each tuple, the first element refers to the PRB (scheduling) for the eMBB slice, the second for the MTC slice, and the third for the URLLC slice.

develop, for example, smart scheduling algorithms that can alternate training and inference/control steps according to the needs of the network operator. Additionally, we showed that models pre-trained on Colosseum can be effective also in over-the-air deployments, making the case for ColO-RAN as a platform to train and test O-RAN ML solutions in a controlled environment.

7 Related Work

The application of ML to wireless networks has received considerable attention in recent years. Existing works span the full protocol stack, with applications to channel modeling, PHY and MAC layers, ML-based routing and transport, and data-driven applications [34], [35], [36].

Several papers review the potential and challenges of ML for wireless networks, discussing open issues and potential solutions. Kibria *et al.* highlight different areas in which ML and Big Data analytics can be applied to wireless networks [36]. Sun *et al.* [37] and Gunduz *et al.* [38] review the key learning techniques that researchers have applied to wireless, together with open issues. Similarly, Chen *et al.* focus on artificial neural network algorithms [39]. Other reviews can be found in [18], [40]. While these papers present a clear overview of open problems associated with learning in wireless networks, and sometimes include some numerical evaluations [3], [41], they do not provide results based on an actual large-scale deployment, as this paper does, thus missing key insights on using real data, with imperfections, and on using closed-loop control on actual radios.

When it comes to cellular networks, ML has been applied throughout the 3GPP protocol stack. Perenda *et al.* automatically classify modulation and coding schemes [42]. Their approach is robust with respect to modulation parameters that are not part of the training set—a typical problem in wireless networks. Again, at the physical layer, Huang *et al.* investigate learning-based link adaptation schemes for the selection of the proper MCS for eMBB in case of preemptive puncturing for URLLC [43]. Others apply ML to 5 G network management and KPM prediction [44], [45], [46]. These papers, however, do not close the loop through the experimental evaluation of the control action or classification accuracy on real testbeds and networks. Chuai *et al.* describe a large-scale, experimental evaluation on a production network, but the evaluation is limited to a single performance metric [47].

DRL has recently entered the spotlight as a promising enabler of self-adaptive RAN control. Nader *et al.* consider a multi-agent setup for centralized control in wireless networks, but not in the context of cellular networks [48]. Wang *et al.* use DRL to perform handover [49]. Other papers analyze the theoretical performance of DRL agents for medium access [50] and user association [51]. Mollahasani *et al.* evaluate actor-critic learning for scheduling [52], and Zhou *et al.* applies Q-learning to RAN slicing [8]. Chinchali *et al.* apply DRL to user scheduling at the base station level [53]. Differently from these papers, we analyze the performance of DRL agents with a closed loop, implementing the control actions on a software-defined testbed with an O-RAN compliant infrastructure to provide insights on how DRL agents impact a realistic cellular network environment. Finally, [6], [7] consider ML/DRL applications in O-RAN, but provide a limited evaluation of the RAN performance without specific insights and results on using ML.

8 Conclusion and Lessons Learned

The paper presents the first large-scale evaluation of ML-driven O-RAN xApps for managing and controlling a cellular network. To this purpose, we introduce ColO-RAN, the implementation of the O-RAN architecture in the Colosseum network emulator. ColO-RAN features a RAN E2 termination, a near-real-time RIC with three different xApps, and a non-real-time RIC for data storage and ML training.

Lessons Learned. The design of ColO-RAN and of its xApps has allowed us to collect a number of key insights into the development of end-to-end ML and Artificial Intelligence (AI) pipelines for O-RAN and next-generation wireless networks. The lessons learned include (i) *system-level* takeaways, related to the practical implementation of the end-to-end ML pipeline described in Section 2 in multiple large scale experimental testbeds (i.e., Colosseum and Arena); and (ii) insights on the *design of ML algorithms* for wireless network control and on their *online* adaptation. The main lessons learned are summarized as follows:

- The overall framework—combined with the capabilities of Colosseum and Arena—makes it possible to collect datasets, and to test ML routines, at scale. This has required the integration and adaptation of multiple components, including open source software developed by different (and

often siloed) projects, such as the OSC and srsRAN, and custom components such as the xApps developed for ColO-RAN. The implementation of ColO-RAN bridges the gap among multiple projects, providing a streamlined tool for AI/ML-based RAN control.

