AdvO-RAN: Adversarial Deep Reinforcement Learning in AI-Driven Open Radio Access Networks

Tanzil Bin Hassan, Francesca Meneghello and Francesco Restuccia
Northeastern University, United States

Abstract

While artificial intelligence (AI) is improving the performance of O-RAN, it will also expose the network to adversarial machine learning (AML) attacks. For this reason, in this paper, we are the first to investigate AML in the context of deep reinforcement learning (DRL)-based O-RAN xApps. What separates AML in O-RAN from traditional settings is the need to design and analyze adversarial attacks based on *RAN-specific* Key Performance Measures (KPMs) such as transmitted bit rate, downlink buffer occupancy, transmitted packets, etc. As such, we propose the AdvO-RAN framework, which includes (i) a new adversarial perturbation generator using preference-based reinforcement learning (PbRL) to learn the perturbation that most violate the user service level agreements (SLA) and (ii) a robust training module for enhancing DRL agent resilience to the attacks in (i). We experimentally evaluate AdvO-RAN on the Colosseum network emulator. Experimental results show that AdvO-RAN can enhance xApp performance by reducing SLA violations from 44% to 27% on average and reducing by 46% the latency under the most challenging attack scenario for Ultra-Reliable Low-Latency Communications (URLLC) traffic. AdvO-RAN can improve up to 75% of throughput in the victim Enhanced Mobile Broadband (eMBB) slice users during a constant bit-rate traffic scenario.

CCS Concepts

• **Computing methodologies** → **Machine learning**; • **Networks** → **Network performance evaluation**.

ACM Reference Format:
Tanzil Bin Hassan, Francesca Meneghello and Francesco Restuccia. 2025. AdvO-RAN: Adversarial Deep Reinforcement Learning in AI-Driven Open Radio Access Networks. In *International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing (MobiHoc '25), October 27–30, 2025, Houston, TX, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3704413.3764425

1 Introduction and Motivation

The O-RAN (Open Radio Access Network) paradigm divides the RAN into interoperable components [1], where Extensible Applications (xApps) deployed inside the Near Real-Time RAN Intelligent Controller (Near-RT-RIC) provide AI-driven techniques to control critical functionalities such as network slicing, scheduling, traffic steering, and interference mitigation [2, 3]. The deployment of AI-driven xApps inevitably opens the door to AML. Figure 1 illustrates the system and threat model we consider in this paper. Specifically, O-RAN transmits sensitive data about network monitoring through the E2 interface, which connects the Near-RT-RIC with each of the Base Stations (BSs) in the network to generate near-real-time network control decisions [4]. Notably, due to the stringent latency requirements of the Near-RT-RIC operations, O-RAN specifications do not encrypt open interfaces such as the E2 [5], which allows attackers to spoof traffic exchange and use the information to design the attack. These attacks are facilitated by the fact that xApps will be shared in third-party xApp marketplaces. This allows attackers to retrieve the AI/ML algorithms embedded in xApps and fine-tune their attack strategy accordingly.

Prior work has shown that DRL is effective in jointly allocating radio resources and scheduling users in wireless network [6]. However, AML- related vulnerabilities in DRL-based xApps have been studied only superficially [7–9]. Specifically, we look beyond the Radio Access Network (RAN)-level Key Performance Measures (KPMs) and consider user-level Quality of Service (QoS) metrics at the application layer. What is still unclear is how effective AML can be in scenarios characterized by different applications with diverse QoS requirements at the application level. Optimizing the network solely on RAN-level metrics is challenging. In addition, incorporating diverse and often unpredictable application-layer demands increases such complexity [10].

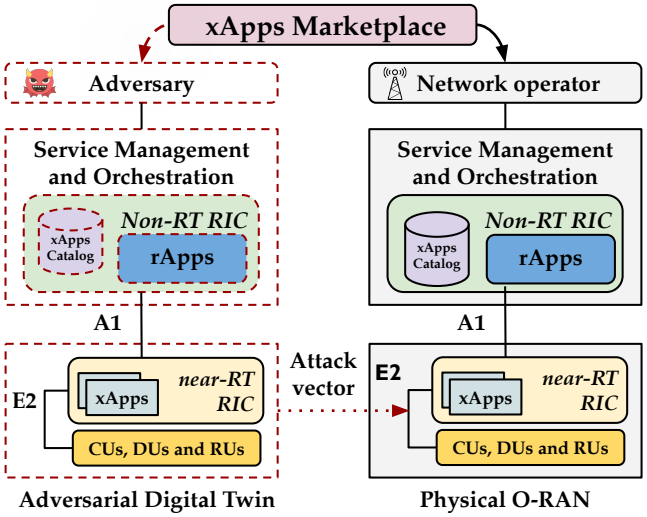


Figure 1: Adversarial machine learning in O-RANs.

Motivating Example. DRL-based xApps are usually designed with internal reward functions that optimize radio resource usage, such as spectral occupation and PRB utilization, or ensure optimal scheduling policy selection. However, these radio-based metrics do not necessarily correlate with the actual application-layer service quality experienced by users measured through metrics such as latency, jitter, or peak throughput. To showcase this, we analyzed the impact of choosing a different slice-scheduling combination on the application layer throughput reported by RAN users. We

MobiHoc '25, Houston, TX, USA

ACM ISBN 979-8-4007-1353-8/25/10
https://doi.org/10.1145/3704413.3764425

configured an eMBB traffic scenario emulating a video-streaming application traffic of a constant bit-rate of 2 Mbps for four users sharing the eMBB slice and we analyze two distinct slice-scheduling actions focusing on two users (*User 1* and *User 2*). The first, *Action A*, allocates 36 Physical Resource Blocks (PRBs) using the Proportional Fair (PF) scheduling algorithm, while the second, *Action B*, allocates 9 PRBs using Round Robin (RR) scheduling. Empirically, Action A emerges as a better allocation strategy for eMBB traffic due to its higher PRB allocation, which enables more efficient bandwidth distribution and improved throughput sustainability. Figure 2 presents the Cumulative Distribution Function (CDF) of the throughput collected from the application layer report for both actions for the two users, and the average downlink bit rate across all slice-users (slice-avg) collected from BS KPM report. The results indicate a noticeable throughput degradation when transitioning from Action A to Action B for both users. Particularly, User 2 experiences a considerably higher probability of achieving lower application layer throughput than the slice average. Interestingly, when observing only the average network metrics, the impact appears less pronounced, potentially masking critical individual user performance degradations. Application-layer QoS should be considered to guarantee adequate user-centric performance when designing robust DRL training strategies.

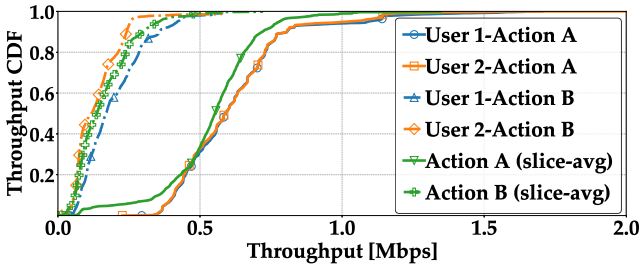


Figure 2: Impact of slice-scheduling combinations (Action A and B) on the application-layer throughput for two eMBB users (User 1 and 2).

Summary of Novel Contributions

• We present AdvO-RAN to analyze the robustness of DRL-based xApps against state-of-the-art test-time evasion attacks and mitigate Service Level Agreement (SLA) violation under such attacks. AdvO-RAN evaluates the effectiveness of DRL-based xApps by testing their behavior under adversarial perturbations and is designed to facilitate the development of DRL-based xApps that are robust and maintain SLA objectives amidst adversarial attacks;

• We consider the case where the adversary obtains a victim xApp from the O-RAN marketplace, learns an adversarial policy that can effectively disrupt QoS of different slice-based SLAs, and publishes a corrupted version of the xApp on the marketplace. This corrupted xApp executes adversarial test-time attacks based on a predefined attack strategy while complying with constraints like perturbation limit to avoid detection by the system's intrusion detection mechanisms;

• We experimentally evaluate the performance of AdvO-RAN on the Colosseum network emulator [11]. Our prototype consists of 49 SDR nodes, with 42 designated as User Equipments (UEs) and 7 as BSs. Specified in Section 5, we consider two sets of robust DRL agents operating under distinct traffic profiles and action modalities. The results indicate that our threat model generalizes across various DRL-based xApps with heterogeneous control strategies and traffic scenarios. Under adversarial conditions, our robust training strategy significantly reduces SLA violations – defined as the percentage of time a slice's DRL-based xApps with different modalities of actions and traffic scenarios. We show that adversarial robust training reduces SLA violations (i.e., the percentage of time the slice Key Performance Indicator (KPI) is 70% less than the required value) from 44% down to 27% under the strongest attacker for URLLC users. Similarly, for eMBB users, AdvO-RAN reduces SLA violations from 33% to 17% in the worst-case attack. Additionally, AdvO-RAN recovers approximately 46% of the degraded End-to-End (E2E) latency and up to 75% of the lost throughput.

