EXPLORA: AI/ML *EXPL*ainability for the *O*pen *RA*N

CLAUDIO FIANDRINO, IMDEA Networks Institute, Spain
LEONARDO BONATI, Institute for the Wireless Internet of Things, Northeastern University, USA
SALVATORE D'ORO, Institute for the Wireless Internet of Things, Northeastern University, USA
MICHELE POLESE, Institute for the Wireless Internet of Things, Northeastern University, USA
TOMMASO MELODIA, Institute for the Wireless Internet of Things, Northeastern University, USA
JOERG WIDMER, IMDEA Networks Institute, Spain

The Open Radio Access Network (RAN) paradigm is transforming cellular networks into a system of disaggregated, virtualized, and software-based components. These self-optimize the network through programmable, closed-loop control, leveraging Artificial Intelligence (AI) and Machine Learning (ML) routines. In this context, Deep Reinforcement Learning (DRL) has shown great potential in addressing complex resource allocation problems. However, DRL-based solutions are inherently hard to explain, which hinders their deployment and use in practice. In this paper, we propose *EXPLORA*, a framework that provides explainability of DRL-based control solutions for the Open RAN ecosystem. *EXPLORA* synthesizes network-oriented explanations based on an attributed graph that produces a link between the actions taken by a DRL agent (*i.e.*, the nodes of the graph) and the input state space (*i.e.*, the attributes of each node). This novel approach allows *EXPLORA* to explain models by providing information on the wireless context in which the DRL agent operates. *EXPLORA* is also designed to be lightweight for real-time operation. We prototype *EXPLORA* and test it experimentally on an O-RAN-compliant near-real-time RIC deployed on the Colosseum wireless network emulator. We evaluate *EXPLORA* for agents trained for different purposes and showcase how it generates clear network-oriented explanations. We also show how explanations can be used to perform informative and targeted intent-based action steering and achieve median transmission bitrate improvements of 4% and tail improvements of 10%.

Additional Key Words and Phrases: 5G, 6G, Mobile networks, O-RAN, Explainable AI

ACM Reference Format:
Claudio Fiandrino, Leonardo Bonati, Salvatore D'Oro, Michele Polese, Tommaso Melodia, and Joerg Widmer. 2023. *EXPLORA*: AI/ML *EXPL*ainability for the *O*pen *RA*N. 1, 1 (November 2023), 26 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

To support this rapidly changing and complex environment foreseen in the sixth generation (6G), the industry is now transitioning toward Radio Access Network (RAN) architectures based upon softwarization, virtualization, and network programmability paradigms, such as the Open RAN [61]. Specifying how to practically realize an Open RAN architecture is one of the goals of the O-RAN Alliance. O-RAN leverages the above principles to provide an alternative to existing

Authors' addresses: Claudio Fiandrino, claudio.fiandrino@imdea.org, IMDEA Networks Institute, Madrid, Spain; Leonardo Bonati, l.bonati@northeastern.edu, Institute for the Wireless Internet of Things, Northeastern University, Boston, USA; Salvatore D'Oro, s.doro@northeastern.edu, Institute for the Wireless Internet of Things, Northeastern University, Boston, USA; Michele Polese, m.polese@northeastern.edu, Institute for the Wireless Internet of Things, Northeastern University, Boston, USA; Tommaso Melodia, melodia@northeastern.edu, Institute for the Wireless Internet of Things, Northeastern University, Boston, USA; Joerg Widmer, joerg.widmer@imdea.org, IMDEA Networks Institute, Madrid, Spain.

inflexible, monolithic equipment with systems based on disaggregated, virtualized, and software-based components that interact via open and standardized interfaces. O-RAN also introduces two RAN Intelligent Controllers (RICs) that act as abstraction layers to monitor, control and manage RAN components at different timescales, namely at near-real-time (or near-RT) and non-real-time (non-RT) timescales. The near-RT and non-RT RICs host custom applications, respectively xApps and rApps, that run Artificial Intelligence (AI)-based closed-loop control routines to optimize the RAN operations and adapt them to current traffic demand and network conditions [61].

Bootstrapped by such initiatives, the application of AI to the Open RAN has become an emerging area of interest [61], with contributions that encompass spectrum management [3, 68], mobility management [13], and resource allocation [17, 32, 60], as well as custom control loops to jointly optimize location and transmission directionality of Unmanned Aerial Vehicles (UAVs) [5]. Among the existing data-driven techniques, Deep Reinforcement Learning (DRL) appears to be particularly suitable to control Open RAN systems [42]. Unlike supervised learning models— tailored to classification or regression tasks—Reinforcement Learning (RL) and DRL focus on decision-making processes where decisions are made through a trial-and-error process to maximize a certain utility function (*e.g.*, the throughput of the network). DRL is widely used for several networking problems related to resource allocation, handover and load balancing [27, 43, 47, 74], among others. Because of its capability of interacting with, and adapting to, complex, highly distributed, dynamic and uncertain environments—such as those typical of cellular RANs—is a compelling candidate AI technique for Open RAN. Successful industry applications of DRL in the RAN cover relevant use cases such as traffic steering (Mavenir [51]), and handover management (Intel [29]), among others.
Motivations and Objectives. Since DRL agents leverage deep neural networks, the logic governing their decisions is frequently hard to understand. This is unlike, for example, Decision Trees (DTs) [45], whose structure and decision-making logic is generally explicit and easy to understand, especially in relatively simple and confined practical applications such as automated Base Station (BS) reconfiguration [49]. Despite their effectiveness, the lack of explainability of DRL models makes them difficult to use in production networks because of the inherent lack of understanding of the logic behind decisions. This complicates troubleshooting and predicting decisions, and makes DRL more vulnerable to adversarial attacks [66]. The issue affects AI at large, not only DRL, and makes understanding why models take certain actions more difficult, especially with complex environments and large action spaces.

To address these problems, the research community is trying to shed light on the inner mechanisms of such models to make them more explainable and interpretable. For instance, Puiutta et al. [63] coined the term EXplainable Reinforcement Learning (XRL) and illustrate several explainability techniques that are specific to the learning paradigm. However, although the interest in promoting trust and interpretability to AI has recently gained momentum [72], explainable AI in the context of mobile networks is still at its early stages and largely unexplored [16, 53].
Our Contribution. In this paper, we try to fill this gap and make DRL for Open RAN applications more robust, resilient and—more importantly— explainable. Specifically, we develop *EXPLORA*, a lightweight O-RAN-compliant framework designed to explain the logic of DRL agents executed within xApps or rApps performing near- and non-real-time closed-loop resource allocation and control. In contrast with the traditional approach, where model explanations only provide intuitions that reveal *how inner mechanisms of a model work* (*e.g.*, neuron activation), we ensure that *EXPLORA* also provides informative explanations on the wireless network behavior to help operators in interpreting AI decisions [53]. Specifically, we advance EXplainable Artificial Intelligence (XAI) and XRL research in several ways.

We show that *EXPLORA* addresses complex challenges (§ 3.3) that prevent the direct use of state-of-the-art XAI tools (§ 3.2). We base the XAI component of *EXPLORA* on attributed graphs to connect the actions taken by the agents (nodes) to the effect on the environment (attributes of the nodes) and to distill knowledge by analyzing the transitions between actions (edges - § 4). This allows *EXPLORA* to explain models by highlighting the circumstances under which a DRL agent takes specific actions. Overall, such combined knowledge (*i.e.*, input-output relations and conditions that trigger certain actions) is useful to domain experts and mobile operators willing to retain full control of the AI-based system and, if needed, to consciously override or inhibit AI decisions on the basis of previously identified intents to be fulfilled.

We experimentally demonstrate the explainability capabilities of *EXPLORA* on Colosseum, the world's largest O-RAN wireless network emulator [52]. Specifically, we apply *EXPLORA* to xApps embedding DRL agents for control of RAN slicing and scheduling, developed via the OpenRAN Gym open-source O-RAN data collection and AI testing toolbox [6] (§ 5.1). We show not only that *EXPLORA* is capable of synthesizing explanations that facilitate monitoring and troubleshooting (§ 6.2), but also that it helps to improve RAN performance by using explanations to proactively identify and substitute actions that could lead to poor performance (§ 6.3).

Key Contributions and Findings. Our far-reaching goal is to contribute toward promoting AI trustworthiness in Open RAN. The key contributions (marked with "C") and findings ("F") of our study are summarized as follows:

C1. We propose *EXPLORA*, a new framework for network-oriented explanations. *EXPLORA* provides informative post-hoc explanations and can evaluate the effectiveness of control actions taken by the DRL agents;

C2. We implement *EXPLORA* as an xApp on an O-RAN-compliant near-RT RIC; and

C3. We release the artifacts of our study on https://github.com/wineslab/explora.

F1. We find that *EXPLORA* provides effective and concise explanations fully characterizing DRL agents behavior.

F2. We find that *EXPLORA* enables the creation of ad-hoc policies for intent-based action steering that ultimately improve users' Key Performance Indicators (KPIs). We observe median transmission bitrate improvements of 4% and tail improvements of 10%.

2 BACKGROUND

In this section, we provide background knowledge on the different technologies considered in our paper: O-RAN (§ 2.1), DRL (§ 2.2), and finally XAI and XRL (§ 2.3).

2.1 Background on O-RAN

The O-RAN specifications introduce a complete architectural model for the Open RAN (see Figure 1). The interactions and interoperability between multi-vendor equipment implementing disaggregated RAN next Generation Node Bs (gNBs) (*i.e.*, central, distributed and remote units—O-CU, O-DU and O-RU) is possible via open and standardized interfaces. Management and control is provided by a set of RAN Intelligent Controllers (RICs) that operate at different timescales, *i.e.*, non-RT (timespan larger than 1 s) and near-RT (timespan between 10 ms to 1 s) [61]. The non-RT RIC enforces policies controlling thousands of devices, including data collection and training phase of the AI/Machine Learning (ML) workflows at large and provides the near-RT RIC with policy-based guidance through the A1 interface. It is embedded in the network Service Management and Orchestrator (SMO), which performs automated monitoring and provisioning of network functions through the O1 interface. The near-RT RIC operates control loops for policy enforcement (*i.e.*, control) at a smaller scale (tens to hundreds of nodes) and governs radio resource management operations such as resource allocation [17, 61] by interacting with the RAN nodes through the E2 interface. The RICs can host third-party applications, *i.e.*, rApps at non-RT scale and xApps at

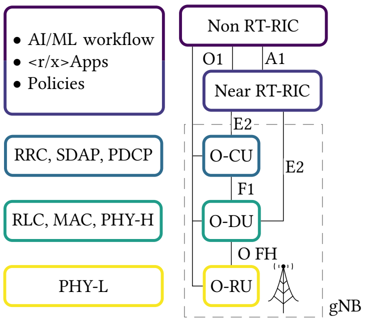


Fig. 1. The O-RAN reference architecture

near-RT scale. These custom applications execute control logic for dynamic network optimization and are a key enabler for enforcing zero-touch network automation and self-configuration.

2.2 Background on DRL

In RL and DRL, agents dynamically interact with an *environment* to maximize a target utility function [69]. Formally, the environment is defined as a Markov Decision Process (MDP). At each time step t, the agent first observes the environment to estimate its *state* $s_t \in \mathcal{S}$, and then takes an *action* $a_t \in \mathcal{A}$ that follows a *policy* $\pi : \mathcal{S} \longrightarrow P(\mathcal{A})$. In general, actions can be multi-modal and consist of combined decisions affecting two or more control parameters. Specifically, any action a_t can be expressed as a c-tuple, *i.e.*, $a_t = (a_t^1, a_t^2, \dots, a_t^c)$, where c represents the number of modes of the action. When the action a_t is taken, the environment transitions from state s_t to the next state s_{t+1} following a transition kernel $T(s_{t+1}|s_t, a_t)$, and generates a *reward* $r_t : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R}$. Starting from an initial state s_0, the tuple $M = (\mathcal{S}, \mathcal{A}, T, r, s_0, \gamma)$ defines the MDP and, in the most general setup, the objective of the agent is to learn a policy π that maximizes the weighted reward $R(\pi) = \sum_{t=0}^{+\infty} \gamma^t r_t$, where γ is a discount factor used to fine-tune the weighted sum of rewards. Specifically, a value of γ closer to 1 would aim at maximizing the long-term reward, while a value closer to 0 would prioritize the short-term reward.

2.3 Background on XAI and XRL

A Primer on AI Explainability. Promoting trustworthiness in AI has received a lot of interest in recent years [20, 54, 72]. While *interpretability* contextualizes the model decision-making in relation to its internal design, *explainability* encompasses *interpretability* and goes beyond by aiming at justifying how and why a model achieves a given output in a human understandable manner [7].

Regarding interpretability, there exist model-agnostic and model-specific XAI techniques. Both reveal which part of the input data or features have more influence on a prediction. SHapely Additive exPlanations (SHAP) [46], Local Interpretable Model-agnostic Explanations (LIME) [64], and Eli5 [35] are model-agnostic and provide model interpretations by identifying input relevance via perturbation techniques. Instead, DeepLIFT [67] and LayeR-wise backPropagation (LRP) [55] provide interpretations by evaluating which activation/neurons were relevant to a prediction via backpropagation. Hence, these techniques need to be specialized for the specific AI model.

XAI and XRL. The literature is inconsistent in the way the terms are classified: [36] defines XAI and XRL as a collection of techniques applicable to supervised and RL respectively, while [10, 63] take a different approach and define XRL as a subset of XAI. We find the latter definition more appropriate as techniques like SHAP and LRP have been applied to DRL [73, 77].

3 MOTIVATION AND CHALLENGES

In this section, we first delve into the design principles of DRL solutions for the RAN, and then illustrate why currently available XAI tools cannot be applied *as-is* to such DRL models.