• End-to-end ML integration requires open interfaces for the data collection and the connectivity between the RIC and the RAN, which need to comply with O-RAN specifications while at the same time adapt to the specific use case (i.e., slicing and scheduling selection). To this end, we equipped ColO-RAN with an O-RAN-compliant E2 interface at the RAN, and developed custom SMs on top, which can be easily extended to study and demonstrate other use cases.

• ColO-RAN strikes a balance between portability and performance. The interfaces and the closed-loop control need to comply with the near-real-time time scale, thus the RAN reporting loop and xApp inference need to be tuned to align to such constraints. At the same time, however, the ColO-RAN RIC has been adapted to be instantiated in a constrained environment (i.e., a single Colosseum SRN or Arena server with LXC containers) and to be an easy-to-deploy solution that can simplify and accelerate research in the O-RAN space.

• The processing and inference based on near-real-time live RAN data requires a careful design of the data ingestion pipelines at the ColO-RAN RIC. Notably, we leveraged autoencoders to aggregate metrics from multiple reporting periods and/or users in a slice, and to handle the lack of data for users which may disconnect or move in the scenario. The autoencoders are trained to handle different zero-padding configurations, thus they accurately represent the RAN state even with missing data. This also makes it possible to provide the DRL agent with a fixed and limited input size, independent on events in the RAN.

• The collection and sharing of data among different components of a networked system, including AI-based xApps, also requires a proper design of the state for each control application. We showed with a large scale wireless dataset that a proper data analysis helps identifying correlation among RAN KPMs and avoid unnecessarily increase the ML algorithm input space. ColO-RAN has been designed to support data collection and analysis across different testbeds and scenarios, thus simplifying the end-to-end ML design process.

• ColO-RAN demonstrates the effectiveness of intelligent control for the RAN through results from a large-scale comparative performance evaluation of multiple xApps. Adaptive policies effectively improve the performance of tenants with different (and often orthogonal) traffic requirements, a key element for next-generation cellular networks.

• Finally, ColO-RAN makes it possible to train and test the same xApps on different wireless environments, e.g., Colosseum and Arena. This allowed us to evaluate online training steps, which make it possible to adapt and fine-tune the performance of a pre-trained ML algorithm to new events and conditions, not part of the training set. We showed however that this has a cost in terms of degraded RAN performance during the exploration phase, prompting the study and development of smart scheduling solutions. In addition, the performance evaluation indicated that models pre-trained on Colosseum data are effective also when deployed on an over-the-air testbed, making hardware-based emulation a viable and powerful step toward the creation of large-scale wireless experimental datasets.

ColO-RAN and the dataset collected for this work are publicly available and will enable O-RAN-based experiments in Colosseum. We believe that this end-to-end experimental infrastructure, together with the insights and lessons summarized here, will enable further research and development in AI and ML solutions for the Open RAN.

REFERENCES

[1] Ericsson, "Ericsson mobility report," 2021. [Online]. Available: https://www.ericsson.com/en/mobility-report

[2] M. Giordani, M. Polese, M. Mezzavilla, S. Rangan, and M. Zorzi, "Toward 6G networks: Use cases and technologies," *IEEE Commun. Mag.*, vol. 58, no. 3, pp. 55–61, Mar. 2020.

[3] Z. Xiong, Y. Zhang, D. Niyato, R. Deng, P. Wang, and L. -C. Wang, "Deep reinforcement learning for mobile 5G and beyond: Fundamentals, applications, and challenges," *IEEE Veh. Technol. Mag.*, vol. 14, no. 2, pp. 44–52, Jun. 2019.

[4] L. Bonati, M. Polese, S. D'Oro, S. Basagni, and T. Melodia, "Open, programmable, and virtualized 5G networks: State-of-the-art and the road ahead," *Comput. Netw.*, vol. 182, pp. 1–28, Dec. 2020.

[5] O-RAN Working Group 2, "O-RAN AI/ML workflow description and requirements–v1.01," O-RAN.WG2.AIML-v01.01 Technical Specification, Apr. 2020.

[6] S. Niknam *et al.*, "Intelligent O-RAN for beyond 5G and 6G wireless networks," May 2020, *arXiv:2005.08374*.

[7] L. Bonati, S. D'Oro, M. Polese, S. Basagni, and T. Melodia, "Intelligence and learning in O-RAN for data-driven NextG cellular networks," *IEEE Commun. Mag.*, vol. 59, no. 10, pp. 21–27, Oct. 2021.