2 System Model

As illustrated in Figure 3, we consider an xApp performing joint *PRBs slicing* and *scheduling* operations by determining optimal PRB allocation and scheduling algorithm for each slice. Specifically, the network slicing policy specifies the number of PRBs that should be allocated to each slice, while the slice-specific scheduling algorithm identifies the policy to be used for the UE downlink communication on each slice among RR, PF and Water Filling (WF).

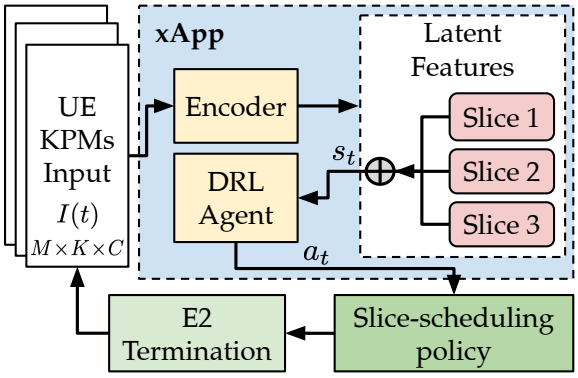


Figure 3: Example of DRL-based controller xApp.

Similar to previous work [12, 13], we consider the set C of $|C| = C = 3$ slices characterized by eMBB, URLLC, and Massive Machine Type Communications (mMTC) traffic classes, respectively. Each slice is associated with a unique traffic profile and the xApp's DRL agent is rewarded following the extent to which it generates actions that meet the slice-specific performance objectives. Specifically, the eMBB slice UEs requires a high bit rate to foster applications such as video streaming or conferencing. The URLLC slice requires extremely low latency which implies the need for a low buffer occupancy at the UEs. Lastly, in the mMTC slice, UEs typically represent Internet of Things (IoT) devices, e.g., small sensors, and should support a massive number of short transmission requests from several devices. We denote by $\mathcal{K}$ the set of KPMs received via the E2 interface. Specifically, we use $|\mathcal{K}| = K = 3$ KPMs, namely, transmission bitrate (indicated by b_{tx}), number of transmitted packets (indicated by pkt_{tx}) and size of the downlink buffer (indicated by buf_{dl}). The xApp is designed to receive slice-specific KPMs from the RAN and compute optimal PRB slicing and scheduling policy for each slice relying on a DRL agent (see Figure 3). Once the control action is generated by the DRL agent, the Near-RT-RIC implements this control decision in the RAN via the E2 Interface.

A DRL-controlled xApp is usually represented by a Markov Decision Process (MDP) $\mathcal{M} = \langle \mathcal{S}, \mathcal{O}, \mathcal{A}, P, R, \gamma \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{O}$ is the observation space, $\mathcal{A}$ is the action space, P is the transition probability function, R is the reward function, and $\gamma \in (0, 1)$ is the discount factor. At each time step t, the agent receives a state representation $s_t \in \mathcal{S}$ and an action $a_t \in \mathcal{A}$ interacting with the environment including BSs and slice users. The transition between two states $s_t \rightarrow s_{t+1}$ is defined by the transition probability $P : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$, where Δ is a function that maps a state-action pair (s, a) to a probability distribution over next possible states s_{t+1}. Based on this action, the agent obtains a reward according to a reward function $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. After training, a fixed policy π is used by the xApp to take actions based on the RAN state it receives. Specifically, each time step t, the RAN state is composed of the KPMs combined into a matrix $I(t) \in \mathbb{R}^{M \times K \times C}$, where M is the number of measurement reports collected at time step t. For example, if $M = 10$ measurement reports are collected for each step t, the monitored KPMs is $K = 3$, and we have $C = 3$ slices, the shape of the input matrix $I(t)$ becomes $10 \times 3 \times 3$. Now, we denote by $k \in \mathcal{K} = \{b_{tx}, pkt_{tx}, buf_{dl}\}$ the KPM index in matrix I, and by $m \in \{1, \ldots, M\}$ and $c \in \{1, \ldots, C\}$ the measurement and slice indices, respectively. An element of the KPM matrix I can then be identified by $i_{m,k,c}$. Indicating with w_c the slice-specific weight component of the reward signal, the reward function is formulated as:

$$R_k(t) = \sum_{c=1}^{C} w_c \cdot \frac{1}{M} \sum_{m=1}^{M} i_{m,k,c}(t). \quad (1)$$

For the eMBB slice the target KPM is the transmit bitrate, while target KPM for mMTC and URLLC slices are the number of transmitted packets and the downlink buffer size respectively. For example, an action that increases bit rate for eMBB UEs, maximizes the number of transmitted packets for mMTC, and decreases downlink buffer occupancy for URLLC UEs UEs generates a high reward as it seeks to maximize network performance and converge to the optimal slice/scheduling policies for each slice. By modulating the weights w_t, the agent is trained to maximize specific objectives for a particular slice or a global objective for the entire system [13]. In this work, we consider two agents, namely *eMBB-MAX (EM)* and *uRLLC-MAX (UM)*. The former has the objective of maximizing the average eMBB user throughput using a reward function based on the average throughput of the slice users. Instead, the uRLLC-MAX (UM) agent targets the minimization of the latency of URLLC traffic flows using a reward function inversely proportional to the downlink buffer metric. Note that, depending on the objective of the agent, the goal of the adversary also needs to be changed (see Section 4).

As depicted in Figure 3, an encoder block preprocesses the RAN KPM matrix into a compact latent representation, which serves as the actual state representation. The input matrix I is processed in a slice-independent fashion, obtaining a latent feature vector for each slice. In this way, each vector contains the encoded information of KPMs of each slice. The $C = 3$ latent vectors are then concatenated to generate the input for the DRL agent. The encoder block is trained as part of an autoencoder to reduce the dimensionality of the input matrix as well as removing noise from data, thus facilitating training and generalization of the DRL agent [14]. After training, only the encoder is retained and included in the system. The concatenated latent feature vector is then fed into the DRL agent.

3 Threat Model

We design an evasion attack strategy to a DRL victim agent in the xApp. In line with recent literature, we consider a worst-case scenario with white-box access to the victim's policy π [15–17]. The attacker has access only to the victim agent's model parameters and can observe its interactions with the environment, but does not have knowledge of w_c, the reward function weights. We are interested in a typical state adversary that perturbs the state observations (i.e., the concatenated latent representations $s(t)$ output by the encoder) before they reach the agent. Such a state adversary is modeled by a function h which perturbs the state $s \in \mathcal{S}$ such that it becomes $\tilde{s} := h(s)$. To ensure stealthiness and practicality, we consider the commonly adopted ℓ_p norm ball as the perturbation. The perturbation applied to s is constrained such that $\tilde{s} \in \mathbb{R}^n$ and $||\tilde{s} - s||_p \leq \epsilon$, where ϵ is perturbation magnitude budget and n is the dimensionality of the states. We make the following assumptions: (1) **Access.** We assume the attacker gains *unauthorised, time-limited access* to the E2 interface, enabling eavesdropping on control messages and KPM data—an exposure typical of open O-RAN interfaces .(2) **Knowledge.** By downloading a *third-party implementation of the victim xApp* from an open marketplace, the attacker can inspect the code and learn the exact state- and action-space dimensions; we also assume it knows the encoder architecture used to train the xApps. (3) **Ability.** Holding a foothold on the E2 link, the attacker can intercept ASN.1-encoded messages and replace them with adversarial ones containing perturbed KPMs; ASN.1 structures data but does not encrypt it. (4) **Perturbation budget.** Perturbations are bounded in an ℓ_p ball: $\|\tilde{s} - s\|_p \leq \epsilon$. The adversary seeks to maximise SLA violations while keeping changes small enough to evade detection. **Remark.** Anomaly-detection defences on the E2 interface are complementary counter-measures and therefore lie outside the scope of this work.

4 Design of AdvO-RAN

AdvO-RAN protects DRL agents against different types of attacks based on the constraints set by the operators (i.e., the traffic profiles, and perturbation budget). A walk-through of the main operations of AdvO-RAN is summarized in Figure 4.

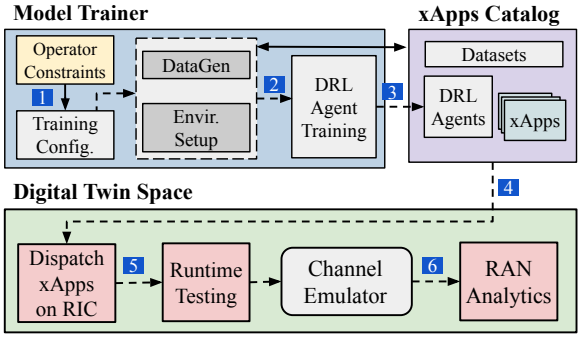


Figure 4: AdvO-RAN workflow.

