Efficient Adversarial Training without Attacking: Worst-Case-Aware Robust Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Recent studies reveal that a well-trained deep reinforcement learning (RL) policy can be particularly vulnerable to adversarial perturbations on input observations. Therefore, it is crucial to train RL agents that are robust against any attacks with a bounded budget. Existing robust training methods in deep RL either treat correlated steps separately, ignoring the robustness of long-term reward, or train the agents and RL-based attacker together, doubling the computational burden and sample complexity of the training process. In this work, we propose a strong and efficient robust training framework for RL, named Worst-case-aware Robust RL (WocaR-RL), that directly estimates and optimizes the worst-case reward of a policy under bounded ℓ_p attacks without requiring extra samples for learning an attacker. Experiments on multiple environments show that WocaR-RL achieves state-of-the-art performance under various strong attacks, and obtains significantly higher training efficiency than prior state-of-the-art robust training methods.

1 Introduction

Deep reinforcement learning (DRL) has achieved impressive results by using deep neural networks (DNN) to learn complex policies in high-dimensional environments. However, well-trained DNNs may drastically fail under adversarial perturbations of the input [1, 5]. Therefore, before deploying DRL policies to real-life applications, there is a crucial need to improve the robustness of deep policies against adversarial attacks, especially worst-case attacks that maximally depraves the performance of trained agents [41].

A line of regularization-based robust methods [52, 32, 39] focuses on improving the robustness of the DNN itself and regularizes the policy network to output similar actions under bounded state perturbations. However, different from supervised learning problems, the vulnerability of a deep policy comes not only from the DNN approximator, but also from the dynamics of the RL environment [50]. These regularization-based methods neglect the intrinsic vulnerability of policies under the environment dynamics, and thus may still fail under strong attacks [41]. For example, in the go-home task shown in Figure 1, both the green policy and the red policy arrive home without rock collision, when there is no attack. However, although regularization-based methods may ensure a minor action change under a state perturbation, the red policy may still be susceptible to a low reward under attacks, as a very small divergence can lead it to the bomb. On the contrary, the green policy is more robust to adversarial attacks since it stays away from the bomb. Therefore, besides promoting the robustness of DNN approximators (such as the policy network), it is also important to learn a policy with stronger intrinsic robustness.

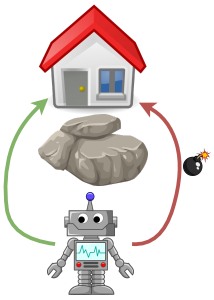


Figure 1: Policies have different vulnerabilities.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

There is another line of work considering the long-term robustness of a deep policy under strong adversarial attacks. In particular, it is theoretically proved [52, 41] that the strongest (worst-case) attacker against a policy can be learned as an RL problem, and training the agent under such a learned attacker can result in a robust policy. Zhang et al. [50] propose the *Alternating Training with Learned Adversaries (ATLA)* framework, which alternately trains an RL agent and an RL attacker. Sun et al. [41] further propose PA-ATLA, which alternately trains an agent and the proposed more efficient PA-AD RL attacker, obtaining state-of-the-art robustness in many MuJoCo environments. However, training an RL attacker requires extra samples from the environment, and the attacker's RL problem may even be more difficult and sample expensive to solve than the agent's original RL problem [50, 41], especially in large-scale environments such as Atari games with pixel observations. Therefore, although ATLA and PA-ATLA are able to achieve high long-term reward under attacks, they double the computational burden and sample complexity to train the robust agent.

The above analysis of existing literature suggests two main challenges in improving the adversarial robustness of DRL agents: (1) correctly characterizing the long-term reward vulnerability of an RL policy, and (2) efficiently training a robust agent without requiring much more effort than vanilla training. To tackle these challenges, in this paper, we propose a generic and efficient robust training framework named *Worst-case-aware Robust RL (WocaR-RL)* that estimates and improves the long-term robustness of an RL agent.

WocaR-RL has 3 key mechanisms. First, WocaR-RL introduces a novel *worst-attack Bellman operator* which uses existing off-policy samples to estimate the lower bound of the policy value under the worst-case attack. Compared to prior works [50, 41] which attempt to learn the worst-case attack by RL methods, WocaR-RL does not require any extra interaction with the environment. Second, using the estimated worst-case policy value, WocaR-RL optimizes the policy to select actions that not only achieve high natural future reward, but also achieve high worst-case reward when there are adversarial attacks. Therefore, WocaR-RL learns a policy with less intrinsic vulnerability. Third, WocaR-RL regularizes the policy network with a carefully designed state importance weight. As a result, the DNN approximator tolerates state perturbations, especially for more important states where decisions are crucial for future reward. The above 3 mechanisms can also be interpreted from a geometric perspective of adversarial policy learning, as detailed in Appendix B.