[8] H. Zhou, M. Elsayed, and M. Erol-Kantarci, "RAN resource slicing in 5G using multi-agent correlated Q-learning," in *Proc. IEEE Int. Symp. Pers., Indoor Mobile Radio Commun.*, 2021, pp. 1179–1184.

[9] L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "SCOPE: An open and softwarized prototyping platform for NextG systems," in *Proc. ACM Int. Conf. Mobile Syst. Appl. Serv.*, 2021, pp. 415–426.

[10] 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.

[11] L. Bertizzolo *et al.*, "Arena: A 64-antenna SDR-based ceiling grid testing platform for sub-6 GHz 5G-and-beyond radio spectrum research," *Comput. Netw.*, vol. 181, pp. 1–17, Nov. 2020.

[12] O-RAN Working Group 1, "O-RAN Architecture Description - v2.00," O-RAN.WG1.O-RAN-Architecture-Description-v02.00 Technical Specification, Jul. 2020.

[13] S. D'Oro, L. Bonati, F. Restuccia, and T. Melodia, "Coordinated 5G network slicing: How constructive interference can boost network throughput," *IEEE/ACM Trans. Netw.*, vol. 29, no. 4, pp. 1881–1894, Aug. 2021.

[14] O-RAN Alliance White Paper, "O-RAN use cases and deployment scenarios," Feb. 2020. [Online]. Available: https://tinyurl.com/8cmtxmyp

[15] S. D'Oro, M. Polese, L. Bonati, H. Cheng, and T. Melodia, "dApps: Distributed applications for real-time inference and control in O-RAN," 2022, *arXiv:2203.02370*.

[16] A. S. Abdalla, P. S. Upadhyaya, V. K. Shah, and V. Marojevic, "Toward next generation open radio access network–what o-ran can and cannot do!," 2022, *arXiv:2111.13754*.

[17] O-RAN Working Group 3, "O-RAN near-real-time RAN intelligent controller E2 service model (E2SM) KPM 1.0," ORAN-WG3.E2SM-KPM-v01.00.00 Technical Specification, Feb. 2020.

[18] J. Wang, C. Jiang, H. Zhang, Y. Ren, K. -C. Chen, and L. Hanzo, "Thirty years of machine learning: The road to pareto-optimal wireless networks," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 1472–1514, Jul.–Sep. 2020.

[19] M. Tehrani-Moayyed, L. Bonati, P. Johari, T. Melodia, and S. Basagni, "Creating RF scenarios for large-scale, real-time wireless channel emulators," in *Proc. 19th Mediterranean Commun. Comput. Netw. Conf.*, 2021, pp. 1–8.

[20] Unwired Labs. OpenCelliD, 2022. Accessed: Apr. 2022. [Online]. Available: https://opencellid.org