First, the robust DRL agent constraints set by the operator are parsed (Step 1 in Figure 4) to extract relevant robust model training configurations such as perturbation budget, traffic profile/scenario, estimated reward function parameters for preference based reinforcement learning (PbRL) – details in Section 4.1. This configuration is used to set up an adversarial O-RAN environment through

the 'DataGen' module (step 2). The adversarial O-RAN environment in the context of DRL agent training requires defining an optimal adversary given a set of constraints (set by the operators) adding perturbations to the RAN KPM. The DRL agent is robustly trained (following the steps detailed in Section 4.1) using adversarial state representations and stored in the xApps catalog (Step 3).

An adversarial dataset (state-action pairs along with corresponding application layer QoS) is collected for training the estimated reward function for PbRL. From the xApps catalog, trained robust agents are dispatched to Near-RT RIC (step 4) for deployment, where they undergo runtime testing (Step 5) within a digital twin.A channel emulator using real-world cellular traces enables runtime testing of the robust agent. The analytics module (Step 6) evaluates performance, revealing how adversarial conditions impact xApp functionality and system adaptability. Thus, AdvO-RAN proactively assesses and optimizes DRL-based xApps before live O-RAN deployment, enhancing reliability and robustness.

4.1 Adversarial Robust Training

To effectively train a robust DRL-based xApp, it is essential to identify the optimal adversarial policy π_a within the same MDP environment in which the victim xApp operates. As detailed by Zhang et al. [17], such an adversary exists within a state-adversarial Markov Decision Process (SA-MDP), formally defined by the tuple $\hat{\mathcal{M}} = (\hat{\mathcal{S}}, \hat{\mathcal{A}}, \hat{h}, \hat{\mathcal{R}}, \hat{\mathcal{P}})$, whereas the victim xApp is originally trained in a clean MDP $\mathcal{M}$. In the SA-MDP, $\hat{\mathcal{S}}$ represents the adversarial state-space, $\hat{\mathcal{A}}$ is the adversarial action space and $\hat{h}(s) \in \hat{S}$ characterizes the state perturbation imposed by the adversarial policy, as discussed in Section 3. To formulate optimal state perturbations, the adversary learns the state-action transition probabilities $\hat{\mathcal{P}}$ within the SA-MDP. Recent state-adversarial threat model approaches, such as those proposed in [17, 18], adopt adversarial learning strategies that optimize the adversarial policy primarily based on the victim's reward function. However, this way, the adversary does not learn the perturbation policy based on the QoS metrics.

Data-driven Adversarial Reward Modeling: To obtain a reward estimation based on QoS, we follow the PbRL framework [19] which does not rely on the ground-truth reward function and is instead based on learning human intention. As we do not require human intervention, we modify the method by incorporating the application layer QoS. We consider training *policy* π_θ with reward function $\hat{r}_\psi$ parameterized by θ and ψ. The latter provides the abstraction between QoS and victim DRL state-action transitions.

The first step to train π_θ entails the collection of a dataset $\mathcal{D}$ consisting of state-action sequence pairs noted as (σ^x, σ^z). Each sequence $\sigma = \{s_t, a_t, \cdots, s_{t+d}, a_{t+d}\}$ contains d state-action transitions of the victim DRL agents from timestep t up to timestep $t+d$. A state-action sequence σ is obtained by using a random initial state s_t and collecting the evolution of the MDP representing the xApp. The initial state s_t is selected through a random sampling technique from the set $\mathcal{S}$ of input states of the victim's DRL agent collected for offline training. Hence, σ^x and σ^z are two possible evolutions of the system that are then labeled using preference-based labeling. Specifically, each pair of sequences (σ^x, σ^z) is associated with a preference label $y \in \{(0,1),(1,0)\}$ indicating which segment is preferred by the attacker based on the application-layer performance metrics. We follow a data-driven approach to automatically mark each (σ^x, σ^z) in the dataset with a preference label. This is done relying on the application-layer KPMs (i.e., end-to-end throughput, latency, packet loss, video stuttering count, etc.) which are collected together with the $\mathcal{D}$ dataset. Based on the KPMs, the attacker uses a thresholding strategy on the SLA violation to determine which state-action sequence aligns better with its goal and, in turn, associates the y label to the pair. To do this, we define a binary indicator function $\mathbb{I}_\sigma(t)$ for each time step t in a sequence σ. The function returns 1 if a SLA violation occurs in that timestep and 0 otherwise. Hence, we define a SLA violation score for the sequence σ as

$$V_{sla}(\sigma) = \frac{1}{d} \sum_{j=t}^{t+d} \mathbb{I}_\sigma(j). \tag{2}$$

The attacker's preferred state-action sequence σ is the one with the highest SLA violation score. Hence, y is set to $(0,1)$ if $V_{sla}(\sigma^x) > V_{sla}(\sigma^z)$, while if $V_{sla}(\sigma^z) > V_{sla}(\sigma^x)$ the y label is set to $(1,0)$. In case of $V_{sla}(\sigma^x) = V_{sla}(\sigma^z)$ label is assigned randomly. After labeling all the entries, the dataset $\mathcal{D}$ becomes a set of triplets in the form (σ^x, σ^z, y). The reward modeling procedure is summarized in Algorithm 1. The algorithm learns the reward model $\hat{r}_\psi$ which provides a scalar reward for a given state-action sequence. The algorithm starts with loading dataset $\mathcal{D}$ and initializing the reward policy π_θ, the reward model $\hat{r}_\psi$, and the target Q-function $Q_{\bar{\phi}}$ used in soft actor-critic learning (lines 1 and 2 in Algorithm 1). The algorithm iteratively (over different epochs, line 3) updates the reward model based on the difference between the y label and the preference predicted for the (σ^x, σ^z) sequence pair. We define a predictor based on a Bradley-Terry [20] model. Using this, the probability that sequence σ^x is preferred over σ^z is obtained as

$$P_\psi[\sigma^x \succ \sigma^z] = \frac{\exp \sum_t \hat{r}_\psi(s_t^x, a_t^x)}{\sum_{i\in\{x,z\}} \exp \sum_t \hat{r}_\psi(s_t^i, a_t^i)}. \tag{3}$$

After obtaining $P_\psi[\sigma^x \succ \sigma^z]$ (lines 4 and 5 in Algorithm 1) for a batch of samples in the dataset $\mathcal{D}$, the reward model parameter ψ is optimized to align the preferences with the preference labels y. To do this, we minimize the cross-entropy loss $\mathcal{L}(\psi)$ between

Algorithm 1 Soft actor-critic adversarial reward learning

1: **Input:** Dataset $\mathcal{D}$, reward model $\hat{r}_\psi$ with initial parameters ψ, learning rate η, discount factor γ, replay buffer B, number of gradient steps
2: **Initialize:** π_θ, Q_ϕ, $\hat{r}_\psi$, and $Q_{\bar{\phi}}$
3: **for** each training epoch **do**
4: **for** each batch $(\sigma^x, \sigma^z, y) \sim \mathcal{D}$ **do**
5: Compute preference probability (see Equation (3))
6: Compute cross-entropy loss (see Equation (4))
7: Update ψ using gradient descent: $\psi \leftarrow \psi - \eta \nabla_\psi \mathcal{L}(\psi)$
8: **for** each timestep t in both σ^x and σ^z **do**
9: $\hat{r}_t \leftarrow \hat{r}_\psi(s_t, a_t)$
10: Store transition $B \leftarrow B \cup \{s_t, a_t, s_{t+1}, \hat{r}_t\}$
11: **for** each gradient step **do**
12: Sample transitions $(s_t, a_t, s_{t+1}, \hat{r}_{t\psi}) \sim B$
13: Compute target soft value $\bar{V}(s_{t+1})$ using Equation (6)
14: Update Q-function parameters using Equation (5)
15: Update π_θ using Equation (7)
16: **Return:** Trained $\hat{r}_\psi$ and π_θ

the prediction $P_\psi[\sigma^x \succ \sigma^z]$ and preference label y as (line 6 in Algorithm 1), which is formulated as

$$\mathcal{L}(\psi) = -\mathbb{E}_{(\sigma^x,\sigma^z,y)\sim D}\left[\sum_{i\in\{x,z\}} y(i)\log P_\psi\left[\sigma^i \succ \sigma^{\{x,z\}\setminus\{i\}}\right]\right]. \quad (4)$$