Our **contributions** can be summarized as below. **(1)** We provide an approach to estimate the worst-case value of any policy under any bounded ℓ_p adversarial attacks. This helps evaluate the robustness of a policy without learning an attacker which requires extra samples and exploration. **(2)** We propose a novel and principled robust training framework for RL, named *Worst-case-aware Robust RL (WocaR-RL)*, which characterizes and improves the worst-case robustness of an agent. WocaR-RL can be used to robustify existing DRL algorithms (e.g. PPO [38], DQN [31]). **(3)** We show by experiments that WocaR-RL achieve **improved robustness** against various adversarial attacks as well as **higher efficiency**, compared with state-of-the-art (SOTA) robust RL methods in many MuJoCo and Atari games. For example, compared to the SOTA algorithm PA-ATLA-PPO [41] in the Walker environment, we obtain 20% more worst-case reward (under the strongest attack algorithm), with only about 50% training samples and 50% running time. Moreover, WocaR-RL learns **more interpretable "robust behaviors"** than PA-ATLA-PPO in Walker as shown in Figure 2.

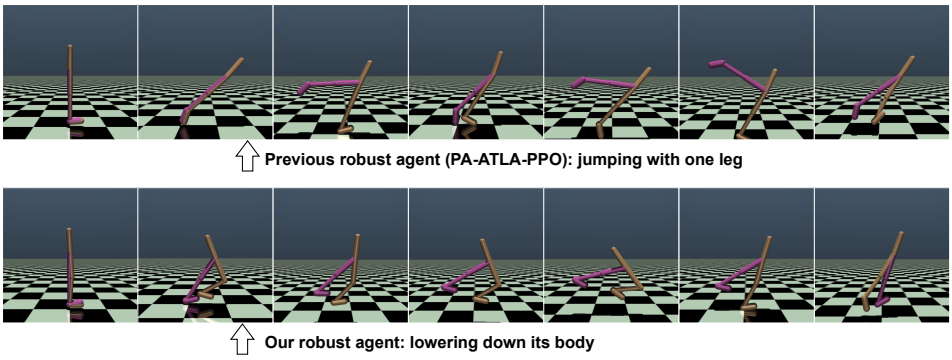


Figure 2: The robust Walker agents trained with **(top)** the state-of-the-art method PA-ATLA-PPO [41] and **(bottom)** our WocaR-RL. Although PA-ATLA-PPO agent also achieves high reward under attacks, it learns to jump with one leg, which is counter-intuitive and may indicate some level of overfitting to a specific attacker. In contrast, our WocaR-RL agent learns to lower down its body, which is more intuitive and interpretable. The full agent trajectories in Walker and other environments are provided in supplementary materials as GIF figures.

2 Related Work

Defending against Adversarial Perturbations on State Observations. **(1)** *Regularization-based methods* [52, 39, 32] enforce the policy to have similar outputs under similar inputs, which achieves certifiable performance for DQN in some Atari games. But in continuous control tasks, these methods may not reliably improve the worst-case performance. A recent work by Korkmaz [20] points out that these adversarially trained models may still be sensible to new perturbations. **(2)** *Attack-driven methods* train DRL agents with adversarial examples. Some early works [21, 4, 28, 33] apply weak or strong gradient-based attacks on state observations to train RL agents against adversarial perturbations. Zhang et al. [50] propose Alternating Training with Learned Adversaries (ATLA), which alternately trains an RL agent and an RL adversary and significantly improves the policy robustness in continuous control games. Sun et al. [41] further extend this framework to PA-ATLA with their proposed more advanced RL attacker PA-AD. Although ATLA and PA-ATLA achieve strong empirical robustness, they require training an extra RL adversary that can be computationally and sample expensive. **(3)** There is another line of work studying *certifiable robustness* of RL policies. Several works [26, 32, 8] computed lower bounds of the action value network Q^π to certify robustness of action selection at every step. However, these bounds do not consider the distribution shifts caused by attacks, so some actions that appear safe for now can lead to extremely vulnerable future states and low long-term reward under future attacks. Moreover, these methods cannot apply to continuous action spaces. Kumar et al. and Wu et al.[22, 48] both extend randomized smoothing [6] to derive robustness certificates for trained policies. But these works mostly focus on theoretical analysis, and effective robust training approaches rather than robust training.

Adversarial Defenses against Other Adversarial Attacks. Besides observation perturbations, attacks can happen in many other scenarios. For example, the agent's executed actions can be perturbed [49, 43, 44, 23]. Moreover, in a multi-agent game, an agent's behavior can create adversarial perturbations to a victim agent [12]. Pinto et al. [34] model the competition between the agent and the attacker as a zero-sum two-player game, and train the agent under a learned attacker to tolerate both environment shifts and adversarial disturbances. We point out that although we mainly consider state adversaries, our WocaR-RL can be extended to action attacks as formulated in Appendix C.5. Note that we focus on robustness against test-time attacks, different from poisoning attacks which alter the RL training process [3, 19, 40, 54, 35].