Data-driven RAN reconfiguration. The RAN is characterized by a large number of parameters and functionalities that can be monitored and configured. In general, these can be categorized according to the timescale at which they are updated. Cell parameters like cell ID, coverage radius, antenna orientation and energy saving mechanisms are usually updated over the course of several seconds, minutes, hours or even days. In contrast, transmission power control policies, interference management, handover, and resource allocation are configured in the sub-second timescale. How to optimally determine these configurations is a well-known problem which has been tackled several times via DTs or DRL. DTs work well in the case of feature selection or rule-based configuration empowered by past historical data like the case of Auric for BSs [49] or Configanator for content-delivery networks [57]. DRL agents are effective solutions for configuring parameters at shorter timescales where dynamic reconfiguration must be achieved by adapting and responding to real-time measurements. Alternatively, they can be used after the exploitation stage for parameter configuration [18]. DTs are self-interpretable vis-a-vis with DRL agents and have been used either directly for rule-based configurations [49, 57], or indirectly to deliver explanations of DRL agents that enforce simple decisions like bitrate selection in Adaptive Bit Rate (ABR) context [53].

Unfortunately, DRL solutions for O-RAN systems are, in general, more complex than those used in the above examples, and they share a common set of features that we elaborate hereafter. *First*, O-RAN networks belong to a class of systems that are very hard to model and observe. Thus, it is common practice to feed DRL agents with a low-dimensionality latent representation of the observed network state rather than with the state itself. Indeed, the latter usually consists of large quantities of heterogeneous and real-time KPI measurements with high variance, which might results in the so-called *state space explosion*, where the number of states is so large that the agents cannot learn an effective policy or would require excessively long training time. The use of autoencoders is a well-established ML tool to mitigate the above issues [38, 62]. *Second*, DRL agents may take hierarchical actions where controllable parameters depend on the value of other non-controllable parameters, previously observed states, or actions taken in the past (*e.g.*, a DRL agent controlling resource allocation policies subject to higher level power control [30]). *Third*, DRL agents may take multi-modal actions that involve diverse control parameters. Practical examples include making joint decisions on user scheduling and antenna allocation [28], RAN scheduling and slicing policies [60], or simultaneous control of computing resources and Modulation and Coding Scheme (MCS) [2].

3.1 Use Case Throughout the Paper

Although *EXPLORA* is general in its design and scope of applicability, in the rest of the paper, we will consider a use case of practical relevance in O-RAN systems. Specifically, we consider [60] where the authors developed a set of xApps jointly controlling RAN slicing and scheduling policies for a set $\mathcal{L}$ of slices: *(i)* enhanced Mobile BroadBand (eMBB); *(ii)* massive Machine-type Communications (mMTC); and *(iii)* Ultra-Reliable and Low Latency Communications (URLLC). We use the configurations and xApps embedding the pre-trained agents, which were provided upon request by the authors. Each agent optimizes resource allocation policies for an Open RAN gNB. For each slice, the DRL agent selects a RAN slicing policy (*i.e.*, the number of Physical Resource Blocks (PRBs) reserved to the slice), and the optimal scheduling policy to serve the User Equipments (UEs) of the slice among Round Robin (RR), Waterfilling (WF), and Proportional Fair (PF).

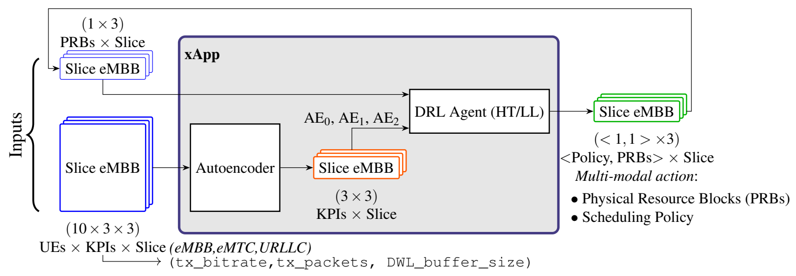


Fig. 2. The DRL framework consisting of an autoencoder and a DRL agent

The DRL agent depicted in Figure 2 takes actions to maximize a target reward by monitoring a set $\mathcal{K}$ of KPIs received from the gNB via the E2 interface and processed by an autoencoder. Specifically, the input $\mathbf{I}$ to the xApp consists of a $M \times K \times L$ matrix where $L = |\mathcal{L}| = 3$ represents the number of slices, $K = |\mathcal{K}| = 3$ is the number of monitored KPIs (*i.e.*, transmission bitrate in Mbps, number of transmitted packets, and size of the downlink (DWL) buffer in bytes), and $M = 10$ is the number of individual measurements collected over the E2 interface for each slice. With a slight abuse of notation, let $\mathcal{K} = \{\texttt{tx\_bitrate}, \texttt{tx\_packets}, \texttt{DWL\_buffer\_size}\}$ be the set of input KPIs. The generic element of the input matrix $\mathbf{I}$ is denoted as $i_{m,k,l}$ with $m = 1, \dots, M$, $k \in \mathcal{K}$, and $l \in \mathcal{L}$.

$\mathbf{I}$ is fed to an autoencoder that produces a latent representation of the input of size $K \times L$ (AE_0, AE_1, AE_2 in Figure 2). This low-dimensional representation is then fed to the DRL agent which embeds a Proximal Policy Optimization (PPO) architecture to compute a c-mode action (*i.e.*, $c = 2$ in our use case) representing the combination of per-slice scheduling and slicing policies. At each time step t, the agents compute a reward function that maximizes the weighted sum of average KPI values for each slice:

$$r_t(\mathbf{I}) = \sum_{l \in \mathcal{L}} w_l \cdot \sum_{m=1}^{M} \frac{i_{m,\kappa(s),l}}{M}, \tag{1}$$

where $\mathbf{I}$ represents the $M \times K \times L$ KPI input matrix, and $\kappa(s) \in \mathcal{K}$ is used to indicate and extract only the target KPI for each slice from $\mathbf{I}$. Specifically, for the eMBB slice the target KPI is `tx_bitrate`, while the target KPIs for the mMTC and URLLC slices are `tx_packets` and `DWL_buffer_size`, respectively. w_l is a real-valued parameter used to weight the importance of each slice toward the reward maximization goal. w_l takes positive values to maximize the reference KPIs of eMBB and mMTC slices and is negative to minimize `DWL_buffer_size` for the URLLC slice as a proxy for minimizing latency. We extend [60] and consider two DRL agent configurations:[1]

- *High-Throughput (HT)* prioritizes eMBB slice's reward contribution over the other two;
- *Low-Latency (LL)* prioritizes the contribution of the URLLC slice over the other two slices.

These agents control the RAN which is a highly non-stationary and dynamic environment with changing channel conditions and they handle diverse traffic profiles in each slice. There are obvious tradeoffs behind the agents' decisions: assigning too many PRBs to the eMBB slice substantially reduces the throughput that UEs of the other two slices will experience (see § 6). Furthermore, agents deal with a continuous state space, the output of the autoencoder, which also makes it hard to quantify the number of states of the system. Therefore, it becomes essential for domain experts access XAI tools to better comprehend agents' decisions under time-varying network conditions.

3.2 Applying XAI Tools Out-of-the-Box

We now elaborate the need for *EXPLORA* by showing the inherent limitations of popular XAI tools (*i.e.*, SHAP and DTs) applied to the HT and LL agents described in § 3.1.

[1] We only retain basic configurations like scaling and normalizing the KPIs in the range $[-1, 1]$.

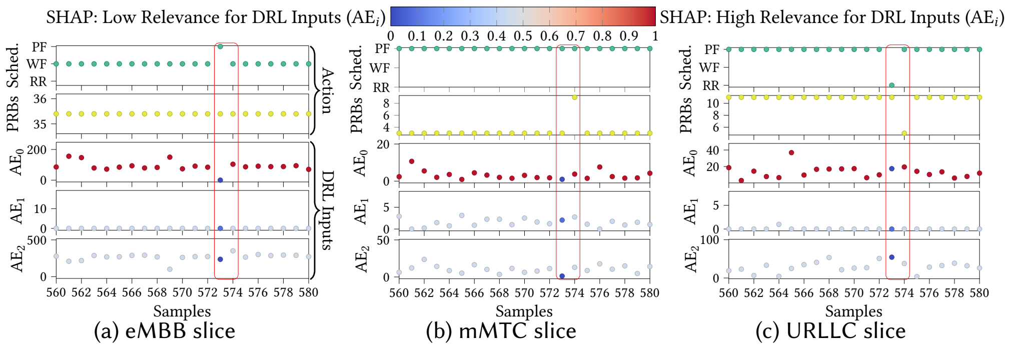


Fig. 3. The fine-grained details of SHAP explanations for the HT agent

SHAP has been proven effective in a number of scenarios in combination with DRL-based solutions. Recent works have leveraged SHAP for power- and UAV-control [25, 77], biomedical scenarios [65] and in the presence of multi-agent systems [26, 41]. In a nutshell, SHAP provides feature-based interpretations by approximating the Shapley values of a prediction and generates global and local explanations in the form of log-odds, which can be turned into a probability distribution with the `softmax` operation.

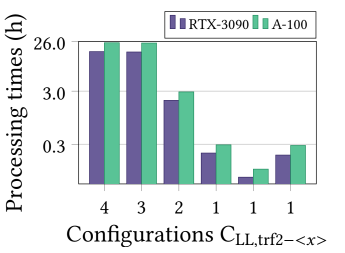
DTs can also explain DRL agents. Metis [53] converts Deep Neural Networks (DNN) and DRL solutions into interpretable rule-based configurations via DTs and hypergraphs. Trustee [31] constructs a high-fidelity and intuitive DT taking as input both the AI model and training dataset. If applied out-of-the-box, SHAP and DTs would be applicable only at DRL agent block of Figure 2. We now elaborate why both are ill-suited to deliver interpretations on the whole architecture.

Figure 3 portrays an example of the outcome of applying SHAP to the HT agent. For each slice, SHAP computes relevance scores by determining the average contribution of each element of the input across all possible permutations of elements' values with respect to the output of the agent, i.e., the corresponding action, expressed as the number of PRBs (taking values in $[0, 50]$) and scheduling policy (among RR, WF, PF) assigned to the slice. The inputs of the DRL agent are the outputs of the autoencoder (AE$_0$, AE$_1$, and AE$_2$ in Figure 2) and not the actual *inputs* of the system, *i.e.*, the KPIs. Formally $\forall i = 1, 2, \ldots, N$, with $N = K \times L$, the score $r_i \in R_N$ is computed as:

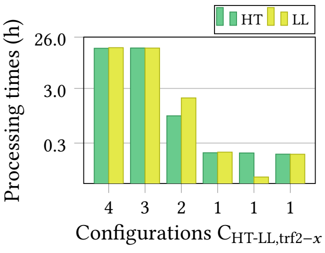
$$r_i(f) = \frac{1}{(N-1)!} \sum_{k=1}^{N-1} \sum_{X_s \subseteq X_t, |s|=k} \left[\binom{N-1}{k} \right]^{-1} \cdot \left(f(X_t) - f(X_s)\right), \tag{2}$$

where $f(X_t)$ is the action taken considering all the features $X_t = \{\text{AE}_0, \text{AE}_1, \text{AE}_2\}$, $s = N-1$ is a subset of the N features of the input sequence, and $f(X_s)$ is the action taken under input X_s.

We now elaborate on the pros and cons of SHAP as a result from extensive tests executed for both HT and LL agents in various settings that include varying the number of users per slice and traffic scenarios (see Table 3 in Appendix A). SHAP is extremely precise in identifying which feature is the most important for taking an action, and reveals the precise operation of the agent. Figure 3 shows for 20 time steps the values of the DRL inputs and outputs. The colors of the color-bar identify the relevance scores of the DRL inputs computed with SHAP: blue and red colors correspond to low and high scores respectively. Samples denoted with low relevance in all the DRL inputs (see index 573 in Figure 3) trigger a change in scheduling policy which follows a change in PRB allocation (*e.g.*, the scheduling policy of the eMBB slice transitions from WF to PF - highlighted with frames in Figure 3). However, SHAP *(i)* is limited by the autoencoder to show non-intuitive explanations, *i.e.*, feature relevance of autoencoder outputs per policy and for each slice and not the actual inputs (the KPIs at user level); and *(ii)* is extremely costly from a computational perspective. Even for few users, computing SHAP values on Nvidia RTX 3090 and A100 SXM4 GPU cards can take hours (see Figure 4(a)): this holds across agents (see Figure 4(b)) and for the other configurations tested. Note that increasing the number of users from 4 to 6 does not produce tangible changes.



(a) On different machines



(b) On different agents

Fig. 4. SHAP computational complexity. Ticklabels on x-axis denote the number of $<x>$ users in the experiment. Experiments with 1 user only are executed for the three slices.

Table 1. Classification accuracy

Config.	DT Accuracy
$C_{LL,trf1-4}$	18.74 %
$C_{HT,trf1-3}$	43.35 %
$C_{LL,trf2-3}$	58.52 %
$C_{LL,trf1-1}$	23.20 %
$C_{HT,trf1-1}$	35.71 %
$C_{HT,trf2-1}$	37.86 %

We also build DTs via an XGBoost model [9] with N features and target label equal to the action, i.e., the output of the DRL agent. Table 1 shows that the ensemble of decision trees is not performing well in the classification task across different configurations. This prevents from explaining how the agent takes an action upon observing X_t. As this part does not become interpretable, it is not possible to provide explanations to the overall framework of Figure 2 via divide and conquer (i.e., explaining first the DRL agent and backpropagate to the input by explaining the autoencoder).

In conclusion, SHAP is too fine-grained and slow. Both SHAP and DTs are unable to provide intuitive explanations to link agent outputs, i.e., actions, with their impact on KPIs, e.g., `tx_bitrate`.