The gradient descent algorithm is used to update ψ based on the loss in Equation (4) (line 7, where η is the learning rate). Hence, for each timestep t in the σ sequences, we compute the estimated reward $\hat{r}_{\psi,t}$ and store the transitions in buffer B (lines 8-10). Following the approaches in work [21], we train a soft actor-critic policy π_θ using reward $\hat{r}_\psi$. The critic Q-function Q_ϕ parameterized by ϕ is optimized by reducing the Bellman residual with a state value function $\bar{V}(.)$ and the estimated reward $\hat{r}_t$. Specifically, for each transition (s_t, a_t, s_{t+1}) sampled from buffer B the Bellman residual is obtained as

$$J_Q(\phi) = \mathbb{E}_{(s_t,a_t,s_{t+1})\sim B}\left[\left(Q_\phi(s_t,a_t) - \hat{r}_t - \gamma\bar{V}(s_{t+1})\right)^2\right], \quad (5)$$

where the state value function $\bar{V}(.)$ is obtained accounting for the target soft Q-value function parameterized by $\bar{\phi}$ as

$$\bar{V}(s_t) = \mathbb{E}_{a_t\sim\pi_\theta}\left[Q_{\bar{\phi}}(s_t,a_t) - \mu\log\pi_\theta(a_t|s_t)\right]. \quad (6)$$

Finally, the reward policy π_θ is updated by minimizing the loss function $J_\pi(.)$ formulated as

$$J_\pi(\theta) = \mathbb{E}_{s_t\sim B, a_t\sim\pi_\theta}\left[\mu\log\pi_\theta(a_t|s_t) - Q_\phi(s_t,a_t)\right]. \quad (7)$$

This way, AdvO-RAN captures the state-action pairs that cause maximum SLA violations while learning reward $\hat{r}_\psi$ and policy π_θ.

Training an adversary for state-space perturbation: After training the preference-based reward model $\hat{r}_\psi$ and the aligned policy π_θ using Algorithm 1, the next step is to train an adversarial state-space perturbation function $\mathcal{V}(s)$, where $s \in \mathcal{S}$ is the actual latent state representation. The function $\mathcal{V}(s)$ outputs a perturbation vector that modifies the state as $\tilde{s} = s + \mathcal{V}(s)$, with the constraint that the perturbation magnitude is bounded by a perturbation magnitude budget $\epsilon > 0$, i.e., $\|\mathcal{V}(s)\|_p \le \epsilon$, where $\|\cdot\|_p$ denotes the ℓ_p norm. The objective is to perturb the input state such that the victim xApp's policy, denoted by π_v produces actions that closely mimic the behavior of the attacker's preference-aligned policy π_θ, when evaluated on the perturbed state $\tilde{s}$. To this end, the adversary is trained to maximize the cumulative reward given by the learned preference-based reward function $\hat{r}_\psi$, while ensuring that the perturbations remain within the norm ball defined by ϵ (see Section 3). The adversary is therefore optimized to maximize the expected cumulative reward using the learned reward model, i.e., it maximizes the function

$$\mathcal{J}(s_t;\omega) = \mathbb{E}_{s\sim\mathcal{D}_\omega}\left[\sum_{t=0}^{T}\gamma^t\cdot\hat{r}_\psi\big(s_t + \mathcal{V}_\omega(s_t), \pi_v(s_t + \mathcal{V}_\omega(s_t))\big)\right], \quad (8)$$

where $\mathcal{D}_\omega$ is the distribution of latent states collected from the environment, T is the episode length and $\gamma \in (0,1)$ is the discount factor that controls the importance of future rewards. The adversary aims to find the parameters ω that maximize this cumulative reward. To enforce the perturbation constraint, we introduce a penalty term that increases the loss when the norm of the perturbation exceeds the budget set by the ℓ_p ball following equation:

$$\mathcal{L}_{\text{pen}}(s_t) = \lambda\cdot\mathbb{E}_{s_t\sim\mathcal{D}_\omega}\left[\max\left(0, \|\mathcal{V}_\omega(s_t)\|_p - \epsilon\right)\right], \quad (9)$$

where $\lambda > 0$ is a regularization coefficient to penalize the constraint. Additionally, to explicitly align the behavior of the victim policy π_v on the perturbed states with the preference-based policy π_θ we introduce a sequence-level imitation loss using the Kullback–Leibler (KL) divergence between $\pi_v(s_t)$ and $\pi_\theta(s_t)$ action probability distributions:

$$\mathcal{L}_{\text{KL}}(s_t) = \beta\cdot\mathbb{E}_{s_t\sim\mathcal{D}}\left[D_{\text{KL}}\left(\pi_v(s_t + \mathcal{V}_\omega(s_t)) \,\|\, \pi_\theta(s_t)\right)\right], \quad (10)$$

where $\beta > 0$ is a weighting factor that controls the strength of this imitation loss. Overall, the loss used to train $\mathcal{V}_\omega$ is a combination of the three components introduced above in Equations 8, 9, 10, i.e.,

$$\mathcal{L}(s_t;\omega) = \mathcal{J}(s_t;\omega) + \mathcal{L}_{\text{pen}}(s_t) + \mathcal{L}_{\text{KL}}(s_t). \quad (11)$$

We minimize $\mathcal{L}(\omega, s_t)$ using stochastic gradient descent with Adam optimizer. The perturbation function $\mathcal{V}_\omega$ is modeled as a fully connected neural network with input size equal to the dimension of the concatenated latent features $s \in \mathcal{S}$ and the output is a perturbation vector $\delta = \mathcal{V}(s)$ of the same dimension.

Optimized training for robustness: Finally, to achieve robustness with a policy that can counter the effect of adversary $\mathcal{V}$, we follow the approaches in ATLA-PPO [22], a generalized training framework that employs an alternating training approach. Algorithm 2 describes the robust training method with a fixed adversary.

Algorithm 2 Robust training with fixed adversary

1: **procedure** AdvTraj($\mathcal{V}_\omega, \hat{r}_\psi, b, \pi_\tau$)
2: Initialize empty dataset $\mathcal{D}_\pi$
3: **for** $i = 0$ to b **do**
4: Sample a starting state $s_0 \sim p(s)$
5: **for** $t = 0$ to T **do**
6: Compute perturbation $\delta_t = \mathcal{V}_\omega(s_t)$
7: Compute perturbed state $\tilde{s}_t = s_t + \delta_t$
8: Sample action $a_t \sim \pi_\tau(\cdot|\tilde{s}_t)$
9: Execute a_t and get next state s_{t+1}
10: Compute reward $r_t = \hat{r}_\psi(\tilde{s}_t, a_t)$
11: Append transition $(s_t, \tilde{s}_t, a_t, r_t, s_{t+1}) \to \mathcal{D}_\pi$
12: **return** $\mathcal{D}_\pi$
13: **Input:** Robust policy π_τ, adversary $\mathcal{V}_\omega$, number of iterations N_{iter}, batch size b.
14: **for** $i = 1$ to N_{iter} **do**
15: $\mathcal{D}_\pi \leftarrow$ AdvTraj($\mathcal{V}_\omega, \hat{r}_\psi, b, \pi_\tau$)
16: **Compute advantage estimates:**
17: **for** $(s_t, \tilde{s}_t, a_t, r_t, s_{t+1}) \in \mathcal{D}_\pi$ **do**
18: Compute return using Equation (1): $\mathcal{R}_t = \sum_{t=0}^{T}\gamma\, R(t)$
19: Estimate value $V(s_t)$
20: Compute advantage $\hat{A}_t = \mathcal{R}_t - V(s_t)$
21: Compute PPO loss $\mathcal{L}_{\text{PPO}}$ using Equation (12)
22: Compute robust loss $\mathcal{L}_{\text{entropy}}$ Equation (13)
23: Update π_τ using stochastic gradient descent
24: **return** π_τ

The input of the algorithm contains robust policy we intend to train π_τ parametrized by τ, the adversary $\mathcal{V}_\omega$, the number of iterations N_{iter}, and the batch size b. During each iteration, a batch of b perturbed trajectory is generated using the AdvTraj procedure (lines 1-12 in Algorithm 2). The procedure starts by initializing an

empty replay buffer $\mathcal{D}_\pi$ (line 2) that is populated iterating over batches of b samples (line 3), starting from a random starting state s_0 sampled from the state distribution $p(s)$ (line 4). For each time step t (line 5 in Algorithm 2), we first compute the adversarial perturbation δ_t using a fixed, pretrained $\mathcal{V}_\omega$, which takes the clean state s_t as input (line 6). The resulting perturbed state is then obtained as $\tilde{s}_t = s_t + \delta_t$, simulating the effect of corrupted or adversarial observations (line 7). Using this perturbed state, the robust policy π_τ samples an action a_t according to its current policy distribution (line 8). The action a_t is then executed in the environment (line 9), and the next state s_{t+1} is observed. A reward r_t is generated using the learned preference-based reward model $\hat{r}_\psi$, which reflects application-layer feedback. Finally, the full transition sequence $(s_t, \tilde{s}_t, a_t, r_t, s_{t+1})$ is stored in the replay buffer $\mathcal{D}_\pi$ (line 11). This process is repeated for each timestep and sequence, and once all trajectories are collected, the buffer $\mathcal{D}_\pi$ is returned (line 13).