[21] U. S. Naval Research Laboratory, "MGEN traffic emulator." Accessed: 2022. [Online]. Available: https://tinyurl.com/beexe8yc
[22] O-RAN software community. Bronze release, 2020. Accessed: Jul. 2021. [Online]. Available: https://wiki.o-ran-sc.org/pages/viewpage.action?pageId=14221635
[23] I. Gomez-Miguelez *et al.*, "srsLTE: An open-source platform for LTE evolution and experimentation," in *Proc. 10th ACM Int. Workshop Wireless Netw. Testbeds, Exp. Eval. Characterization*, 2016, pp. 25–32.
[24] O-RAN software community. O-DU-L2 documentation, 2021. Accessed: Jul. 2021. [Online]. Available: https://docs.o-ran-sc.org/projects/o-ran-sc-o-du-l2/en/latest/index.html
[25] F. Capozzi, G. Piro, L. Grieco, G. Boggia, and P. Camarda, "Downlink packet scheduling in LTE cellular networks: Key design issues and a survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 2, pp. 678–700, Apr.–Jun. 2013.
[26] O-RAN software community. xApp framework, 2020. Accessed: Jul. 2021. [Online]. Available: https://wiki.o-ran-sc.org/display/ORANSDK/xAppFramework
[27] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: https://www.tensorflow.org/
[28] 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
[29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," Jul. 2017, *arXiv:1707.06347*.
[30] 3GPP, "5G performance measurements," Technical Specification (TS) 28.552, Jun. 2021, version 17.3.1.
[31] 3GPP, "Performance measurements Evolved Universal Terrestrial Radio Access Network (E-UTRAN)" Technical Specification (TS) 32.425, Jun. 2021, version 17.1.0.
[32] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proc. 2nd Workshop Mach. Learn. Sensory Data Anal.*, 2014, pp. 4–11.
[33] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 1352–1361.
[34] T. J. O'Shea, K. Karra, and T. C. Clancy, "Learning to communicate: Channel auto-encoders, domain specific regularizers, and attention," in *Proc. IEEE Int. Symp. Signal Process. Inf. Technol.*, 2016, pp. 223–228.
[35] S. Abbasloo, C.-Y. Yen, and H. J. Chao, "Wanna make your TCP scheme great for cellular networks? Let machines do it for you!," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 1, pp. 265–279, Jan. 2021.
[36] M. G. Kibria, K. Nguyen, G. P. Villardi, O. Zhao, K. Ishizu, and F. Kojima, "Big data analytics, machine learning, and artificial intelligence in next-generation wireless networks," *IEEE Access*, vol. 6, pp. 32 328–32 338, 2018.
[37] Y. Sun, M. Peng, Y. Zhou, Y. Huang, and S. Mao, "Application of machine learning in wireless networks: Key techniques and open issues," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3072–3108, Oct.–Dec. 2019.
[38] D. Gunduz, P. de Kerret, N. D. Sidiropoulos, D. Gesbert, C. R. Murthy, and M. van der Schaar, "Machine learning in the air," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 10, pp. 2184–2199, Oct. 2019.
[39] M. Chen, U. Challita, W. Saad, C. Yin, and M. Debbah, "Artificial neural networks-based machine learning for wireless networks: A tutorial," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3039–3071, Oct.–Dec. 2019.
[40] C. Jiang, H. Zhang, Y. Ren, Z. Han, K. -C. Chen, and L. Hanzo, "Machine learning paradigms for next-generation wireless networks," *IEEE Wireless Commun.*, vol. 24, no. 2, pp. 98–105, Apr. 2017.
[41] Y. Fu, S. Wang, C.-X. Wang, X. Hong, and S. McLaughlin, "Artificial intelligence to manage network traffic of 5G wireless networks," *IEEE Netw.*, vol. 32, no. 6, pp. 58–64, Nov./Dec. 2018.
[42] E. Perenda, S. Rajendran, G. Bovet, S. Pollin, and M. Zheleva, "Learning the unknown: Improving modulation classification performance in unseen scenarios," in *Proc. IEEE Conf. Comput. Commun.*, 2021, pp. 1–10.
[43] Y. Huang, T. Hou, and W. Lou, "A deep-learning-based link adaptation design for eMBB/URLLC multiplexing in 5G NR," in *Proc. IEEE Conf. Comput. Commun.*, 2021, pp. 1–10.
[44] M. Polese, R. Jana, V. Kounev, K. Zhang, S. Deb, and M. Zorzi, "Machine learning at the edge: A data-driven architecture with applications to 5G cellular networks," *IEEE Trans. Mobile Comput.*, vol. 20, no. 12, pp. 3367–3382, Dec. 2021.
[45] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "DeepCog: Cognitive network management in sliced 5G networks with deep learning," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 280–288.
[46] J. Wang *et al.*, "Spatiotemporal modeling and prediction in cellular networks: A Big Data enabled deep learning approach," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.
[47] J. Chuai *et al.*, "A collaborative learning based approach for parameter configuration of cellular networks," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1396–1404.
[48] 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.
[49] Z. Wang, L. Li, Y. Xu, H. Tian, and S. Cui, "Handover control in wireless systems via asynchronous multiuser deep reinforcement learning," *IEEE Internet Things J.*, vol. 5, no. 6, pp. 4296–4307, Dec. 2018.
[50] S. Wang, H. Liu, P. H. Gomes, and B. Krishnamachari, "Deep reinforcement learning for dynamic multichannel access in wireless networks," *IEEE Trans. Cogn. Commun. Netw.*, vol. 4, no. 2, pp. 257–265, Jun. 2018.
[51] N. Zhao, Y. -C. Liang, D. Niyato, Y. Pei, M. Wu, and Y. Jiang, "Deep reinforcement learning for user association and resource allocation in heterogeneous cellular networks," *IEEE Trans. Wireless Commun.*, vol. 18, no. 11, pp. 5141–5152, Nov. 2019.
[52] S. Mollahasani, M. Erol-Kantarci, M. Hirab, H. Dehghan, and R. Wilson, "Actor-critic learning based QoS-aware scheduler for reconfigurable wireless networks," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 1, pp. 45–54, Jan./Feb. 2022.
[53] S. Chinchali *et al.*, "Cellular network traffic scheduling with deep reinforcement learning," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 766–774.