3.3 XAI Challenges

Based on the above discussion, we identify the following three major challenges:

• *Challenge 1:* While autoencoders are of great help to reduce the input dimension and facilitate generalization, they also eliminate the direct *input-output* connection that state-of-the art XAI techniques such as SHAP and LRP build upon. That is, when using autoencoders, SHAP and LRP can only reveal the contribution that the latent space representation had on the action taken by the agent, but lose any information on how the actual input affected the decision-making process.
• *Challenge 2:* Whenever actions depend on either *past* actions or states of the environment (*e.g.*, previous PRB allocation), the decision-making process relies on memory. Such feedback loop adds an additional layer of complexity to the already complex and non-linear relationship between inputs and outputs. This prevents the use of well-established tools such as casual models [48] as it becomes harder to identify the direction of causality, primary cause and primary effect of an action.
• *Challenge 3:* Unlike many popular DRL-based agents that control individual parameters [50, 75], actions that agents take in O-RAN systems are likely multi-modal and involves several control parameters at the same time like RAN slicing and scheduling policy. This makes it hard to leverage existing XAI tools which are primarily tailored to explaining much simpler control actions.

With *EXPLORA*, we aim to address these challenges and provide explainability for the class of DRL-based Open RAN solutions described above. Specifically, we seek explanations that are intuitive, *e.g.*, which actions determine changes in KPIs. We believe that a clear understanding and programmability of the agent behavior is key to lower the barrier for adopting DRL in operational O-RAN networks and provide operators with the necessary tools to understand why the AI has taken a certain decision thus building useful knowledge to design and deploy more efficient networks.

4 THE *EXPLORA* FRAMEWORK

Because of above-mentioned challenges, we deal with systems where the *input-output* link is not available (*Challenge 1*), the decision making process relies on memory (*Challenge 2*), and actions are multi-modal (*Challenge 3*). We address these challenges by embedding into attributed graphs multi-modal actions (nodes) and their impact on the future state (attributes). This allows recording the *consequence* of an action. To distill knowledge, we analyze transitions over time (edges) and quantify statistically the distance between the attributes of the respective nodes. Thus, we can explain the agent behavior by determining the effect of its decisions on the environment with details on the contribution of each component of the multi-modal action.

Next, we first motivate the choice of attributed graphs and elaborate why other data structures like DTs, hypergraphs or multi-layer graphs are not effective (§ 4.1), describe the *EXPLORA* architecture and elaborate on how it interacts with DRL agents (§ 4.2). Then, we explain how to distill knowledge from the attributed graph (§ 4.3). Finally, we show how to leverage the explanations to improve the DRL agent's decision-making process (§ 4.4).

4.1 Design Choices

Domain experts seek to receive from XAI tools explanations inherently related to understanding the logic and dynamics that tie *inputs* to *outputs* [53]. For DRL, this directly translates into understanding the reason why an agent has taken action a_t at time t when observing state during a window δ before t, *i.e.*, $s_{(t-1)+\delta}$. Deriving such logic is not easy, as the unidirectional link *input-output* is essentially broken by the introduction of the autoencoder (*Challenge 1*). However, we leverage the fact that any action a_t will alter the future state $s_{t+\delta}$, which is observable even with the autoencoder. Next, we generate knowledge by analyzing the changes in subsequent actions $a_t \longrightarrow a_{t+1}$ and the corresponding change of states $s_{t+\delta} \longrightarrow s_{t+1+\delta}$. We thus seek a data structure capable of capturing such connections.

In the past, explainability has been delivered via several data structures [14, 19], with the most well-established being DTs [53, 56] and hypergraphs [53]. For example, Metis [53] combines both DTs and hypergraphs to distill knowledge and has proved successful when applied to Pensieve [50], an Adaptive Bit Rate (ABR) DRL system that optimizes video bitrate selection (*i.e.*, the action) by observing and adapting to past video chunk bitrate, throughput, buffer occupancy (*i.e.*, the state). However, XAI tools that use DTs and hypergraphs can be applied only to DRL agents like Pensieve that take unimodal actions such as selecting the bitrate on a per-user basis, which makes them unsuitable to address both *Challenge 1* and *Challenge 3*. When applying DTs to DRL agents with multi-modal actions, we observed a lack of generalization resulting from attempts of pruning the excessive growth scale of the DTs.

We resort to attributed graphs [37]. Formally, an attributed graph G is defined as $G = (N, E, B)$ where $\mathcal{N}$ is a set of nodes, $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$ is a set of edges, and $\mathcal{B}$ is a set of attributes associated with $\mathcal{N}$. Specifically, for each node $n \in \mathcal{N}$, there exists an attribute $b(n) \in \mathcal{B}$ consisting of at most P elements: $b(n) = \{b_1(n), \dots, b_p(n)\}$. Multi-layered graphs and hypergraphs are useful mathematical tools to model complex and multiple relations among multiple entities, like resource allocation in cloud-RAN systems where users are connected to radio units depending on channel conditions and radio units are connected to centralized units according to traffic load [76]. In contrast, attributed graphs capture effectively individual relationships between entities against a common set of properties. This is precisely the case of DRL agents where entities are actions, the relation between entities is temporal (i.e., action a_{t+1} occurs after a_t) and the common set of properties of the entities maps to the state s associated to each action a. We will describe how to build the attributed graph in § 4.2.

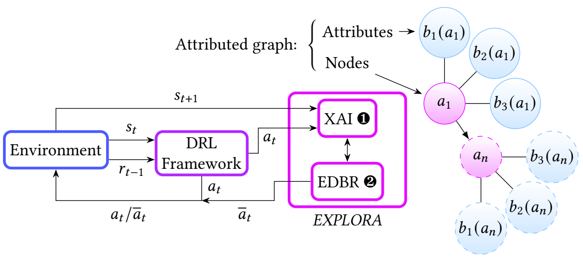


Fig. 5. *EXPLORA* interaction with a DRL agent

4.2 *EXPLORA*: System Architecture

The architecture. *EXPLORA* consists of the *XAI module* (❶ in Figure 5) and the XAI-aided *Explanation-Driven Behavior Refiner (EDBR) module* (❷ in Figure 5). ❶ generates post-hoc explanations about the agent behavior by building the attributed graph G throughout an observation window W. ❷ leverages such explanations to understand the decision-making process (§ 4.3), identifies inefficiencies and improves overall network performance (§ 4.4).

Figure 5 shows how these two blocks interface with the DRL agent as well as their workflow execution. The DRL agent interacts with the environment as described in § 2.2, with the sole exception that the input is the latent representation and not the state. This approach can be easily applied to a variety of DRL models such as Deep Q-Network (DQN) [75], PPO [60] or Asynchronous Advantage Actor-Critic (A3C) [50]. These differ in the way the agent learns the optimal policy.

Generating the attributed graph. To generate the attributed graph G described in the previous section, *EXPLORA*'s XAI module (❶ in Figure 5) performs the following operations. During $t \in W$, each action $a_t = (a_t^1, \dots, a_t^c)$ (representing the decision of RAN slicing and scheduling policies with $c = 2$ in our use case) is mapped to a node n of the graph. The impact of a_t on the future environment state, s_{t+1}, is instead mapped to the attribute $b(n)$. For our use case, we embed the distribution for each monitored KPIs and for each slice as the $b_p(n)$ element of the attribute, where $p \in \mathcal{P}, \mathcal{P} = \mathcal{K} \times \mathcal{L}$ and $P = K \cdot L$. From § 3.1 and § 2.2, the action $a \in \mathcal{A}$ is multi-modal and can be defined as a c-tuple $a = (a^1, \dots, a^c)$. Similarly, the state $s_t \in \mathcal{S}$ can be defined as a tuple which, in our case, is the input matrix $\mathbf{I}$. An edge connecting actions a_t and a_{t+1} (a_1 and a_n in Figure 5) indicates the transition between actions taken at subsequent time instances. As $b(a_t)$ indicates the effect of a_t on s_{t+1}, then the unidirectional edge $a_t \longrightarrow a_{t+1}$ indicates that $b(a_t)$ is the input state space for a_{t+1}. This closes the loop broken by the presence of the autoencoder. DTs and hypergraphs fail precisely in providing a link connecting actions to their consequences onto the environment and then on the next action. The same would apply to multi-layer graphs embedding one of the components of the multi-modal action in each layer. After W observations, G makes it possible to distill knowledge regarding the agent's behavior (§ 4.3) and the expected outcome of an action is known in advance and before it is actually enforced.

In the majority of DRL-based systems for Open RAN, the state and action spaces $\mathcal{S}$ and $\mathcal{A}$ of the DRL agents might be very large, as they embed real-valued variables such as throughput, channel conditions, buffer size, latency, to name a few. Therefore, to tackle this complexity and contain the size of the attribute space, we operate as follows. First, the set $\mathcal{P}$ has limited dimension, as it depends on the number K of KPIs included in the state s and the number L of slices. Second, each element $b_p(n) \in b(n)$ only stores the distribution of each KPI. In the right portion of Figure 5, we show an example of the attributed graph G with two actions (*i.e.*, nodes) a_1 and a_n, each storing 3 attributes (i.e, $b_1(a_1)$, $b_2(a_1)$, and $b_3(a_1)$ for a_1). In Appendix B, we provide explanations about the generation of G for three consecutive step. Next, we show how to distill knowledge from G (§ 4.3), and how this knowledge can be used by module ❷ to improve overall performance (§ 4.4).

4.3 Synthesizing Network-Oriented Explanations

EXPLORA distills knowledge by analyzing transitions between actions (edges in G). This allows to characterize the individual contribution of each component of a multi-modal action. For example, in our use case, the graph will reveal if a multi-modal action (recall that $c = 2$) produces changes in KPIs like throughput because of a change in RAN slicing, scheduling policies or both. This would not be possible with other data structures that do not link attributes (KPIs) to nodes (actions). Any action taken at time t can be expressed as a 2-tuple, (in our case, slicing and scheduling policy) $a_t = (a_t^1, a_t^2)$. Then, at any transition between t and $t + 1$, there exists 2^c possible combinations to describe how actions transition between a_t and a_{t+1}. For example, in one case $a_t^1 = a_{t+1}^1$ but $a_t^2 \neq a_{t+1}^2$. In another, $a_t^1 \neq a_{t+1}^1$ and $a_t^2 = a_{t+1}^2$. The remaining two cases are: the case where the action remains the same, *i.e.*, $a_t = a_{t+1}$, and the one where they are completely different, *i.e.*, $a_t^i \neq a_{t+1}^i$ for $i = 1, 2$.

EXPLORA builds a set of ordered pairs (π, v), where each π maps the action transition $a_t \longrightarrow a_{t+1}$ and v maps the corresponding change of impact to the respective states $s_{t+1} \longrightarrow s_{t+2}$ (in our use case, each π would correspond to a RAN slicing and policy scheduling enforced in two subsequent timesteps). We can now quantify such impact via the attributes $b(n)$ and $b(n + 1)$ of the nodes n and $n + 1$ representing the action transition $a_t \longrightarrow a_{t+1}$. Recalling that each $b_p(n)$, with $p \in \mathcal{P}$ embeds a distribution of each KPI for each slice, we can compare each $b_p(n + 1) \longrightarrow b_p(n + 2)$ using either statistical techniques like the Jensen Shannon divergence or, for example, directly comparing $\text{avg}\{b_p(n + 1)\}$ and $\text{avg}\{b_p(n + 1)\}$. The result of such comparison is stored in $v = v_1, v_2, \ldots, v_p$ and is the knowledge we leverage to produce informative explanations, *i.e.*, which is the effect that changes of RAN slicing and/or scheduling policies produce on KPIs like throughput or buffer size. To distill knowledge, *EXPLORA* uses DTs. Specifically, it builds a DT where the set of features is v and the target label is the corresponding class of action transition among the 2^c possible combinations. The resulting DT identifies which changes on KPIs are associated to each class of transitions and the visual inspection of the tree provides informative explanations in the form: *"the agent uses completely different transitions ($a_t^i \neq a_{t+1}^i$) to increase the* `tx_bitrate`." Note that the use of DTs for knowledge distillation does not imply that DTs could substitute the original DRL agents in taking joint decisions on PRBs allocation and scheduling policy. We will show in § 6.2 how to turn these explanations into a concise and effective summary of the agent's behavior.

4.4 Optimizations Enabled by *EXPLORA*

With the distilled knowledge generated by ❶, *EXPLORA*'s EDBR module (❷) makes it possible to perform informed and targeted ad-hoc adjustments to the agent's behavior to modify its decision-making process with the goal of improving the overall network performance. This fits well with intent-based networking [39] which aims at delivering a simplified and agile network management where complex configurations are translated into high-level intents.

Following O-RAN specifications, DRL agents are usually trained offline[2] on data that might not necessarily accurately reflect the type of data observed in real-time. This is because an ideal training is extremely complex[3] and impractical for production networks where the number of scenarios to be accounted for is huge (e.g., number of served users, traffic dynamics, propagation characteristics of the environment, among others). Optimal actions on training data may thus perform poorly in a live network, *e.g.*, because of different traffic profiles and topologies [60]. *EXPLORA* builds and updates G over time, and hence can be used to identify inefficient actions that are attributed to

[2]Note that this is a strict requirement for any AI-based xApp and rApp [61].
[3]To train agents HT and LL, the data collection took two and a half months on Colosseum and the actual training operation took approximately 10-15 hours on a single NVIDIA A100 GPU, depending on the model.

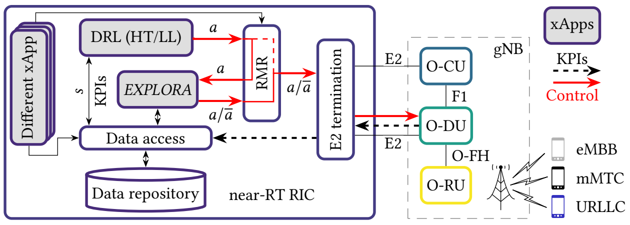


Fig. 6. *EXPLORA* integration into the O-RAN reference architecture

imperfect training and offer mechanisms that prevent inefficiencies. While the explanations can simply be used to understand how agents make decisions, we further discuss two concrete possible uses that domain experts can make of the distilled knowledge.