By using the AdvTraj procedure, we collect a batch of perturbed trajectories using the current robust policy π_τ and reward model $\hat{r}_\psi$. Next, the algorithm estimates the advantage function $\hat{A}_t = R_t - V(s_t)$ for each timestep in the collected transitions (line 20 in Algorithm 2). This involves computing the discounted return $R_t = \sum_{t=0}^{T} \gamma\, R(t)$ (line 18) and subtracting the estimate value $V(s_t)$ (line 19). Once the advantages are computed, we evaluate the PPO surrogate loss (line 21) where the objective is to maximize a clipped policy improvement to ensure stable updates, i.e.,

$$\mathcal{L}_{\text{PPO}}(\tau) = \mathbb{E}_{(\tilde{s},a)\sim\mathcal{D}_\pi}\Bigg[\min\Big[\rho_t(\tau)\hat{A}_t, \\ \text{clip}\left(\rho_t(\tau), 1-\epsilon_{\text{clip}}, 1+\epsilon_{\text{clip}}\right)\hat{A}_t\Big]\Bigg]. \tag{12}$$

Hence, the robust loss is computed by adding an entropy regularization term to the PPO loss to encourage policy stochasticity and exploration (line 22). Mathematically, the robust loss is obtained as

$$\mathcal{L}_{\text{robust}}(\tau) = \mathcal{L}_{\text{PPO}}(\tau) - \eta_{\text{ent}} \cdot \mathbb{E}_{\tilde{s}\sim\mathcal{D}_\pi}\left[\mathcal{H}[\pi_\tau(\cdot|\tilde{s})]\right], \tag{13}$$

where the term $\mathcal{H}[\pi_\tau(\cdot|\tilde{s})]$ denotes the entropy of the action distribution under the policy given the perturbed state $\tilde{s}$. The term η_{ent} is a hyperparameter that controls the strength of the entropy regularization term. The term $\rho_t(\tau) = \frac{\pi_\tau(a_t|\tilde{s}_t)}{\pi_{\tau_{old}}(a_t|\tilde{s}_t)}$ is the importance sampling ratio between the new and old policy probabilities for action a_t. This robust loss is minimized over each iteration using a stochastic gradient descent algorithm and the policy π_τ is updated accordingly (line 23). After training the π_τ in this way, we obtain a robust agent with a policy that is resilient to adversarial perturbations generated by $\mathcal{V}(s)$. This policy enables the agent to maintain adequate slice-level performance under adversarial conditions.

Although the Bradley–Terry model transforms each sequence pair (σ^x, σ^z) into the probability $P_\psi[\sigma^x \succ \sigma^z]$ in Eq. (3), it assumes the associated label is correct and weights every observation equally. Thus, mislabeled or extreme-reward pairs may still influence the maximum-likelihood fit, as the model offers no guarantee of statistical robustness to such outliers.

5 AdvO-RAN Experimental Evaluation

We prototyped AdvO-RAN on the Colosseum network emulator [12, 23]. We utilized the ColO-RAN and SCOPE frameworks [24] which allow simulating realistic O-RAN deployments. We benchmarked AdvO-RAN in multiple network scenarios comprising 7 BSs, each serving 6 UEs, along with O-RAN components such as the Near-RT-RIC and xApps interfaced with BSs via the E2 interface. BSs and UEs are implemented as Linux Containerss (LXCs). Specifically, we conducted our experiments using the SCOPE Rome Urban scenario, where BS placement reflects real cell deployments from the OpenCellID database [25]. Traffic generation is handled by the Multi-Generator (MGEN) tool, which allows emulating different traffic profiles and capturing application-layer metrics at the UE side for each transmitted packet.

We considered two traffic profiles (**TP1** and **TP2**) associated with different agents' action modalities. In TP1, we consider an agent that allocates a total of 50 PRBs for a 10 MHz channel across the slices and also selects the scheduling policy for each slice. The maximum application data rate for eMBB, mMTC and URLLC users in TP1 emulation is set to 4 Mbps, 44 kbps and 89.3 kbps, respectively. An agent that only selects the scheduling policy while keeping the PRBs allocation fixed for a 3 MHz channel is instead considered for TP2. The maximum application data rate for eMBB, mMTC and URLLC users in TP2 emulation is set to 1 Mbps, 30 kbps and 10 kbps respectively. The agent's actions are applied through SCOPE. Observations for the agent are generated by periodically sampling data reflecting the most recent 250 ms of network activity, corresponding to a single timestep in the environment. For training purposes, we define each episode to span 10 timesteps. We employ a Proximal Policy Optimization (PPO) to train each victim model. The actor and value networks are composed of 5 fully connected layers each, with 30 neurons, using the hyperbolic tangent (`tanh`) activation function. The agent is trained using a learning rate of 1e−3 and a replay buffer batch size of 64. The discount factor γ is set to 0.1, and Generalized Advantage Estimation (GAE) is leveraged to reduce variance. Entropy regularization is applied with a coefficient of 0.1 to encourage exploration and an importance ratio clipping of 0.2 is used to stabilize policy updates. Regarding SLA violation and E2E metric comparison, we choose 2 PRBs for the eMBB slice and 4 PRBs for the URLLC slice to generate a resource-constrained traffic scenario for eMBB users.

5.1 Attack and Robust Training Baselines

We consider three baseline attacks in addition to our perturbation strategy. The four approaches are first compared to assess their attack power and then used to evaluate AdvO-RAN robust training.

- **Random Attack (RA):** Perturbations are uniformly sampled within an L_2-norm ball of radius ϵ centered around the state space;
- **Worst-Action Attack (WA):** In this attack – introduced in [16] – the adversary perturbs the state by following the sign of the gradient of the loss between the victim policy and an alternative policy that always selects the least likely action (i.e., the action with the lowest probability under the victim's current policy). We set $p = \infty$ for ℓ_p-norm to generate perturbation noise in the input state following the original implementation;
- **Policy Adversarial Actor Director Attack (PA-AD):** It is a two-stage procedure involving a director and an actor. The director solves an MDP to identify the most effective gradient direction to perturb the xApp observations. The actor then executes a gradient-based adversarial attack — aligned with the direction suggested

by the director — to generate the perturbation [18]. We use Projected Gradient Descent (PGD) in [26] as a perturbation generation method with 20 iterations. We set $p=2$ for ℓ_p-norm based PGD to find the perturbation direction;

• **Adv-ORAN Attack (AO):** It leverages the perturbation generated by our AdvO-RAN framework. We perform a comprehensive grid search to tune its hyperparameters and observe optimal attack performance with a learning rate of $\eta = 10^{-3}$ and a loss coefficient ratio of $\beta/\lambda = 0.2$ between the KL divergence loss (Equation (10)) and the penalty loss (Equation (9)). All trainable components are optimized using Adam. For the data-driven adversarial reward, we evaluate different state-action sequence segment lengths $d \in \{1, 2, 5, 10\}$ and find that $d = 10$ yields the fastest convergence.

In addition, we compare our AdvO-RAN robust training approach with the following two state-of-the-art strategies:

• **RADIAL-PPO [27]:** This method modifies the PPO objective using an additional adversarial loss term. The goal is to reduce the sensitivity to input perturbation. It defines a worst-case policy distribution by explicitly shifting probability mass away from the optimal policy. The adversarial loss term is defined as the expected clipped PPO loss between the original and worst-case policy distributions under perturbations;

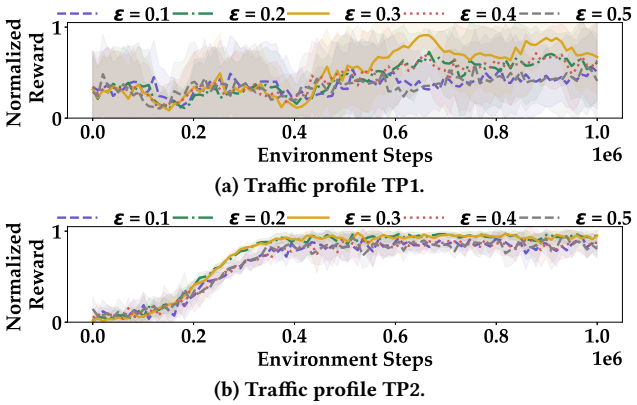
• **SA-PPO [28]:** This framework trains PPO agent integrating a robust regularizer term in the policy optimization phase derived from the SA-MDP framework. This approach explicitly models the presence of worst-case perturbations in the state observations and penalizes policies that are overly sensitive to such attacks.

While these prior methods use the environment's native reward, our method introduces perturbations in the state space guided by a preference-based reward function, thus allowing for more targeted and context-aware adversarial behavior.