Michele Polese (Member, IEEE) received the PhD degree from the Department of Information Engineering, University of Padova, in 2020. He is currently a principal research scientist with the Institute for the Wireless Internet of Things, Northeastern University, Boston, since March 2020. He also was an adjunct professor and postdoctoral researcher in 2019/2020 with the University of Padova, and a part-time lecturer in Fall 2020 and 2021 with Northeastern University. During his PhD, he visited New York University (NYU), AT&T Labs in Bedminster, New Jersey, and Northeastern University. His research interests are in the analysis and development of protocols and architectures for future generations of cellular networks (5G and beyond), in particular for millimeter-wave and terahertz networks, spectrum sharing and passive/active user coexistence, open RAN development, and the performance evaluation of end-to-end, complex networks. He has contributed to O-RAN technical specifications and submitted responses to multiple FCC and NTIA notice of inquiry and requests for comments, and is a member of the Committee on Radio Frequency Allocations of the American Meteorological Society (2022-2024). He collaborates and has collaborated with several academic and industrial research partners, including AT&T, Mavenir, NVIDIA, InterDigital, NYU, University of Aalborg, King's College, and NIST. He was awarded with several best paper awards, is serving as TPC co-chair for WNS3 2021-2022, as an associate technical editor for the *IEEE Communications Magazine*, and has organized the Open 5G Forum in Fall 2021.

Leonardo Bonati (Student Member, IEEE) received the BS degree in information engineering and the MS degree in telecommunication engineering from University of Padova, Italy, in 2014 and 2016, respectively. He is currently working toward the PhD degree in computer engineering with Northeastern University, Boston, Massachusetts. His research interests include 5G and beyond cellular networks, network slicing, and software-defined networking for wireless networks.

Salvatore D'Oro (Member, IEEE) received the PhD degree from the University of Catania, in 2015. He is currently a research assistant professor with Northeastern University. He is an area editor of *Elsevier Computer Communications Journal* and serves on the Technical Program Committee (TPC) of multiple conferences and workshops such as IEEE INFOCOM, IEEE CCNC, IEEE ICC and IFIP Networking. His research interests include optimization, artificial intelligence, security, network slicing and their applications to 5G networks and beyond.

Stefano Basagni (Senior Member, IEEE) received the PhD degree in electrical engineering from the University of Texas at Dallas, in 2001, and the PhD degree in computer science from the University of Milano, Italy, in 1998. He is currently with the Institute for the Wireless Internet of Things and a professor with the ECE Department, Northeastern University, Boston, Massachusetts. His current interests concern research and implementation aspects of mobile networks and wireless communications systems, wireless sensor networking for IoT (underwater, aerial and terrestrial), and definition and performance evaluation of network protocols. He has published more than ten dozen of highly cited, refereed technical papers and book chapters. His h-index is currently 46 (May 2022). He is also co-editor of three books. He served as a guest editor of multiple international ACM/IEEE, Wiley and Elsevier journals. He has been the TPC co-chair of international conferences. He is a distinguished scientist of the ACM, and a member of the Council for Undergraduate Education (CUR).

Tommaso Melodia (Fellow, IEEE) received the PhD degree in electrical and computer engineering from the Georgia Institute of Technology, in 2007. He is currently the William Lincoln Smith chair professor with the Department of Electrical and Computer Engineering, Northeastern University in Boston. He is also the founding director of the Institute for the Wireless Internet of Things and the director of research for the PAWR Project Office. He is a recipient of the National Science Foundation CAREER Award. He has served as associate editor of *IEEE Transactions on Wireless Communications*, *IEEE Transactions on Mobile Computing*, *Elsevier Computer Networks*, among others. He has served as Technical Program Committee chair for IEEE Infocom 2018, general chair for IEEE SECON 2019, ACM Nanocom 2019, and ACM WUWnet 2014. He is the director of research for the Platforms for Advanced Wireless Research (PAWR) Project Office, a $100 M public-private partnership to establish four city-scale platforms for wireless research to advance the US wireless ecosystem in years to come. His research on modeling, optimization, and experimental evaluation of Internet-of-Things and wireless networked systems has been funded by the National Science Foundation, the Air Force Research Laboratory the Office of Naval Research, DARPA, and the Army Research Laboratory. He is a senior member of the ACM.