• *Opt 1: Intent-based action steering* replaces an action selected by the agent with another extracted from the attributed graph G to fulfill specific intents. For example, if an agent takes an action that G marks as potentially resulting in an expected low reward, *EXPLORA* can suggest an alternative action from G that would yield a higher expected reward.
• *Opt 2: Action shielding* prevents the agent from taking specific actions considered dangerous. Unlike action steering, action shielding completely inhibits specific actions and is mainly used for security purposes [1] (*e.g.*, for active voltage control [8]).

Given the highly non-stationary dynamics of the RAN and the class of resource allocation DRL-based solutions considered in this work, *Opt 1* looks more attractive as it enables to programmatically control the agent behavior *with consciousness* and thanks to the understanding of the impact that certain actions have on the state of the network. We will discuss policies for action steering in § 5.2, and show such benefits in § 6.3.

5 *EXPLORA* IMPLEMENTATION

In this section, we illustrate how to embed *EXPLORA* into xApps (§ 5.1), and elaborate on possible action replacement strategies to improve overall KPI performance (§ 5.2).

5.1 Integrating *EXPLORA* in xApps

Thanks to the flexibility offered by the O-RAN architecture, we identify three possible strategies to make *EXPLORA* an operational component of the near-RT RIC. Specifically, it can be: *(i)* a base component of the RIC itself hosted outside the xApp domain; *(ii)* a component of each xApp, embedded in the microservice that executes the DRL agent; or *(iii)* a standalone xApp that interacts with one or more xApps hosting DRL agents. Strategy *(iii)* provides the best trade-off between flexibility and cost: the dedicated *EXPLORA* xApp can be replicated as needed to support diverse use cases and interacts directly with the xApps hosting DRL agents without requiring changes as in strategy *(ii)*, or to the near-RT RIC platform.

The left part of Figure 6 illustrates how the *EXPLORA* xApp interacts with other components of the O-RAN architecture. The E2 termination of the near-RT RIC platform routes E2 data (KPIs) to a data access microservice that stores it in a data repository inside the RIC. The same microservice is queried by the DRL and *EXPLORA* xApps to access stored data. As discussed in § 3.1, the DRL xApp uses the KPIs to first feed the autoencoder and then the agent (HT or LL in our use case), thus making such KPIs represent the state s observed by both the DRL and *EXPLORA* xApps.

The DRL xApp then generates an action a in a RIC internal message (*e.g.*, for the E2 service model RAN Control) that is sent to the RIC Message Router (RMR), as shown in Figure 6. The RMR is in charge of dispatching internal messages across different components, *e.g.*, xApps and E2 termination, and can be configured with ad-hoc routes based on the endpoints and message ID [59].

Therefore, to seamlessly integrate the *EXPLORA* xApp within the RIC, we configure the RMR to route the RAN control messages to the *EXPLORA* xApp. If the *EXPLORA* xApp is not deployed, the RMR delivers the control message to be enforced at the O-DU from the DRL agent xApp via the E2 termination directly (see the red dashed line in Figure 6).

Once the *EXPLORA* xApp receives action a, it can decide whether to forward it *as is* to the RMR, which then sends it to the E2 termination, or to update it with an action $\overline{a}$ computed by following one of the strategies discussed in § 5.2. The *EXPLORA* xApp can also interact with the data access microservice to save the explanations and information on the state/action/explanation tuple in the data repository. This can later be accessed by the network operator for quality assurance, debugging, and/or datasets generation purposes.

5.2 Strategies for Action Steering

We now show ad-hoc adjustments that domain experts (*e.g.*, network operators) can use to ultimately improve users' KPIs by defining high-level intents. In the case of imperfect training (see § 4.4), the agent might observe previously unseen input data, which might result in taking a sub-optimal (or inefficient) action a_t. a_t is sub-optimal if its expected reward could be improved by another action $a_t^{'}$. The EDBR module uses the knowledge built and distilled via the synthesis of explanations from G to suggest another action $\overline{a}_t$ whose expected reward and impact on the future state is statistically known from the past. Algorithm 1 provides the details of the implemented intent-based action steering strategies, which are summarized hereafter. For the sake of demonstration, and due to space limitations, we limit the graph exploration to the first hop nodes of G only. This restriction highlights the benefits of the strategies in a worst-case scenario.

- *AR 1* - "Max-reward": this strategy replaces an action a_t suggested by the agent that is expected to yield a low reward with one extracted from the graph (a_G) that is expected to yield a higher reward. This can be achieved by extracting the attributes $b(a_t)$ and $b(a_G)$ from the attributed graph G, and computing the expected reward (defined in (1)) using the average KPI values stored in G. For the High-Throughput (HT) agent, we expect this strategy to favor the eMBB slice.
- *AR 2* - "Min-reward": this strategy substitutes an action a_t suggested by the agent and expected to result in a high reward with an action a_G from G that is expected to yield a lower reward. For the Low-Latency (LL) agent, we expect this strategy to favor the URLLC slice.
- *AR 3* - "Improve bitrate": similarly to "Max-reward", this strategy replaces an action a_t computed by the DRL agent with an expected low reward with another action a_G that is expected to deliver a high `tx_bitrate`. We expect this strategy to always favor the eMBB slice for any agent.

With a slight abuse of notation, let $r(b(a_t))$ be the reward computed from (1) where instantaneous KPIs are replaced with their average values computed from the distributions stored in the attributes $b(a_t)$ for the action a_t. For all the strategies, we compare the expected reward $r(b(a_t))$ that would be achieved by taking action a_t with the measured average reward $\text{avg}_{x=t-O-1}^{t-1} r(a_x)$ we have observed across the last O time steps.

6 EXPERIMENTAL EVALUATION

In this section, we first describe the O-RAN platform used to validate *EXPLORA* in a broad range of settings (§ 6.1). Then, we empirically evaluate *EXPLORA* explanations (§ 6.2) and benchmark its ability to programmatically improve the agents' behavior with ad-hoc adjustments (§ 6.3).

6.1 O-RAN Testbed and Setup Description

We leverage the open-source OpenRAN Gym framework [6] to deploy a reference O-RAN architecture (as that of Figure 6) with our custom xApps and perform data collection. If features an O-RAN-compliant near-RT RIC [58]; instances of the E2 interface to connect the RIC and the RAN;

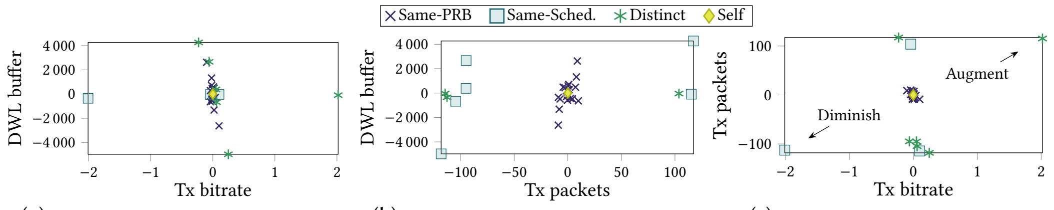
Algorithm 1 Strategies for intent-based action steering

```
Require: $G = (N, E, B)$; action suggested by the agent $a_t$; previous action $a_{t-1}$; $O$; Steering strategy $\alpha \in \{AR\,1, AR\,2, AR\,3\}$
 1: $\omega \leftarrow$ Result of $r(b(a_t)) < \text{avg}_{x=t-O-1}^{t-1} r(a_x)$
 2: if $(\omega, \alpha)$ == $(\texttt{True}, AR\,1)$ OR $(\omega, \alpha)$ == $(\texttt{False}, AR\,2)$ OR $(\omega, \alpha)$ == $(\texttt{True}, AR\,3)$ then
 3:     if $n_{t-1} \in N$ then
 4:         Initialize $Q \leftarrow \emptyset$
 5:         Mark $n_{t-1}$ as visited, add $n_{t-1}, b(a_{t-1})$ to $Q$
 6:         for each neighbor $w$ of $n_{t-1}$ do
 7:             if $w$ is not visited then
 8:                 Mark node $w$ as visited
 9:                 $a \leftarrow$ Action corresponding to node $w$
10:                 Add $w, b(a)$ to $Q$
11:         Execute procedure $\alpha \in \{AR\,1, AR\,2, AR\,3\}$
12:     else
13:         Send $a_t$ to RMR
14: procedure $AR\,1$: "MAX-REWARD"($Q$,$a_t$)
15:     $a_{max} = \arg\max_a \{r(b(a)) : b(a) \in (w, b(a)) \in Q\}$
16:     if $r(b(a_{max})) > r(b(a_t))$ then
17:         $a_t \leftarrow a_{max}$
18:     Send $a_t$ to RMR
19: procedure $AR\,2$: "MIN-REWARD"($Q$,$a_t$)
20:     $a_{min} = \arg\min_a \{r(b(a)) : b(a) \in (w, b(a)) \in Q\}$
21:     if $r(b(a_{min})) < r(b(a_t))$ then
22:         $a_t \leftarrow a_{min}$
23:     Send $a_t$ to RMR
24: procedure $AR\,3$: "IMPROVE BITRATE"($Q$,$a_t$)
25:     $j \leftarrow$ Index of tx_bitrate KPI in the attributes $b(\cdot) \in G$
26:     $a_{br} = \arg\max_a \{b_j(a)) : b_j(a) \in b(a) \in (w, b(a)) \in Q\}$
27:     if $b_j(a_{br})) > b_j(a_t))$ then
28:         $a_t \leftarrow a_{br}$
29:     Send $a_t$ to RMR
```

integrates gNBs and UEs from open-source software-defined 3GPP stacks (specifically, we used srsRAN BSs and UEs for this study); and features stubs for xApps that can be extended to implement the desired control functionalities, *i.e.*, *EXPLORA* and the DRL agents.

We deploy OpenRAN Gym components in Colosseum, a wireless emulation testbed with hardware in the loop [52]. Colosseum provides 128 pairs of programmable compute nodes with a white-box server and a software-defined radio (NI/Ettus USRP X310). These nodes can be remotely accessed by researchers to run experiments on a catalog of scenarios capturing diverse path loss, shadowing, and fading conditions extracted from real-world wireless environments.

For this work, we consider an urban scenario with 42 UEs and 7 BSs whose locations are extracted from the OpenCelliD database [70] to match that of a real cellular deployment in Rome, Italy. Users are deployed uniformly near the BSs, which offer connectivity via a 10 MHz radio channel with a sub-carrier spacing of 15 KHz. Each UE is assigned to a network slice (*i.e.*, eMBB, mMTC, and URLLC) and receives traffic according to the slice it belongs to. Specifically, we use Colosseum's traffic generator (based on MGEN [71]) to generate two traffic profiles. In the first one (TRF1), eMBB UEs receive 4 Mbit/s constant bitrate traffic in DWL, while mMTC and URLLC UEs receive



(a) `tx_bitrate` and `DWL_buffer_size` (b) `tx_packets` and `DWL_buffer_size` (c) `tx_bitrate` and `tx_packets`

Fig. 7. Detailed explanations for the HT agent's behavior

a Poisson traffic in DWL with average 44.6 kbit/s and 89.3 kbit/s, respectively, as in [60]. TRF1 is used for generating the training dataset and for live experiments. The second profile (TRF2) features 2 Mbit/s constant bitrate traffic for eMBB, and 133.9 kbit/s and 178.6 kbit/s Poisson traffic for mMTC and URLLC, respectively.

Table 3 (in the Appendix A) lists the complete set of experiment configurations. All run for 30 minutes. The experiments on the left portion of the table are used in § 3.2 and in § 6.2. The experiments on the right portion of the table are used in § 6.3 and run with an additional online training phase, which helps agents adapt to a change in the environment. Overall, we run 28 hours of experiments in Colosseum to gather the data for all the 40 experiments in Table 3. This adds to the more than 150 hours of data collection for the offline dataset used to train the DRL agents.

6.2 System Explanations

We distill post-hoc explanations for the HT and LL agents. Without loss of generality and due to space limitations, Figure 7 reports the case of TRF1 for the HT agent only (the resulting graph for agent HT is shown in Appendix B). However, the corresponding discussion related to the agent LL is given in Appendix C. Unlike SHAP, which takes 26 h to generate non-intuitive explanations for experiments with 4+ users (see § 3.2), *EXPLORA* takes on average only 2.3 *s* to generate, process the graph and synthesize the intuitive explanations relating actions to KPIs variation (40695× faster).

EXPLORA provides explanations by breaking down at the level of individual KPIs the effect of transitions between multi-modal actions. As actions have two modes, we find in G the following categories of transitions: *(i)* "Same-PRB" is a transition between actions with the same PRB allocation, *(ii)* "Same-Sched." is a transition between actions with the same scheduling policy, *(iii)* "Distinct" is a transition between actions with different PRB allocation and scheduling policies, and *(iv)* "Self" denotes no transition, *i.e.*, the same action is repeated in two subsequent steps.

Figure 7 provides domain experts with the required information to understand how agents operate. Each point of the scatter plots represents a KPIs variation observed from state s_t to s_{t+1} determined by the transition from action a_{t-1} to action a_t. Overall, "Distinct" transitions produce large variations of `DWL_buffer_size` while "Same-PRB" triggers lower `DWL_buffer_size` variations with no change in `tx_bitrate` (see Figure 7(a)). Across all agents and settings, "Self" and "Distinct" transitions are respectively around 5% and 50% of the total.