5.2 Evaluation Metrics

To understand the E2E impact of AdvO-RAN robust training we considered two application layer metrics: E2E throughput (measured in bit per second, [bps]) and E2E latency (in seconds, [s]). These metrics are collected on the user side during each experiment using MGEN traffic. For eMBB slice users, the E2E throughput represents the most significant metric that should be sufficiently high to support applications such as video streaming. For URLLC users, the key performance-related aspect is captured by the E2E latency as URLLC communication demands strict delay bounds to ensure timely and deterministic message delivery. At the medium access control (MAC) layer we collect information about the slice PRBs and scheduling algorithm for the users to determine resource allocation strategy under attack and during robust agents operation.

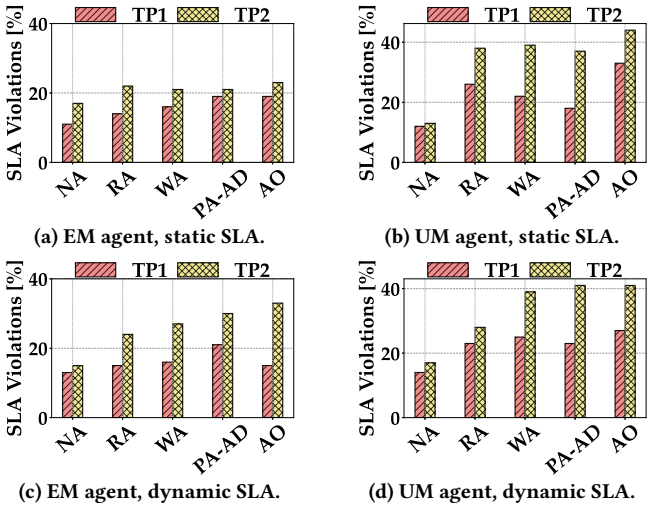
Using these metrics, we assess SLA violations under two conditions: *static* and *dynamic* thresholds. In the **static SLA** scenario, a violation occurs when the system delivers less than 70% of the predefined E2E performance target. For the eMBB slice, this corresponds to receiving less than 70% of the target E2E throughput (4 Mbps for TP1 and 0.7 Mbps for TP2), while for the URLLC slice, it corresponds to E2E latency exceeding 70% of the latency bounds (5 ms for TP1 and 10 ms for TP2). In the **dynamic SLA** scenario, we capture traffic demand variability by sampling the expected performance level at each time step from a Gaussian distribution



(a) Traffic profile TP1.

(b) Traffic profile TP2.

Figure 5: Normalized mean reward per episode under varying perturbation budgets. The lines show medians, shaded areas show variance.



(a) EM agent, static SLA.

(b) UM agent, static SLA.

(c) EM agent, dynamic SLA.

(d) UM agent, dynamic SLA.

Figure 6: Median SLA violations of AdvO-RAN (AO) and baseline attacks compared with the no-attack (NA) scenario.

$\mathcal{N}(0.7, 0.2^2)$. A violation is detected if the system fails to meet 70% of the sampled throughput target for eMBB, or if the latency exceeds 70% of the sampled latency bound for URLLC. SLA violations are evaluated over non-overlapping 5-second windows throughout the simulation.

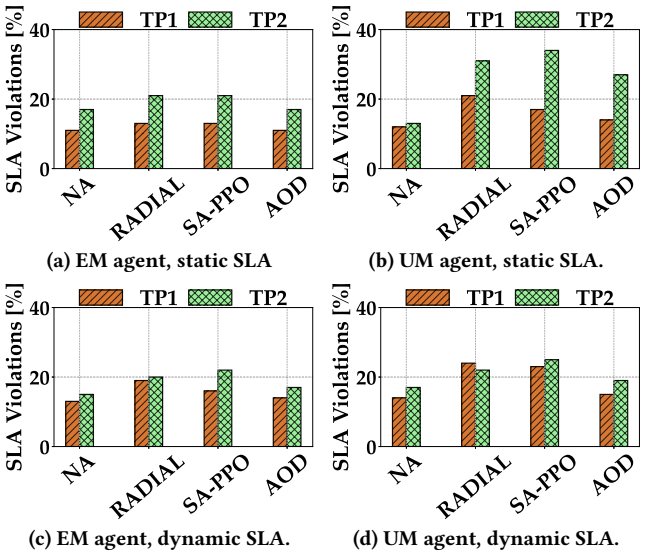
5.3 Experimental Results

In the following, we assess the effectiveness of AdvO-RAN in training robust PPO agents the previously defined metrics.

Perturbation budget hyperparameter selection: We used perturbation budget $\epsilon \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. Figure 5 presents the normalized average reward per episode over 1 million training steps for robust agents trained under each perturbation level. In the TP1 traffic scenario (Figure 5a), $\epsilon = 0.3$ yields the highest average reward, indicating more effective robustness. For TP2 (Figure 5b), both $\epsilon = 0.2$ and $\epsilon = 0.3$ result in comparable performance, with $\epsilon = 0.3$ slightly outperforming over longer training durations. Based on these findings, we select $\epsilon = 0.3$ as the perturbation budget for all subsequent evaluations.

SLA violation under different attacks: Figure 6 compares the median SLA violation of the AdvO-RAN attack (AO) and the baseline attacks presented in Section 5.1. A no-attack baseline (NA) has been included to quantify performance under benign conditions. Downlink traffic flows are simulated for 600 seconds, during which SLA violations are recorded according to the definitions provided in Section 5.2 for the EM and UM agents in static and dynamic SLA scenarios. Each attack scenario is executed for 10 iterations, and the median SLA violations is reported. The results in Figure 6 show that the UM agent is significantly more vulnerable than the EM agent under adversarial attacks. For example, in static SLA settings (Figure 6b), the UM agent experiences the highest SLA violation rate of approximately 44% under the AdvO-RAN attack for traffic profile TP2, compared to around 40% for WA and slightly less for PA-AD. Instead, the EM agent in the static SLA scenario (Figure 6a) maintains SLA violations below 25% across all attacks, demonstrating higher robustness compared to the UM agent. A similar pattern is observed in the dynamic SLA scenario. The UM agent (Figure 6d) suffers maximum SLA violations under the AO attack, reaching around 44% in TP2, while the EM agent (Figure 6c) limits SLA violations to below 30% even under the strongest attack (WA). Noticeably, our new *AO attack leads to consistently higher SLA degradation* compared to the RA, WA, and PA-AD baselines across all scenarios, especially when considering the UM agent. Furthermore, the TP1 scene consistently exhibits lower SLA violations than TP2 across all attacks and agent types, indicating that TP1 provides larger discrete action space as the agent can select both the allocation and scheduling policy, enabling higher robustness and thus reducing SLA violations. In contrast, in TP2 the agent's decision space is limited as it only selects the scheduling policy. This makes it more susceptible to well-crafted attacks like AO. These results confirm that agents operating under resource-constrained spaces (like TP2) are more susceptible to adversarial perturbations. To summarize, (i) The AO (AdvO-RAN) attack induces the highest degradation in both throughput and latency; (ii) TP2, associated with a smaller action space, is significantly more vulnerable under AO than TP1; (iii) TP1 demonstrates stronger resilience due to its higher action space, maintaining relatively stable performance even under attack; (iv) WA and PA-AD show moderate impact on the SLA violation, while RA is the least disruptive.

SLA violation under different robust training approaches. Figure 7 compares the effectiveness of AdvO-RAN (AOD) in the figure) and the other baseline robust training methods in reducing the SLA violation rates when the robust agents are attacked with the AdvO-RAN (AO) attack. The results are presented for both EM and UM agents under static and dynamic SLA scenarios. As shown in the figure, AdvO-RAN maintains SLA violation rates closer to the NA scenario than other robust training baselines. Under dynamic SLA conditions for EM agents (Figure 7c), the no-attack violation rate for TP1 is 13%, while AOD maintains violations at 15%. In contrast, SA-PPO and RADIAL result in violations of 23% and 24%, respectively, introducing a much larger gap of 10–11%. Similar trends are observed in TP2, where AOD reduces violations to 17–19%, while the baseline agents remain above 22-25%. For UM agents (Figure 7d & 7b), AOD does not perform as closely to the no-attack scenario as it does for EM agents, particularly in



(a) EM agent, static SLA (b) UM agent, static SLA.

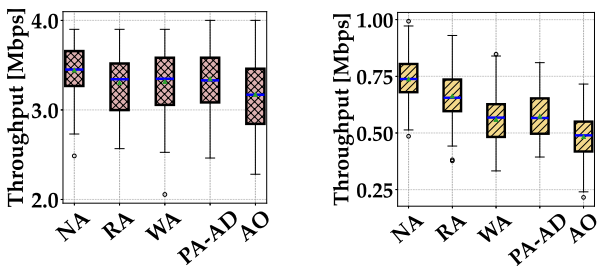
(c) EM agent, dynamic SLA. (d) UM agent, dynamic SLA.

Figure 7: Median SLA violations of AdvO-RAN robust agent and baseline robust training methods when the AdvO-RAN (AO) is in place compared with the no-attack (NA) situation.

TP2 under dynamic SLA. However, it still achieves improvements over the baselines, reducing SLA violations by up to 24% compared to SA-PPO and RADIAL. Overall, AOD remains effective across both static and dynamic SLA conditions, outperforming baselines with lower violations under adversarial attacks. Comparing the worst SLA violations observed in Figure 6b with the corresponding recovery shown in Figure 7b, *we can observe that AdvO-RAN method reduces SLA violations from 44% to 27% under the static and 41% to 19% in dynamic SLA scenario for TP2 traffic profile.*

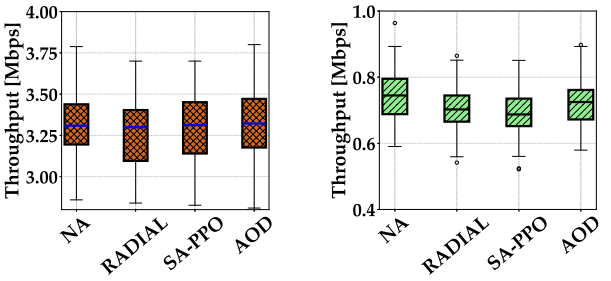
End-to-End Throughput and Latency: Complementing the SLA improvements, AdvO-RAN (AOD) also substantially enhances E2E application-layer performance, compensating for the degradation introduced by the AdvO-RAN (AO) attack. Figure 8 shows that our proposed AO attack reduces the median E2E throughput of eMBB slice users by 8% in TP1 and 45% in the TP2 traffic scenario—representing the highest degradation among all baseline attacks compared to the no-attack (NA) case. Analyzing Figures 8 and 9, we observe that AdvO-RAN (AOD) improved the E2E throughput by 3.33% and 75% compared to the worst-case attack scenario (AO) in TP1 and TP2 respectively. Similarly, for the URLLC slice agent (UM), the AdvO-RAN attack (AO) increases latency by 20% in TP1 and 91% in TP2 when compared to the NA case (see Figure 10a). Under the same AO attack environment, AdvO-RAN (AOD) robust training reduces the median latency by 15% in TP1 and 46% in TP2 (see Figure 10b).

AdvO-RAN in resource-constrained scenarios. To validate the effectiveness of AdvO-RAN in robust training under resource constrained conditions, we used TP2, in a single action modality (scheduling only with fixed PRBs per slice), chosen for its ease of visualizing the action distribution. In Figure 11 presents the scheduling policy distributions over 20 runs under three conditions: NA, AdvO-RAN (AO) attack, and AdvO-RAN (AOD) robust agent with AdvO-RAN (AO) attack. When the slice is provisioned with abundant resources (PRB = 8), the agent predominantly selects the WF



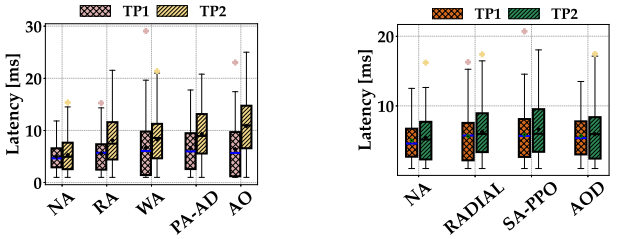
(a) Traffic profile TP1. (b) Traffic profile TP2.

Figure 8: End to End throughput, different attacks (EM agent).



(a) Traffic profile TP1. (b) Traffic profile TP2.

Figure 9: End to end throughput, different robust training approaches (EM agent).

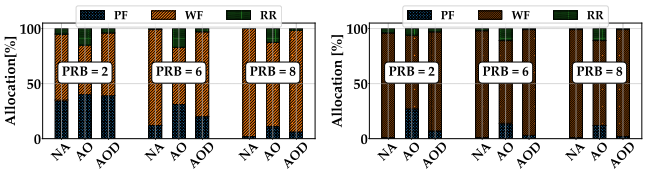


(a) Attack comparison. (b) Robust agent comparison

Figure 10: End to End latency under different attacks and using different robust training approaches for the UM agent.

policy for all users. In contrast, under limited PRB availability, the agent tends to favor the PF policy to ensure fairness and compensate for PRB scarcity. From Figure 11a, under attack the victim agent allocates policy RR more (14.56%) than in the NA condition (4.0%). AOD robust training reinstates this behavior of the agent and reduces the RR allocation to just 4.6%. Similarly, Figure 11b shows, AOD consistently reduces RR allocation, often replacing it with more effective scheduling strategies (PF). Notably, when the slice receives larger PRB allocation (PRB=8), AOD reduces RR usage from 12.55% to 1.33% in the URLLC slice, and from 13.05% to 0.02% in the eMBB slice.

Scaling factor. The encoder processes a fixed-length slice-level metric vector. If N_{UE} denotes the total number of user equipments and N_{BS} denotes the total number of base stations, its per-BS processing cost is constant and the aggregate latency grows as (N_{BS}) while remaining $O(1)$ with respect to N_{UE}. Because the attacker's and defender's policies take a single latent representation per BS



(a) UM agent scheduling policy distribution for URLLC slice (b) EM agent scheduling policy distribution for eMBB slice

Figure 11: Comparison of EM agent scheduling policy distributions across fixed slices (UM and eMBB) with TP2.

in each batch, their forward-pass complexity follows the same $O(N_{\text{BS}})$ trend, ensuring that the combined encoder–policy delay stays within the near-real-time control window.

6 Related Work

Security in O-RAN. Despite progress in machine learning (ML)-based RAN control, the cyber threats and privacy risks to ML-driven xApps, rApps, and closed-loop O-RAN system are largely unexplored. The 11 Working Groups of the O-RAN Alliance have identified key ML-related security issues in the near-RT RIC architecture, relevant interfaces, xApps, and APIs [29]. Although prior work – see survey [30] – provides a detailed review of security and privacy risks in O-RAN, it lacks a comprehensive analysis and validation of how these threats could be executed. This gap underscores the need for further research and practical validation to enhance the security of O-RAN.

ML-Based xApps. Existing xApps use RNNs [31], DRL [32], and CNN-based spectrogram detectors [33], all trained on RAN data via the E2 Interface. However, none address security threats like data poisoning or intrusion that could disrupt the control loop and degrade performance. Hence, O-RAN benchmarking frameworks like AdvO-RAN are essential for training and testing DRL models to ensure robust xApps perform reliably in adversarial environment.

Adversarial DRL. The vulnerability in DRL has been explored in [15], where the authors proposed a method to uniformly attack the state space. Additional state-space attacks and defense strategies mostly assume white-box settings [34, 35]. A number of black-box approaches [36] attempt to infer agent action sequences to identify optimal attack points.In [37] the authors constrained the action space by spatio-temporal criteria and showed vulnerabilities in cyber-physical systems [38] and temporally perturbed both action and state space [37]. Existing adversarial approaches primarily focus on degrading the victim agent's reward, which may result in suboptimal attacks in the context of O-RAN, where the primary objective of the network is to provide better QoS rather than maximizing reward.

7 Limitations and Future Work

Our study has several limitations as follows: (1) To adapt to a new traffic patterns, the DRL agents in the xApps should be retrained or fine-tuned with the updated KPM distributions. Similarly, the attacker must realign its policy with the fine-tuned slicing agent. (2) Our reward design assumes fixed, noise-free SLA thresholds when deriving preference labels. In practice, these thresholds can drift or become noisy as traffic pattern changes, potentially mis-aligning the learned preferences. Developing attacker and defender policies that remain reliable under dynamic or noisy SLAs is a promising direction for future work.

8 Concluding Remarks

We presented AdvO-RAN, a framework to mitigate adversarial threats to DRL-based xApps in AI-driven O-RAN. AdvO-RAN uses preference-based reward modeling to strengthen SLA compliance and provides an O-RAN-compliant platform for developing, training, and benchmarking xApps under evasion attacks. Prototyped on the Colosseum emulator, AdvO-RAN reduced SLA violations from 44% to 27% on average, reduce latency by up to 46% under the strongest attacks, and recovered up to 75% of median end-to-end eMBB throughput in the most challenging scenarios.

Acknowledgement

This work has been supported by the National Science Foundation under grants CNS-2312875 and OAC-2530896; by the Air Force Office of Scientific Research under grant FA9550-23-1-0261; and by the Office of Naval Research under grant N00014-23-1-2221.

References

[1] Michele Polese, Leonardo Bonati, Salvatore D'Oro, Stefano Basagni, and Tommaso Melodia. Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges. *IEEE Communications Surveys & Tutorials*, 2023.

[2] Luca Baldesi, Francesco Restuccia, and Tommaso Melodia. ChARM: NextG Spectrum Sharing Through Data-Driven Real-Time O-RAN Dynamic Control. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, May 2022.