To simplify the results in Figure 7 for non-expert users, we summarize concise explanations that *generate new knowledge* about the reason why the agents use the different categories of actions. In other words, *EXPLORA* spots the intertwined relation between KPIs and how actions determine their change. Figure 8 and Table 2 show such summary in a more human-friendly form. Specifically, Figure 8 is obtained by constructing a DT on the data shown in Figure 7 (*i.e.*, the outcome of *EXPLORA*) *and not on the agent itself.* As discussed in § 4.1, this is necessary as DTs perform poorly if applied directly to the agent. The new knowledge is generated by tracing the decisions taken at each branch while traversing the tree from the root to the leaves. This process pinpoints the decision-making criteria of the agent for a given category of action transitions. If the same category appears in more than one leaf, then the decision-making process is complex and the same class applies to different KPI variations. Surprisingly, the agent uses "Same-Sched" to reduce the throughput (*i.e.*, `tx_bitrate`, see the node on the left branch of the DT) and the number of `tx_packets` (on the right branch of the DT, which corresponds to the bottom left point in Figure 7(c). The agent uses "Same-PRB" to sustain current throughput by using actions that produce minor variations in the other KPIs (`tx_packets` and `DWL_buffer_size`), and uses "Self" when it does not observe any variation KPIs. Unlike Figure 7, Figure 8 and Table 2 deliver intuitive explanations that are effective to shed light on the agent's behavior.

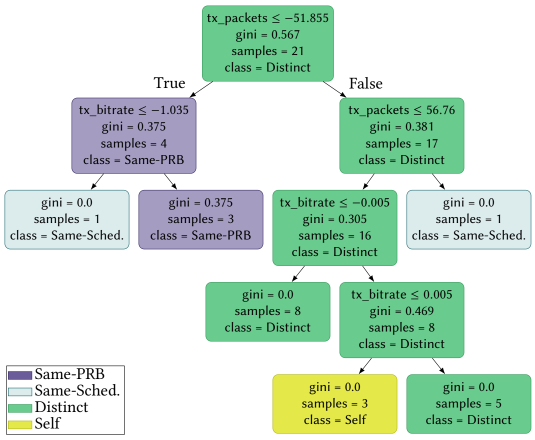


Fig. 8. DT on *EXPLORA* explanations for the HT agent

Table 2. HT agent: summary of explanations

Transition	Interpretation
Same-PRB	Produces minor changes in KPIs
Same-Sched.	Diminishes `tx_bitrate`, other KPIs augment/diminish according to the previous state
Distinct	Increases `tx_bitrate`, other KPIs augment/diminish according to the previous state
Self	No change in KPIs

6.3 Ad-hoc Adjustments

We now illustrate how explanations can steer the decision-making process toward desired goals defined as intents. To this end, we implement the three policies described in § 5.2 within the *Improve* module of *EXPLORA* and benchmark their capability in reducing the `DWL_buffer_size` for the LL agent ("Min-reward" or *AR 2*, in Figure 9) and improve `tx_bitrate` for the HT agent ("Max-reward" or *AR 1*, and "Improved bitrate" or *AR 3* in Figure 10). We also show the corresponding counterpart effect on the other KPIs. We derive the distribution of the KPIs once G is made available after online training and compare against a baseline without action change.

"Max-reward" and "Improved bitrate" provide different levels of bitrate improvement (see Figure 10): the latter is a more aggressive strategy because it explores the graph looking for first-hop nodes with bitrate attributes directly. On the contrary, "Max-reward" changes action solely on the basis of the expected reward that a given first-hop node would provide. The "Min-reward" strategy is effective as it reduces significantly the tail of the buffer size occupancy with minor changes in `tx_bitrate`, thus allowing for faster transmission of URLLC traffic (see Figure 9). Finally, we assert that the intent-based action steering policies do not harm the capabilities of the agent to generalize and adapt to changes in the systems (see § D).

7 DISCUSSION ON *EXPLORA*

Solving the Challenges. To summarize, the use of attributed graphs solves the challenges presented in § 3.3. It makes it possible to establish the "actions-outputs" link, thereby making the decision-making process observable when it would not be otherwise (*Challenge 1*); avoids primary cause and effect being undetermined because of memory (*e.g.*, it determines whether an action is mainly attributed to the past PRB allocation or to the change of KPIs) (*Challenge 2*); and enables understanding the implications of individual components of multi-modal actions by comparing the distributions of KPIs (attributes) between transitions (edges) of interconnected nodes (actions) with the same or distinct individual multi-modal component (*Challenge 3*).

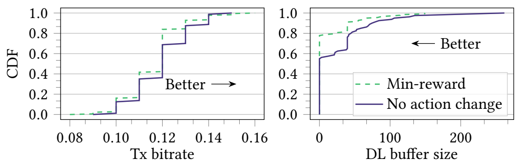
Generalizability. We now comment on how *EXPLORA* can generalize within and outside O-RAN networks, e.g., by interfacing with AI/ML algorithms running on the BSs directly, or on generic network controllers not tied to O-RAN specifications. Further, we will discuss how *EXPLORA* can be applied to nodes external to the RAN. The discussion holds as long as the general system architecture is based on the use of autoencoders and DRL agents, and actions operate on two or more variables as in [21, 28]. Both are examples of agents enforcing multi-modal actions in the RAN. The multi-agent framework in [21] jointly determines BS selection and power requirement to maximize user throughput during handovers. The agent in [28] observes the distribution of channel quality indicators per user, amount of data to transmit, and traffic type as state s and takes joint decisions on user scheduling and antenna allocation with multi-modal actions a that hierarchically *(i)* prioritize users, *(ii)* decide the number of antennas per user, and *(iii)* select a precoding algorithm that maximizes spectral efficiency according to the above decisions. If applied to this agent, *EXPLORA* would pinpoint the rationale for the three modes of the actions individually and any combination thereof, *e.g.*, which precoding algorithm is typically used by the high priority users or how many antennas are typically allocated to regular users. Aside from O-RAN networks, *EXPLORA* can be interfaced with WiFi access points in which DRL-based closed-control loops are enacted to jointly tune parameters and configurations of the access point (e.g., a multi-modal action handling power control and beam steering). In this case, network metrics and channel measurements (e.g., channel state information with other WiFi nodes) would be first passed through an autoencoder (either running locally or hosted on an external controller) for dimensionality reduction, and then forwarded to a DRL agent that computes control actions to be enforced at the WiFi access point. Similarly to the previous case, *EXPLORA* can be set up to read the input to the autoencoder, and steer the output of the DRL agent to tailor the network control to specific conditions and environment.

From Laboratory to Production Environments. *EXPLORA* can also be used to speed up the transition of AI/ML solutions from laboratory to production-like environments. Indeed, xApps can be first developed, pre-trained offline and tested in controlled laboratory environments, thus complying to the O-RAN specifications. Then, they can be transitioned to production-like environments, where they are used to manage complex and large networks. However, the very many production environments where xApps are deployed might not always reflect the laboratory conditions where they were originally pre-trained and tested. Hence, a tool such as *EXPLORA* can help steer the actions, and adapt xApps to the novel environments with minimal online re-training, as it usually happens for DRL agents to adapt to new environments [23, 60].

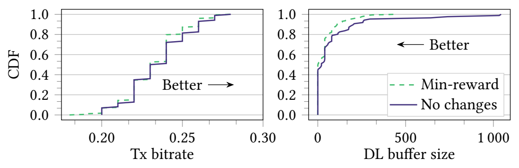
8 RELATED WORK

The last years have seen a surge in the uptake of XAI for AI-based networking systems. Seminal works [11, 80] opened the path forward to interpretability in networking contexts. Auric [49] and NeuroCuts [44] rely on DTs that are self-interpretable AI models. AT&T developed Auric to automatically configure BSs parameters while NeuroCuts performs packet classification with RL. Unfortunately, and as mentioned earlier, DTs are not very effective in complex networking problem such as the RAN.

XAI for Networking. XAI is at an early stage of conceptualization and adoption in mobile networks [16, 22, 40]. The lack of explainability leads to poor AI/ML model design, which might facilitate adversarial attacks [15, 66, 78]. Metis [53] interprets diverse Deep Learning (DL)-based networking systems, converting DNN policies into interpretable rule-based controllers and highlighting critical components. Metis works well when the relation between *inputs* and *outputs* is

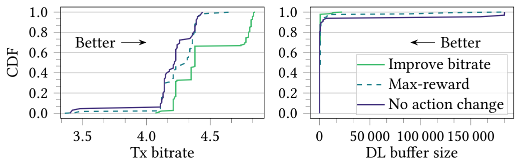


(a) TRF1

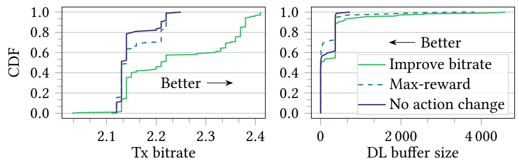


(b) TRF2

Fig. 9. `Min reward` policy effect on URLLC slice in different traffic scenarios



(a) TRF1



(b) TRF2

Fig. 10. `Max reward` policy and `improve bitrate` effect on eMBB slice in different traffic scenarios

explicit, which does not address *Challenge 1*. The work in [79] uses a teacher-student framework to improve robustness of DRL-based networks. The teacher infuses a set of white-box logic rules defined by humans into a DRL-agent, *i.e.*, the student. Specifically, the student maximizes the expected cumulative reward while minimizing the distance to the teaching data. Unfortunately, in complex systems like the RAN, defining white-box logic rules a priori may be prohibitively time-consuming and inefficient.

The above research efforts still remain preliminary in regard to the design of explainability techniques applicable to complex AI models (see § 3.3) for real networking systems. *EXPLORA* fills precisely this gap for the Open RAN case.

Robustness. Ensuring model robustness is important and can be done through anomaly detection [24] or formal verification [12, 33]. *Shielding* is a safety mechanism that inhibits an agent from taking a potentially risky actions provisioning DRL agents with an additional layer of robustness. Shields may be constructed in post-training by programmatically determining forbidden actions [1], or infer the dynamics of the system and preventing it from reaching hazardous states [4]. At training times, shields can restrict the exploration of the state space [34] thereby limiting the agent knowledge right from the start. By contrast, *EXPLORA* performs adaptive action-steering.

9 CONCLUSIONS

In this paper, we propose *EXPLORA*, a new framework to explain the class of resource allocation DRL-based solutions for cellular networks. *EXPLORA* synthesizes model explanations that facilitate monitoring and troubleshooting for domain experts. We showcase the benefits of *EXPLORA* in a typical Open RAN scenario and apply it to a set of DRL agents executing as O-RAN xApps that govern RAN slicing and scheduling policies. *EXPLORA* not only synthesizes clear and concise explanations, but can also improve overall performance by using explanations to proactively identify and substitute actions that would lead to low expected rewards programmatically.

ACKNOWLEDGMENT

Dr. Fiandrino visited Northeastern University with the support of a José Castillejo/Fulbright scholarship (CAS21/00500). This work is partially supported by the Spanish Ministry of Science and Innovation through the Juan de la Cierva grant IJC2019-039885-I and grant PID2021-128250NB-I00 ("bRAIN"), by the project RISC-6G, reference TSI-063000-2021-63, granted by the Ministry of Economic Affairs and Digital Transformation and the European Union-NextGenerationEU through the UNICO-5G R&D Program of the Spanish Recovery, Transformation and Resilience Plan, and by the U.S. National Science Foundation under grants CNS-2112471, CNS-1925601, and CNS-2312875.

REFERENCES

[1] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. 2018. Safe Reinforcement Learning via Shielding. *Proc. of AAAI Conference on Artificial Intelligence* 32, 1 (Apr. 2018). https://doi.org/10.1609/aaai.v32i1.11797

[2] Jose A. Ayala-Romero, Andres Garcia-Saavedra, Marco Gramaglia, Xavier Costa-Perez, Albert Banchs, and Juan J. Alcaraz. 2019. VrAIn: A Deep Learning Approach Tailoring Computing and Radio Resources in Virtualized RANs. In *Proc. of ACM MobiCom.* https://doi.org/10.1145/3300061.3345431

[3] Luca Baldesi, Francesco Restuccia, and Tommaso Melodia. 2022. ChARM: NextG Spectrum Sharing Through Data-Driven Real-Time O-RAN Dynamic Control. In *Proc. of IEEE INFOCOM.* 240–249. https://doi.org/10.1109/INFOCOM48880.2022.9796985

[4] Osbert Bastani. 2021. Safe Reinforcement Learning with Nonlinear Dynamics via Model Predictive Shielding. In *Proc. of American Control Conference.* 3488–3494. https://doi.org/10.23919/ACC50511.2021.9483182

[5] Lorenzo Bertizzolo, Tuyen Xuan Tran, John Buczek, Bharath Balasubramanian, Rittwik Jana, Yu Zhou, and Tommaso Melodia. 2021. Streaming from the Air: Enabling Drone-sourced Video Streaming Applications on 5G Open-RAN Architectures. *IEEE Transactions on Mobile Computing* (2021), 1–1. https://doi.org/10.1109/TMC.2021.3129094

[6] Leonardo Bonati, Michele Polese, Salvatore D'Oro, Stefano Basagni, and Tommaso Melodia. 2022. OpenRAN Gym: An Open Toolbox for Data Collection and Experimentation with AI in O-RAN. In *Proc. of IEEE WCNC.* 518–523. https://doi.org/10.1109/WCNC51071.2022.9771908

[7] David A Broniatowski et al. 2021. Psychological foundations of explainability and interpretability in artificial intelligence. *NIST, Tech. Rep* (2021).

[8] Pengcheng Chen, Shichao Liu, Xiaozhe Wang, and Innocent Kamwa. 2023. Physics-Shielded Multi-Agent Deep Reinforcement Learning for Safe Active Voltage Control With Photovoltaic/Battery Energy Storage Systems. *IEEE Transactions on Smart Grid* 14, 4 (2023), 2656–2667. https://doi.org/10.1109/TSG.2022.3228636

[9] Tianqi Chen and Carlos Guestrin. 2016. XGboost: A scalable tree boosting system. In *Proc. of ACM SIGKDD.* 785–794.