[3] Leonardo Bonati, Salvatore D'Oro, Michele Polese, Stefano Basagni, and Tommaso Melodia. Intelligence and Learning in O-RAN for Data-driven NextG Cellular Networks. *IEEE Communications Magazine*, 59(10):21–27, October 2021.

[4] Joshua Groen, Salvatore D'Oro, Utku Demir, Leonardo Bonati, Michele Polese, Tommaso Melodia, and Kaushik Chowdhury. Implementing and evaluating security in o-ran: Interfaces, intelligence, and platforms. *IEEE Network*, 2024.

[5] Joshua Groen, Salvatore D'Oro, Utku Demir, Leonardo Bonati, Davide Villa, Michele Polese, Tommaso Melodia, and Kaushik Chowdhury. Securing o-ran open interfaces. *IEEE Transactions on Mobile Computing*, pages 1–13, 2024.

[6] Hoa TT Nguyen, Minh T Nguyen, Hai T Do, Hoang T Hua, and Cuong V Nguyen. Drl-based intelligent resource allocation for diverse qos in 5g and toward 6g vehicular networks: a comprehensive survey. *Wireless Communications and Mobile Computing*, 2021(1):5051328, 2021.

[7] Yared Abera Ergu, Van-Linh Nguyen, Ren-Hung Hwang, Ying-Dar Lin, Chuan-Yu Cho, and Hui-Kuo Yang. Unmasking vulnerabilities: Adversarial attacks against drl-based resource allocation in o-ran. In *ICC 2024-IEEE International Conference on Communications*, pages 2378–2383. IEEE, 2024.

[8] Naveen Naik Sapavath, Brian Kim, Kaushik Chowdhury, and Vijay K Shah. Experimental study of adversarial attacks on ml-based xapps in o-ran. In *GLOBECOM 2023-2023 IEEE Global Communications Conference*, pages 6352–6357. IEEE, 2023.

[9] Azuka Chiejina, Brian Kim, Kaushik Chowhdury, and Vijay K Shah. System-level analysis of adversarial attacks and defenses on intelligence in o-ran based cellular networks. In *Proceedings of the 17th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 237–247, 2024.

[10] Arjun Balasingam, Manikanta Kotaru, and Paramvir Bahl. {Application-Level} service assurance with 5g {RAN} slicing. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 841–857, 2024.

[11] Leonardo Bonati, Pedram Johari, Michele Polese, Salvatore D'Oro, Subhramoy Mohanti, Miead Tehrani-Moayyed, Davide Villa, Shweta Shrivastava, Chinenye Tassie, Kurt Yoder, et al. Colosseum: Large-scale wireless experimentation through hardware-in-the-loop network emulation. In *2021 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pages 105–113. IEEE, 2021.

[12] Michele Polese, Leonardo Bonati, Salvatore D'Oro, Stefano Basagni, and Tommaso Melodia. ColO-RAN: Developing Machine Learning-based xApps for Open RAN Closed-loop Control on Programmable Experimental Platforms. *IEEE Transactions on Mobile Computing*, pages 1–14, July 2022.

[13] Maria Tsampazi, Salvatore D'Oro, Michele Polese, Leonardo Bonati, Gwenael Poitau, Michael Healy, and Tommaso Melodia. A comparative analysis of deep reinforcement learning-based xapps in o-ran. In *GLOBECOM 2023-2023 IEEE Global Communications Conference*, pages 1638–1643. IEEE, 2023.

[14] Michele Polese, Leonardo Bonati, Salvatore D'Oro, Stefano Basagni, and Tommaso Melodia. Colo-ran: Developing machine learning-based xapps for open ran closed-loop control on programmable experimental platforms. *IEEE Transactions on Mobile Computing*, 22(10):5787–5800, 2022.

[15] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

[16] Anay Pattanaik, Zhenyi Tang, Shuijing Liu, Gautham Bommannan, and Girish Chowdhary. Robust deep reinforcement learning with adversarial attacks. *arXiv preprint arXiv:1712.03632*, 2017.

[17] Huan Zhang, Hongge Chen, Chaowei Xiao, Bo Li, Mingyan Liu, Duane Boning, and Cho-Jui Hsieh. Robust deep reinforcement learning against adversarial perturbations on state observations. *Advances in Neural Information Processing Systems*, 33:21024–21037, 2020.

[18] Yanchao Sun, Ruijie Zheng, Yongyuan Liang, and Furong Huang. Who is the strongest enemy? towards optimal and efficient evasion attacks in deep rl. *arXiv preprint arXiv:2106.05087*, 2021.

[19] Kimin Lee, Laura Smith, and Pieter Abbeel. Pebble: Feedback-efficient interactive reinforcement learning via relabeling experience and unsupervised pre-training. *arXiv preprint arXiv:2106.05091*, 2021.

[20] Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

[21] Kimin Lee, Laura M Smith, and Pieter Abbeel. PEBBLE: Feedback-Efficient Interactive Reinforcement Learning via Relabeling Experience and Unsupervised Pre-training. In *International Conference on Machine Learning*. PMLR, 2021.

[22] Huan Zhang, Hongge Chen, Duane Boning, and Cho-Jui Hsieh. Robust reinforcement learning on state observations with learned optimal adversary. *arXiv preprint arXiv:2101.08452*, 2021.

[23] Davide Villa, Miead Tehrani-Moayyed, Clifton Paul Robinson, Leonardo Bonati, Pedram Johari, Michele Polese, and Tommaso Melodia. Colosseum as a digital twin: Bridging real-world experimentation and wireless network emulation. *IEEE Transactions on Mobile Computing*, pages 1–17, 2024.

[24] Leonardo Bonati, Salvatore D'Oro, Stefano Basagni, and Tommaso Melodia. SCOPE: An Open and Softwarized Prototyping Platform for NextG Systems. In *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 415–426, 2021.

[25] OpenCelliD. OpenCelliD. https://opencellid.org/, 2023. [Online; accessed 2023].

[26] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[27] Tuomas Oikarinen, Wang Zhang, Alexandre Megretski, Luca Daniel, and Tsui-Wei Weng. Robust deep reinforcement learning through adversarial loss. *Advances in Neural Information Processing Systems*, 34:26156–26167, 2021.

[28] Huan Zhang, Hongge Chen, Chaowei Xiao, Bo Li, Mingyan Liu, Duane Boning, and Cho-Jui Hsieh. Robust deep reinforcement learning against adversarial perturbations on state observations. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21024–21037. Curran Associates, Inc., 2020.

[29] O-RAN Alliance. The o-ran alliance security working group continues to advance o-ran security. https://www.o-ran.org/blog/the-o-ran-alliance-security-working-group-continues-to-advance-o-ran-security, June 9 2023.

[30] Madhusanka Liyanage, An Braeken, Shahriar Shahabuddin, and Pasika Ranaweera. Open ran security: Challenges and opportunities. *Journal of Network and Computer Applications*, 214:103621, 2023.

[31] Guillem Reus-Muns, Pratheek S Upadhyaya, Utku Demir, Nathan Stephenson, Nasim Soltani, Vijay K Shah, and Kaushik R Chowdhury. Senseoran: O-RAN based radar detection in the cbrs band. *IEEE Journal on Selected Areas in Communications*, 2023.

[32] Leonardo Bonati, Salvatore D'Oro, Michele Polese, Stefano Basagni, and Tommaso Melodia. Intelligence and learning in o-ran for data-driven nextg cellular networks. *IEEE Communications Magazine*, 59(10):21–27, 2021.

[33] Andrea Lacava, Michele Polese, Rajarajan Sivaraj, Rahul Soundrarajan, Bhawani Shanker Bhati, Tarunjeet Singh, Tommaso Zugno, Francesca Cuomo, and Tommaso Melodia. Programmable and customized intelligence for traffic steering in 5g networks using open ran architectures. *IEEE Transactions on Mobile Computing*, 2023.

[34] Ajay Mandlekar, Yuke Zhu, Animesh Garg, Li Fei-Fei, and Silvio Savarese. Adversarially robust policy learning: Active construction of physically-plausible perturbations. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3932–3939. IEEE, 2017.

[35] Jianwen Sun, Tianwei Zhang, Xiaofei Xie, Lei Ma, Yan Zheng, Kangjie Chen, and Yang Liu. Stealthy and efficient adversarial attacks against deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5883–5891, 2020.

[36] Yiren Zhao, Ilia Shumailov, Han Cui, Xitong Gao, Robert Mullins, and Ross Anderson. Blackbox attacks on reinforcement learning agents using approximated temporal information. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 16–24. IEEE, 2020.

[37] Chen Tessler, Yonathan Efroni, and Shie Mannor. Action robust reinforcement learning and applications in continuous control. In *International Conference on Machine Learning*, pages 6215–6224. PMLR, 2019.

[38] Xian Yeow Lee, Sambit Ghadai, Kai Liang Tan, Chinmay Hegde, and Soumik Sarkar. Spatiotemporally constrained action space attacks on deep reinforcement learning agents. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 4577–4584, 2020.