[10] Richard Dazeley, Peter Vamplew, and Francisco Cruz. 2021. Explainable reinforcement learning for broad-XAI: a conceptual framework and survey. *arXiv preprint arXiv:2108.09003* (2021).

[11] Arnaud Dethise, Marco Canini, and Srikanth Kandula. 2019. Cracking open the black box: What observations can tell us about reinforcement learning agents. In *Proc. of ACM NetAI.* 29–36.

[12] Arnaud Dethise, Marco Canini, and Nina Narodytska. 2021. Analyzing Learning-Based Networked Systems with Formal Verification. In *Proc. of IEEE INFOCOM.* 1–10. https://doi.org/10.1109/INFOCOM42981.2021.9488898

[13] Marcin Dryjański, Łukasz Kułacz, and Adrian Kliks. 2021. Toward Modular and Flexible Open RAN Implementations in 6G Networks: Traffic Steering Use Case and O-RAN xApps. *Sensors* 21, 24 (2021). https://doi.org/10.3390/s21248173

[14] Mengnan Du, Ninghao Liu, and Xia Hu. 2019. Techniques for interpretable machine learning. *Commun. ACM* 63, 1 (2019), 68–77.

[15] Wang Fei, Hugh Ethan, and Li Baochun. 2023. More than Enough is Too Much: Adaptive Defenses against Gradient Leakage in Production Federated Learning. In *Proc. of IEEE INFOCOM.* 1–10. https://doi.org/10.1109/INFOCOM53939.2023.10228919

[16] Claudio Fiandrino, Giulia Attanasio, Marco Fiore, and Joerg Widmer. 2022. Toward native explainable and robust AI in 6G networks: Current state, challenges and road ahead. *Computer Communications* 193 (2022), 47–52. https://doi.org/10.1016/j.comcom.2022.06.036

[17] Andres Garcia-Saavedra and Xavier Costa-Pérez. 2021. O-RAN: Disrupting the Virtualized RAN Ecosystem. *IEEE Communications Standards Magazine* 5, 4 (2021), 96–103. https://doi.org/10.1109/MCOMSTD.101.2000014

[18] Changhan Ge, Zihui Ge, Xuan Liu, Ajay Mahimkar, Yusef Shaqalle, Yu Xiang, and Shomik Pathak. 2023. Chroma: Learning and Using Network Contexts to Reinforce Performance Improving Configurations. In *Proc. of ACM MobiCom.* https://doi.org/10.1145/3570361.3613256

[19] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. 2018. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)* 51, 5 (2018), 1–42.

[20] David Gunning and David Aha. 2019. DARPA's explainable artificial intelligence (XAI) program. *AI magazine* 40, 2 (2019), 44–58.

[21] Delin Guo, Lan Tang, Xinggan Zhang, and Ying-Chang Liang. 2020. Joint Optimization of Handover Control and Power Allocation Based on Multi-Agent Deep Reinforcement Learning. *IEEE Transactions on Vehicular Technology* 69, 11 (2020), 13124–13138. https://doi.org/10.1109/TVT.2020.3020400

[22] Weisi Guo. 2020. Explainable Artificial Intelligence for 6G: Improving Trust between Human and Machine. *IEEE Communications Magazine* 58, 6 (2020), 39–45. https://doi.org/10.1109/MCOM.001.2000050

[23] Pouya Hamadanian, Malte Schwarzkopf, Siddartha Sen, and Mohammad Alizadeh. 2022. How Reinforcement Learning Systems Fail and What to do About It. In *Proc. of EuroMLSys.*

[24] Dongqi Han, Zhiliang Wang, Wenqi Chen, Ying Zhong, Su Wang, Han Zhang, Jiahai Yang, Xingang Shi, and Xia Yin. 2021. DeepAID: Interpreting and Improving Deep Learning-Based Anomaly Detection in Security Applications. In *Proc. of ACM SIGSAC CCS.* 3197–3217. https://doi.org/10.1145/3460120.3484589

[25] Lei He, Nabil Aouf, and Bifeng Song. 2021. Explainable Deep Reinforcement Learning for UAV autonomous path planning. *Aerospace Science and Technology* 118 (2021), 107052. https://doi.org/10.1016/j.ast.2021.107052

[26] Alexandre Heuillet, Fabien Couthouis, and Natalia Díaz-Rodríguez. 2022. Collective EXplainable AI: Explaining Cooperative Strategies and Agent Contribution in Multiagent Reinforcement Learning With Shapley Values. *IEEE Comp. Intell. Mag.* 17, 1 (Feb 2022), 59–71. https://doi.org/10.1109/MCI.2021.3129959

[27] Tai Manh Ho and Kim-Khoa Nguyen. 2022. Joint Server Selection, Cooperative Offloading and Handover in Multi-Access Edge Computing Wireless Network: A Deep Reinforcement Learning Approach. *IEEE Transactions on Mobile Computing* 21, 7 (July 2022), 2421–2435.

[28] Chih-Wei Huang, Ibrahim Althamary, Yen-Cheng Chou, Hong-Yunn Chen, and Cheng-Fu Chou. 2023. A DRL-Based Automated Algorithm Selection Framework for Cross-Layer QoS-Aware Scheduling and Antenna Allocation in Massive MIMO Systems. *IEEE Access* 11 (2023), 13243–13256. https://doi.org/10.1109/ACCESS.2023.3243068

[29] Intel. 2022. Intelligent Connection Management for Automated Handover Reference Implementation. https://www.intel.com/content/www/us/en/developer/articles/reference-implementation/intelligent-connection-management.html.

[30] Pedro Enrique Iturria-Rivera, Han Zhang, Hao Zhou, Shahram Mollahasani, and Melike Erol-Kantarci. 2022. Multi-agent team learning in virtualized open radio access networks (O-RAN). *Sensors* 22, 14 (2022), 5375.

[31] Arthur S. Jacobs, Roman Beltiukov, Walter Willinger, Ronaldo A. Ferreira, Arpit Gupta, and Lisandro Z. Granville. 2022. AI/ML for Network Security: The Emperor Has No Clothes. In *Proc. of ACM SIGSAC CCS.* 1537–1551. https://doi.org/10.1145/3548606.3560609

[32] David Johnson, Dustin Maas, and Jacobus Van Der Merwe. 2022. NexRAN: Closed-Loop RAN Slicing in POWDER - A Top-to-Bottom Open-Source Open-RAN Use Case. In *Proc. of ACM WiNTECH.* 17–23. https://doi.org/10.1145/3477086.3480842

[33] Yafim Kazak, Clark Barrett, Guy Katz, and Michael Schapira. 2019. Verifying Deep-RL-Driven Systems. In *Proc. of ACM NetAI.* 83–89. https://doi.org/10.1145/3341216.3342218

[34] Bettina Könighofer, Florian Lorber, Nils Jansen, and Roderick Bloem. 2020. Shield Synthesis for Reinforcement Learning. In *Leveraging Applications of Formal Methods, Verification and Validation: Verification Principles*, Tiziana Margaria and Bernhard Steffen (Eds.). Springer International Publishing, 290–306.

[35] M. Korobov and K. Lopuhin. 2021. ELI5 is a Python library - v. 0.11. Available at (accessed 26/10/2021): https://eli5.readthedocs.io/en/latest/.

[36] Agneza Krajna, Mario Brcic, Tomislav Lipic, and Juraj Doncevic. 2022. Explainability in reinforcement learning: perspective and position. *arXiv preprint arXiv:2203.11547* (2022).

[37] Lawrence Berkeley National Laboratory, United States. Department of Energy. Office of Scientific, and Technical Information. 2013. *Attributed Graph Models: Modeling Network Structure with Correlated Attributes.* United States. Department of Energy.

[38] Sascha Lange and Martin Riedmiller. 2010. Deep auto-encoder neural networks in reinforcement learning. In *International Joint Conference on Neural Networks (IJCNN).* 1–8. https://doi.org/10.1109/IJCNN.2010.5596468

[39] Aris Leivadeas and Matthias Falkner. 2023. A Survey on Intent-Based Networking. *IEEE Communications Surveys & Tutorials* 25, 1 (2023), 625–655. https://doi.org/10.1109/COMST.2022.3215919

[40] Chen Li, Weisi Guo, Schyler Chengyao Sun, Saba Al-Rubaye, and Antonios Tsourdos. 2020. Trustworthy Deep Learning in 6G-Enabled Mass Autonomy: From Concept to Quality-of-Trust Key Performance Indicators. *IEEE Vehicular Technology Magazine* 15, 4 (2020), 112–121. https://doi.org/10.1109/MVT.2020.3017181

[41] Jiahui Li, Kun Kuang, Baoxiang Wang, Furui Liu, Long Chen, Fei Wu, and Jun Xiao. 2021. Shapley Counterfactual Credits for Multi-Agent Reinforcement Learning. In *Proc. of ACM SIGKDD.* 934–942. https://doi.org/10.1145/3447548.3467420

[42] Peizheng Li, Jonathan Thomas, Xiaoyang Wang, Ahmed Khalil, Abdelrahim Ahmad, Rui Inacio, Shipra Kapoor, Arjun Parekh, Angela Doufexi, Arman Shojaeifard, et al. 2021. RLOps: Development Life-cycle of Reinforcement Learning Aided Open RAN. *arXiv preprint* (2021).

[43] Yihong Li, Tianyu Zeng, Xiaoxi Zhang, Jingpu Duan, and Chuan Wu. 2023. TapFinger: Task Placement and Fine-Grained Resource Allocation for Edge Machine Learning. In *Proc. of IEEE INFOCOM.* 1–10. https://doi.org/10.1109/INFOCOM53939.2023.10229031

[44] Eric Liang, Hang Zhu, Xin Jin, and Ion Stoica. 2019. Neural packet classification. In *Proc. of ACM SIGCOMM.* 256–269.

[45] Scott M Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. 2020. From local explanations to global understanding with explainable AI for trees. *Nature machine intelligence* 2, 1 (2020), 56–67.

[46] Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *Proc. of NIPS.* 4768–4777.

[47] Nguyen Cong Luong, Dinh Thai Hoang, Shimin Gong, Dusit Niyato, Ping Wang, Ying-Chang Liang, and Dong In Kim. 2019. Applications of Deep Reinforcement Learning in Communications and Networking: A Survey. *IEEE Communications Surveys & Tutorials* 21, 4 (2019), 3133–3174. https://doi.org/10.1109/COMST.2019.2916583

[48] Prashan Madumal, Tim Miller, Liz Sonenberg, and Frank Vetere. 2020. Explainable Reinforcement Learning through a Causal Lens. *Proc. of AAAI Conference on Artificial Intelligence* 34, 03 (Apr. 2020), 2493–2500. https://doi.org/10.1609/aaai.v34i03.5631

[49] Ajay Mahimkar, Ashiwan Sivakumar, Zihui Ge, Shomik Pathak, and Karunasish Biswas. 2021. Auric: Using Data-Driven Recommendation to Automatically Generate Cellular Configuration. In *Proc. of the ACM SIGCOMM*. 807–820. https://doi.org/10.1145/3452296.3472906

[50] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural Adaptive Video Streaming with Pensieve. In *Proc. of ACM SIGCOMM*. 197–210. https://doi.org/10.1145/3098822.3098843

[51] Mavenir. 2023. Building the World's First O-RAN-Compliant, AI-Powered, Closed-Loop Near-RT RIC. https://www.mavenir.com/blog/building-the-worlds-first-o-ran-compliant-ai-powered-closed-loop-near-rt-ric/.

[52] Tommaso Melodia, Stefano Basagni, Kaushik R. Chowdhury, Abhimanyu Gosain, Michele Polese, Pedram Johari, and Leonardo Bonati. 2021. Colosseum, the World's Largest Wireless Network Emulator. In *Proc. of ACM MobiCom*. 860–861. https://doi.org/10.1145/3447993.3488032

[53] Zili Meng, Minhu Wang, Jiasong Bai, Mingwei Xu, Hongzi Mao, and Hongxin Hu. 2020. Interpreting Deep Learning-Based Networking Systems. In *Proc. of ACM SIGCOMM*. 154–171. https://doi.org/10.1145/3387514.3405859

[54] Stuart E. Middleton, Emmanuel Letouzé, Ali Hossaini, and Adriane Chapman. 2022. Trust, Regulation, and Human-in-the-Loop AI: Within the European Region. *Commun. ACM* 65, 4 (mar 2022), 64–68.

[55] Grégoire Montavon, Alexander Binder, Sebastian Lapuschkin, Wojciech Samek, and Klaus-Robert Müller. 2019. *Layer-Wise Relevance Propagation: An Overview.* Springer International Publishing, 193–209. https://doi.org/10.1007/978-3-030-28954-6_10

[56] Mohamed Moulay, Rafael Garcia Leiva, Pablo J Rojo Maroni, Fernando Diez, Vincenzo Mancuso, and Antonio Fernandez Anta. 2022. Automated identification of network anomalies and their causes with interpretable machine learning: The CIAN methodology and TTrees implementation. *Computer Communications* 191 (2022), 327–348.

[57] Usama Naseer and Theophilus A Benson. 2022. Configanator: A Data-driven Approach to Improving CDN Performance. In *Proc. of USENIX NSDI*. 1135–1158.

[58] O-RAN Software Community. [n. d.]. Amber Release. https://wiki.o-ran-sc.org/pages/viewpage.action?pageId=14221337. Accessed July 2020.

[59] O-RAN Working Group 3. 2021. O-RAN Near-RT RAN Intelligent Controller Near-RT RIC Architecture 2.00. O-RAN.WG3.RICARCH-v02.00.

[60] Michele Polese, Leonardo Bonati, Salvatore D'Oro, Stefano Basagni, and Tommaso Melodia. 2022. ColO-RAN: Developing Machine Learning-based xApps for Open RAN Closed-loop Control on Programmable Experimental Platforms. *IEEE Transactions on Mobile Computing* (2022), 1–14. https://doi.org/10.1109/TMC.2022.3188013

[61] Michele Polese, Leonardo Bonati, Salvatore D'Oro, Stefano Basagni, and Tommaso Melodia. 2023. Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges. *IEEE Communications Surveys & Tutorials* (2023), 1–1. https://doi.org/10.1109/COMST.2023.3239220

[62] Bharat Prakash, Mark Horton, Nicholas R. Waytowich, William David Hairston, Tim Oates, and Tinoosh Mohsenin. 2019. On the Use of Deep Autoencoders for Efficient Embedded Reinforcement Learning. In *Proc. of ACM GLSVLSI*. 507–512. https://doi.org/10.1145/3299874.3319493

[63] Erika Puiutta and Eric M. S. P. Veith. 2020. Explainable Reinforcement Learning: A Survey. In *Machine Learning and Knowledge Extraction*, Andreas Holzinger, Peter Kieseberg, A Min Tjoa, and Edgar Weippl (Eds.). Springer International Publishing, 77–95.

[64] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proc. of ACM SIGKDD*. 1135–1144. https://doi.org/10.1145/2939672.2939778

[65] Daniele Scapin, Giulia Cisotto, Elvina Gindullina, and Leonardo Badia. 2022. Shapley Value as an Aid to Biomedical Machine Learning: a Heart Disease Dataset Analysis. In *Proc. of IEEE CCGrid*. 933–939. https://doi.org/10.1109/CCGrid54584.2022.00113

[66] Moghadas Gholian Serly, Fiandrino Claudio, Collet Alan, Attanasio Giulia, Fiore Marco, and Widmer Joerg. 2023. Spotting Deep Neural Network Vulnerabilities in Mobile Traffic Forecasting with an Explainable AI Lens. In *Proc. of IEEE INFOCOM*. 1–10. https://doi.org/10.1109/INFOCOM53939.2023.10228989

[67] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. 2017. Learning important features through propagating activation differences. In *Proc. of ICML*. 3145–3153.

[68] Rob Smith, Connor Freeberg, Travis Machacek, and Venkatesh Ramaswamy. 2021. An O-RAN Approach to Spectrum Sharing Between Commercial 5G and Government Satellite Systems. In *Proc. of IEEE MILCOM*. 739–744. https://doi.org/10.1109/MILCOM52596.2021.9653140

[69] Richard S Sutton and Andrew G Barto. 2018. Reinforcement learning: An introduction second edition. *Adaptive computation and machine learning: The MIT Press, Cambridge, MA and London* (2018).

[70] Unwired Labs. Accessed March 2023. OpenCelliD. https://opencellid.org.

[71] U.S. Naval Research Laboratory. [n. d.]. MGEN Traffic Emulator. https://www.nrl.navy.mil/Our-Work/Areas-of-Research/Information-Technology/NCS/MGEN. Accessed September 2021.

[72] Shen Wang, M. Atif Qureshi, Luis Miralles-Pechuán, Thien Huynh-The, Thippa Reddy Gadekallu, and Madhusanka Liyanage. 2021. Explainable AI for B5G/6G: Technical Aspects, Use Cases, and Research Challenges. https://doi.org/10.48550/ARXIV.2112.04698

[73] Rui Xu, Siyu Luan, Zonghua Gu, Qingling Zhao, and Gang Chen. 2022. LRP-based Policy Pruning and Distillation of Reinforcement Learning Agents for Embedded Systems. In *Proc. of IEEE ISORC*. 1–8. https://doi.org/10.1109/ISORC52572.2022.9812837

[74] Zhiyuan Xu, Jian Tang, Jingsong Meng, Weiyi Zhang, Yanzhi Wang, Chi Harold Liu, and Dejun Yang. 2018. Experience-driven Networking: A Deep Reinforcement Learning based Approach. In *Proc. of IEEE INFOCOM*. 1871–1879. https://doi.org/10.1109/INFOCOM.2018.8485853

[75] Hao Yu, Tarik Taleb, and Jiawei Zhang. 2022. Deep Reinforcement Learning based Deterministic Routing and Scheduling for Mixed-Criticality Flows. *IEEE Transactions on Industrial Informatics* (2022), 1–11. https://doi.org/10.1109/TII.2022.3222314

[76] Hongliang Zhang, Lingyang Song, Yonghui Li, and Geoffrey Ye Li. 2017. Hypergraph Theory: Applications in 5G Heterogeneous Ultra-Dense Networks. *IEEE Communications Magazine* 55, 12 (2017), 70–76. https://doi.org/10.1109/MCOM.2017.1700400

[77] Ke Zhang, Jun Zhang, Pei-Dong Xu, Tianlu Gao, and David Wenzhong Gao. 2022. Explainable AI in Deep Reinforcement Learning Models for Power System Emergency Control. *IEEE Transactions on Computational Social Systems* 9, 2 (2022), 419–427. https://doi.org/10.1109/TCSS.2021.3096824

[78] Tianhang Zheng and Baochun Li. 2022. Poisoning Attacks on Deep Learning based Wireless Traffic Prediction. In *Proc. of IEEE INFOCOM*. 660–669. https://doi.org/10.1109/INFOCOM48880.2022.9796791

[79] Ying Zheng, Lixiang Lin, Tianqi Zhang, Haoyu Chen, Qingyang Duan, Yuedong Xu, and Xin Wang. 2022. Enabling Robust DRL-Driven Networking Systems via Teacher-Student Learning. *IEEE Journal on Selected Areas in Communications* 40, 1 (2022), 376–392. https://doi.org/10.1109/JSAC.2021.3126085

[80] Ying Zheng, Ziyu Liu, Xinyu You, Yuedong Xu, and Junchen Jiang. 2018. Demystifying deep learning in networking. In *Proc. of APNet*. 1–7.

A EXPERIMENTAL CONFIGURATIONS

Table 3. The configurations utilized in the experiments

Agent	Traf.	Num. Users						Action Steering Strategy (Users: 6, drop to 5)			
	Scen.	6	5	4	3	2	1	*AR 1*	*AR 2*	*AR 3*	Baseline
HT	TRF1	$C_{HT,trf1-6}$	$C_{HT,trf1-5}$	$C_{HT,trf1-4}$	$C_{HT,trf1-3}$	$C_{HT,trf1-2}$	$C_{HT,trf1-1}$	$C_{HT,trf1\text{-}a1\text{-}10}$, $C_{HT,trf1\text{-}a1\text{-}20}$	$C_{HT,trf1\text{-}a2\text{-}10}$, $C_{HT,trf1\text{-}a2\text{-}20}$	$C_{HT,trf1\text{-}a3\text{-}10}$, $C_{HT,trf1\text{-}a3\text{-}20}$	$C_{HT,trf1\text{-}b\text{-}10}$, $C_{HT,trf1\text{-}b\text{-}20}$
	TRF2	$C_{HT,trf2-6}$	$C_{HT,trf2-5}$	$C_{HT,trf2-4}$	$C_{HT,trf2-3}$	$C_{HT,trf2-2}$	$C_{HT,trf2-1}$	$C_{HT,trf2\text{-}a1\text{-}10}$, $C_{HT,trf2\text{-}a1\text{-}20}$	$C_{HT,trf2\text{-}a2\text{-}10}$, $C_{HT,trf2\text{-}a2\text{-}20}$	$C_{HT,trf2\text{-}a3\text{-}10}$, $C_{HT,trf2\text{-}a3\text{-}20}$	$C_{HT,trf2\text{-}b\text{-}10}$, $C_{HT,trf2\text{-}b\text{-}20}$
LL	TRF1	$C_{LL,trf1-6}$	$C_{LL,trf1-5}$	$C_{LL,trf1-4}$	$C_{LL,trf1-3}$	$C_{LL,trf1-2}$	$C_{LL,trf1-1}$	$C_{LL,trf1\text{-}a1\text{-}10}$, $C_{LL,trf1\text{-}a1\text{-}20}$	$C_{LL,trf1\text{-}a2\text{-}10}$, $C_{LL,trf1\text{-}a2\text{-}20}$	$C_{LL,trf1\text{-}a3\text{-}10}$, $C_{LL,trf1\text{-}a3\text{-}20}$	$C_{LL,trf1\text{-}b\text{-}10}$, $C_{LL,trf1\text{-}b\text{-}20}$
	TRF2	$C_{LL,trf2-6}$	$C_{LL,trf2-5}$	$C_{LL,trf2-4}$	$C_{LL,trf2-3}$	$C_{LL,trf2-2}$	$C_{LL,trf2-1}$	$C_{LL,trf2\text{-}a1\text{-}10}$, $C_{LL,trf2\text{-}a1\text{-}20}$	$C_{LL,trf2\text{-}a2\text{-}10}$, $C_{LL,trf2\text{-}a2\text{-}20}$	$C_{LL,trf2\text{-}a3\text{-}10}$, $C_{LL,trf2\text{-}a3\text{-}20}$	$C_{LL,trf2\text{-}b\text{-}10}$, $C_{LL,trf2\text{-}b\text{-}20}$

Table 3 shows the configurations C used in the experiments (§ 3.2 and § 6) for different agents and different traffic scenarios. Overall, we run 48 different experiments. Each configuration in the left part of the table refers to experiments with different numbers of users, *e.g.*, the tuple "HT,tr1-6" indicates a configuration C with the HT agent, traffic profile 1, and 6 users. Users are equally assigned to each slice in the experiments with 6 and 3 users. For the experiments with 5, 4 and 2 users we use the following assignment:

- 5 users: 2 users to the slice eMBB, 1 user to the slice mMTC and 2 users to the slice URLLC;
- 4 users: 1 user to the slice eMBB, 1 user to the slice mMTC and 2 users to the slice URLLC;
- 2 users: 1 user to the slice eMBB, no users to the slice mMTC and 1 user to the slice URLLC.

The experiments with 1 user are executed three times, one for each of the the three slices eMBB, mMTC, URLLC. The right part of the table refe5rs to the experiments with intent-based action steering strategies (discussed in § 6.3), *e.g.*, the tuple "HT,trf1-a1-10" denotes a configuration C with the HT agent, traffic profile 1, policy replacement *AR 1*, and an observation window O of 10 entries.

B GENERATING ATTRIBUTED GRAPHS

With the help of an example, we now explain how *EXPLORA* builds the attributed graph for the case of the agent HT and traffic scenario TRF1. We first provide a step-by-step explanation and then show the overall graph.

In Figure 11, we show both nodes (*i.e.*, the actions) and attributes during three consecutive time steps t_0, t_1, and t_2 in the observation window W. Two users per slice are present in the system for a total of six users. Each node in the graph contains a multi-modal action. The node ([36, 3, 11], [2, 0, 1]) represents the RAN slicing PRB allocation on the left, *i.e.*, [36, 3, 11] and scheduling policy allocation on the right, *i.e.*, [2, 0, 1] (0 denotes Round Robin (RR), 1 - Waterfilling (WF), and 2 - Proportional Fair (PF)). The positions inside the array refer to the slices. For example, [36, 3, 11] denotes an allocation of 36 PRB to the slice eMBB, 3 to the slice mMTC, and 11 to the slice URLLC. Each attribute in the graph contains the distribution of KPIs per slice of each user. For example, SL0 [225,234] of the attribute `tx_packets` indicates that two users have transmitted 225 and 234 packets for the slice 0. At time t_0, the agent enforces the action ([36, 3, 11], [1, 2, 2]). We record in three different attributes, one per KPI (i.e., `tx_bitrate`, `tx_packets`, and `DWL_buffer_size`), the corresponding data observation for each slice (i.e., SL0, SL1, and SL2) and for each user. These attributes describe the impact of the action taken at t_0 on the future state $t_{0+\delta}$ (with $\delta = 250$ ms) and that is taken as input by the DRL framework (autoencoder with DRL agent) to take action ([36, 3, 11], [2, 0, 1]) at t_1. As this is a new action, *EXPLORA* adds a new node, monitors the future state $t_{1+\delta}$ and records it in form of attributes. In the next iteration at t_2, the agent takes again the action ([36, 3, 11], [1, 2, 2]). This is not a new action, hence *EXPLORA* updates the existing attributes recorded at $t_{0+\delta}$ with the new attributed at $t_{2+\delta}$. Throughout W, the attributes build in form of a distribution that contains as many samples as the occurrences of the action. The analysis of the distributions for each KPI and slice between transitions allows to reveal the rationale for which the agent changes action. For example, in the transition from $t_1 \longrightarrow t_2$, the `tx_bitrate` for SL0 increases. Finally, in Figure 12, we show the complete graph and we only represent nodes and not attributes for readability.

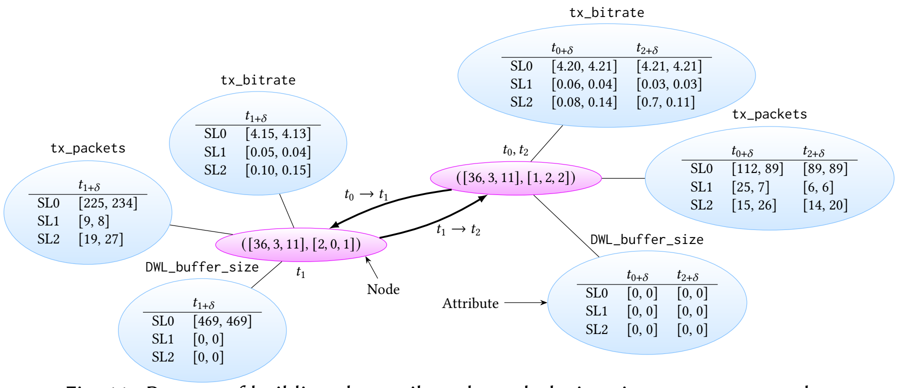


Fig. 11. Process of building the attributed graph during time steps t_0, t_1 and t_2

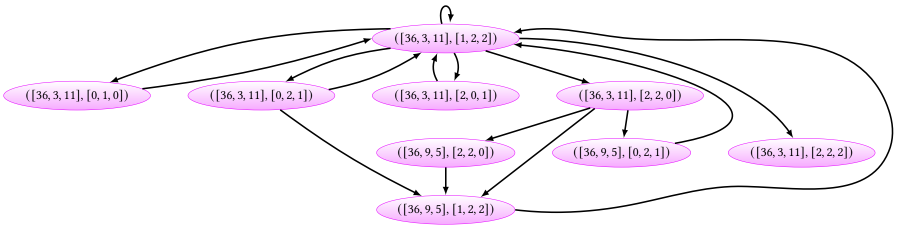
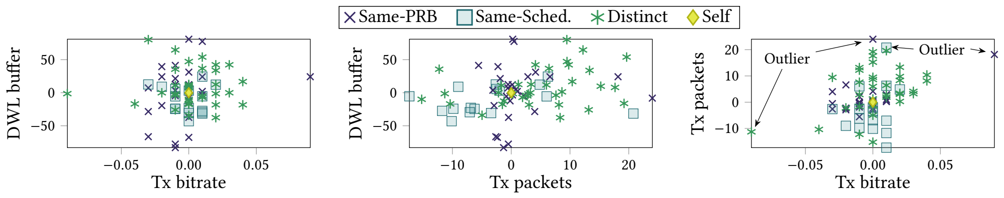


Fig. 12. The resulting graph for the HT agent utilized in the presence of TRF1 traffic with a focus on actions



(a) tx_bitrate and DWL_buffer_size (b) tx_packets and DWL_buffer_size (c) tx_bitrate and tx_packets

Fig. 13. Detailed explanations for the LL agent's behavior

C CONSIDERATIONS ON THE AGENT LL

Figure 13, Figure 14, and Table 4 show the process to obtain the summary of explanations for the agent LL similarly to what Figure 7, Figure 8, and Table 2 represent for the agent HT. With respect to the HT counterpart, the agent LL presents significant differences.

• First, the number and magnitude of KPI variations (for measurement units, refer to § 3.1) is, as expected, lower for the LL agent than for the HT agent. Indeed, the latter agent strives to guarantee high throughput for traffic flows with higher volume. The flows of the URLLC slice exhibit much lower volume. Maximizing throughput is comparatively easier than striving to minimize the buffer size (as proxy to low latency). This phenomenon can be observed in the higher number of transitions that the LL agent performs if compared to its HT counterpart.

• Second, note that the LL agent shows a few outliers. These are transitions that generate an unexpected explanation. For example, in Figure 13(c) the "Same-Sched." transition denoted as outlier is unexpected as all other transitions of such category usually lead to a decrease of tx_packets. We attribute this behavior to the fact that while traffic is deterministic in terms of statistical behavior, the actual instantaneous traffic load may deviate from the trend and thus produce outliers.

• Third, unlike the HT agent that mainly uses the "Same-PRB" class (40%), the LL agent uses the two classes evenly.

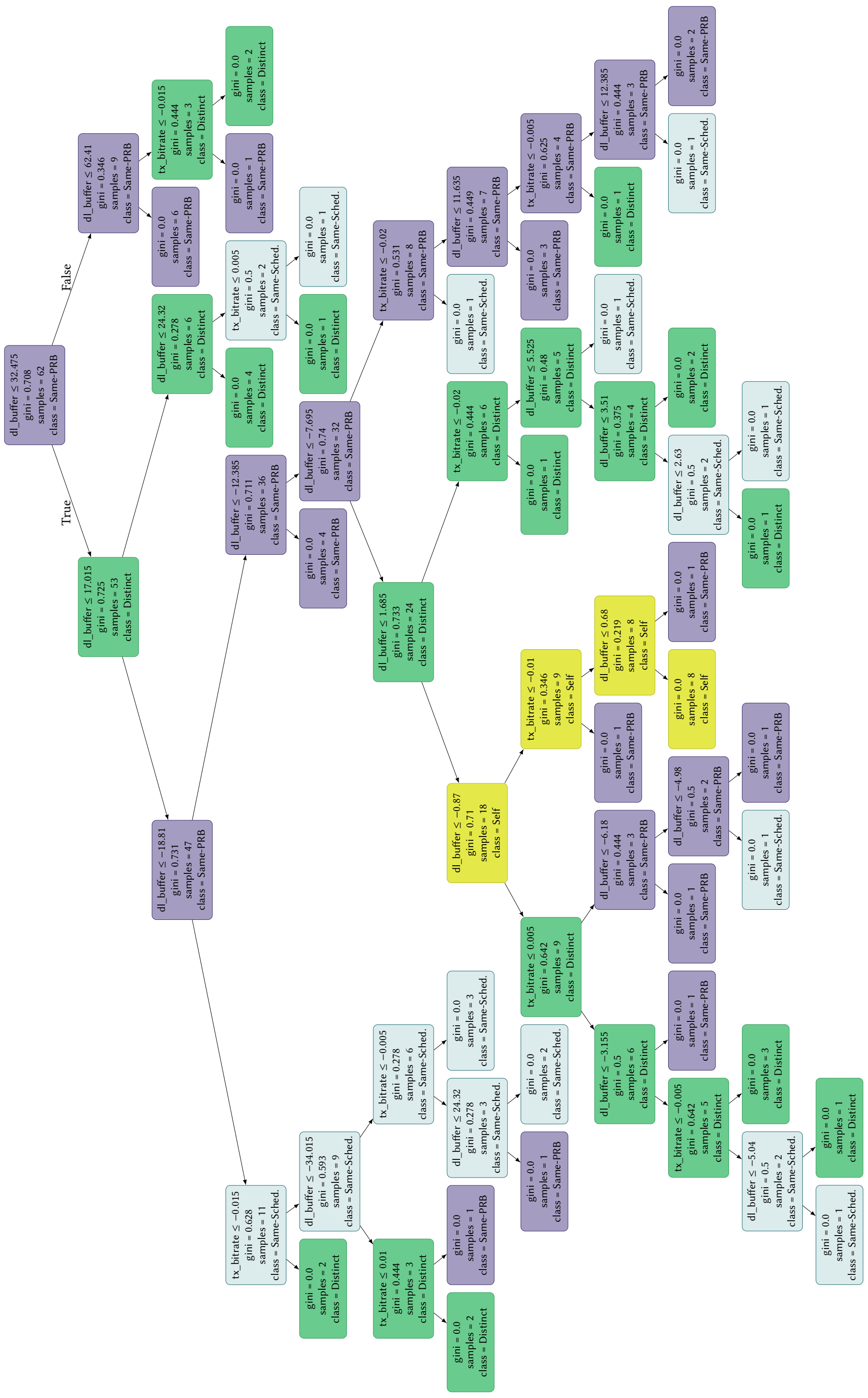

dl_buffer ≤ 32.475 gini = 0.708 samples = 62 class = Same-PRB
True
False
dl_buffer ≤ 62.41 gini = 0.346 samples = 9 class = Same-PRB
gini = 0.0 samples = 6 class = Same-PRB
tx_bitrate ≤ −0.015 gini = 0.444 samples = 3 class = Distinct
gini = 0.0 samples = 1 class = Same-PRB
gini = 0.0 samples = 2 class = Distinct
dl_buffer ≤ 17.015 gini = 0.725 samples = 53 class = Distinct
dl_buffer ≤ 24.32 gini = 0.278 samples = 6 class = Distinct
gini = 0.0 samples = 4 class = Distinct
tx_bitrate ≤ 0.005 gini = 0.5 samples = 2 class = Same-Sched.
gini = 0.0 samples = 1 class = Distinct
gini = 0.0 samples = 1 class = Same-Sched.
dl_buffer ≤ −18.81 gini = 0.731 samples = 47 class = Same-PRB
dl_buffer ≤ −12.385 gini = 0.711 samples = 36 class = Same-PRB
gini = 0.0 samples = 4 class = Same-PRB
dl_buffer ≤ −7.695 gini = 0.74 samples = 32 class = Same-PRB
tx_bitrate ≤ −0.02 gini = 0.531 samples = 8 class = Same-PRB
gini = 0.0 samples = 1 class = Same-Sched.
dl_buffer ≤ 11.635 gini = 0.449 samples = 7 class = Same-PRB
gini = 0.0 samples = 3 class = Same-PRB
tx_bitrate ≤ −0.005 gini = 0.625 samples = 4 class = Same-PRB
gini = 0.0 samples = 1 class = Distinct
dl_buffer ≤ 12.385 gini = 0.444 samples = 3 class = Same-PRB
gini = 0.0 samples = 1 class = Same-Sched.
gini = 0.0 samples = 2 class = Same-PRB
dl_buffer ≤ 1.685 gini = 0.733 samples = 24 class = Distinct
tx_bitrate ≤ −0.02 gini = 0.444 samples = 6 class = Distinct
gini = 0.0 samples = 1 class = Distinct
dl_buffer ≤ 5.525 gini = 0.48 samples = 5 class = Distinct
gini = 0.0 samples = 1 class = Same-Sched.
dl_buffer ≤ 3.51 gini = 0.375 samples = 4 class = Distinct
gini = 0.0 samples = 2 class = Distinct
dl_buffer ≤ 2.63 gini = 0.5 samples = 2 class = Same-Sched.
gini = 0.0 samples = 1 class = Same-Sched.
gini = 0.0 samples = 1 class = Distinct
dl_buffer ≤ −0.87 gini = 0.71 samples = 18 class = Self
tx_bitrate ≤ −0.01 gini = 0.346 samples = 9 class = Self
gini = 0.0 samples = 1 class = Same-PRB
dl_buffer ≤ 0.68 gini = 0.219 samples = 8 class = Self
gini = 0.0 samples = 1 class = Same-PRB
gini = 0.0 samples = 8 class = Self
tx_bitrate ≤ 0.005 gini = 0.642 samples = 9 class = Distinct
dl_buffer ≤ −6.18 gini = 0.444 samples = 3 class = Same-PRB
dl_buffer ≤ −4.98 gini = 0.5 samples = 2 class = Same-PRB
gini = 0.0 samples = 1 class = Same-PRB
gini = 0.0 samples = 1 class = Same-Sched.
gini = 0.0 samples = 1 class = Same-PRB
dl_buffer ≤ −3.155 gini = 0.5 samples = 6 class = Distinct
gini = 0.0 samples = 1 class = Same-PRB
tx_bitrate ≤ −0.005 gini = 0.642 samples = 5 class = Distinct
gini = 0.0 samples = 3 class = Distinct
dl_buffer ≤ −5.04 gini = 0.5 samples = 2 class = Same-Sched.
gini = 0.0 samples = 1 class = Distinct
gini = 0.0 samples = 1 class = Same-Sched.
tx_bitrate ≤ −0.015 gini = 0.628 samples = 11 class = Same-Sched.
dl_buffer ≤ −34.015 gini = 0.593 samples = 9 class = Same-Sched.
tx_bitrate ≤ −0.005 gini = 0.278 samples = 6 class = Same-Sched.
gini = 0.0 samples = 3 class = Same-Sched.
dl_buffer ≤ 24.32 gini = 0.278 samples = 3 class = Same-Sched.
gini = 0.0 samples = 2 class = Same-Sched.
gini = 0.0 samples = 1 class = Same-PRB
tx_bitrate ≤ 0.01 gini = 0.444 samples = 3 class = Distinct
gini = 0.0 samples = 1 class = Same-PRB
gini = 0.0 samples = 2 class = Distinct
gini = 0.0 samples = 2 class = Distinct

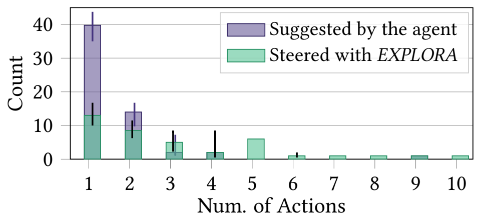
Fig. 14. DT on *EXPLORA* explanations for the LL agent

Table 4. Summary of explanations for the LL agent

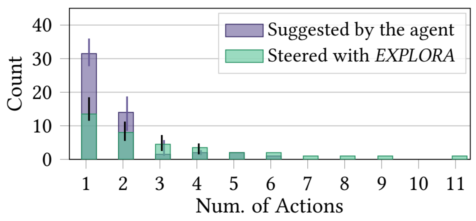
TRANSITION	INTERPRETATION
SAME-PRB	Diminishes lightly `tx_bitrate` and augments `tx_packets`
SAME-SCHED.	Diminishes `DWL_buffer_size`, usually diminishes `tx_packets` and seldom marginally augments `tx_packets`
DISTINCT	Mainly augments `tx_packets`
SELF	No change in KPIs

D FURTHER CONSIDERATIONS ON ACTION STEERING

We now assert that the intent-based action steering policies defined in § 5.2 do not harm the capabilities of the agent to generalize and adapt to changes in the systems. Figure 15 portrays the median, first and third quartiles of the distributions obtained across all configuration settings for *AR 1* (*i.e.*, HT and LL agents, TRF1 and TRF2 traffic scenarios) and varying the size of the past history O. We report both the number of times the attributed graph G "suggests" to replace an action (purple bars), as well as the number of times such action is actually being replaced with the one suggested by the graph (green bars). Lower values of O trigger comparatively a slightly higher number of action changes than higher values of O (on average, 63% and 59%, respectively). However, through action changes, the agent probes less often new actions in a fully controlled fashion (the reduction between potentially used actions and those actually used is 25% for $O = 10$ and 18% for $O = 20$). The important remark from Figure 15 is that our intent-based action steering strategies are not preventing the agent to take a specific action (like shielding would do) because it is rare that the same action gets substituted more than 3 times. On the contrary, leveraging the explanations, the same actions suggested by the graph as replacement are used more often.



(a) $O = 10$



(b) $O = 20$

Fig. 15. Distribution of actions with *AR 1*

E ETHICAL CONSIDERATIONS

This work does not raise any ethical issues as per the ACM Code of Ethics.