Safe RL and Risk-sensitive RL. There are several lines of work that study RL under safety/risk constraints [17, 10, 9, 2, 45] or under intrinsic uncertainty of environment dynamics [25, 29]. However, these works do not deal with adversarial attacks, which can be adaptive to the learned policy. More comparison between these methods and our proposed method is discussed in Section 4.

3 Preliminaries and Background

Reinforcement Learning (RL). An RL environment is modeled by a Markov Decision Process (MDP), denoted by a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, where $\mathcal{S}$ is a state space, $\mathcal{A}$ is an action space, $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is a stochastic dynamics model[1], $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function and $\gamma \in [0, 1)$ is a discount factor. An agent takes actions based on a policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$. For any policy, its *natural performance* can be measured by the value function $V^\pi(s) := \mathbb{E}_{P,\pi}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s]$, and the action value function $Q^\pi(s, a) := \mathbb{E}_{P,\pi}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s, a_0 = a]$. We call V^π the *natural value* and Q^π the *natural action value* in contrast to the values under attacks, as will be introduced in Section 4.

Deep Reinforcement Learning (DRL). In large-scale problems, a policy can be parameterized by a neural network. For example, value-based RL methods (e.g. DQN [31]) usually fit a Q network and take the greedy policy $\pi(s) = \mathrm{argmax}_a Q(s, a)$. In actor-critic methods (e.g. PPO [38]), the learner directly learns a policy network and a critic network. In practice, an agent usually follows a stochastic policy during training that enables exploration, and executes a trained policy deterministically in test-time, e.g. the greedy policy learned with DQN. Throughout this paper, we use π_θ to denote the training-time stochastic policy parameterized by θ, while π denotes the trained deterministic policy that maps a state to an action.

Test-time Adversarial Attacks. After training, the agent is deployed into the environment and executes a pre-trained fixed policy π. An attacker/adversary, during the deployment of the agent, may

[1] $\Delta(\mathcal{X})$ denotes the space of probability distributions over $\mathcal{X}$.

perturb the state observation of the agent/victim at every time step with a certain attack budget ϵ. Note that the attacker only perturbs the inputs to the policy, and the underlying state in the environment does not change. This is a realistic setting because real-world observations can come from noisy sensors or be manipulated by malicious attacks. For example, an auto-driving car receives sensory observations; an attacker may add imperceptible noise to the camera, or perturb the GPS signal, although the underlying environment (the road) remains unchanged. In this paper, we consider the ℓ_p *thread model* which is widely used in adversarial learning literature: at step t, the attacker alters the observation s_t into $\tilde{s}_t \in \mathcal{B}_\epsilon(s_t)$, where $\mathcal{B}_\epsilon(s_t)$ is a ℓ_p norm ball centered at s_t with radius ϵ. The above setting (ℓ_p constrained observation attack) is the same with many prior works [18, 33, 52, 50, 41].

4 Worst-case-aware Robust RL

In this section, we present *Worst-case-aware Robust RL (WocaR-RL)*, a generic framework that can be fused with any DRL approach to improve the adversarial robustness of an agent. We will introduce the three key mechanisms in WocaR-RL: worst-attack value estimation, worst-case-aware policy optimization, and value-enhanced state regularization, respectively. Then, we will illustrate how to incorporate these mechanisms into existing DRL algorithms to improve their robustness.

Mechanism 1: Worst-attack Value Estimation

Traditional RL aims to learn a policy with the maximal value V^π. However, in a real-world problem where observations can be noisy or even adversarially perturbed, it is not enough to only consider the natural value V^π and Q^π. As motivated in Figure 1, two policies with similar natural rewards can get totally different rewards under attacks. To comprehensively evaluate how good a policy is in an adversarial scenario and to improve its robustness, we should be aware of the lowest possible long-term reward of the policy when its observation is adversarially perturbed with a certain attack budget ϵ at every step (with an ℓ_p attack model introduced in Section 3).

The worst-case value of a policy is, by definition, the cumulative reward obtained under the optimal attacker. As justified by prior works [52, 41], for any given victim policy π and attack budget $\epsilon > 0$, there exists an optimal attacker, and finding the optimal attacker is equivalent to learning the optimal policy in another MDP. We denote the optimal (deterministic) attacker's policy as h^*. However, learning such an optimal attacker by RL algorithms requires extra interaction samples from the environment, due to the unknown dynamics. Moreover, learning the attacker by RL can be hard and expensive, especially when the state observation space is high-dimensional.

Instead of explicitly learning the optimal attacker with a large amount of samples, we propose to directly estimate the worst-case cumulative reward of the policy by characterizing the vulnerability of the given policy. We first define the *worst-attack action value* of policy π as $\underline{Q}^\pi(s,a) := \mathbb{E}_P[\sum_{t=0}^{\infty} \gamma^t R\left(s_t, \pi(h^*(s_t))\right) \mid s_0 = s, a_0 = a]$. The *worst-attack value* $\underline{V}^\pi$ can be defined using h^* in the same way, as shown in Definition A.1 in Appendix A. Then, we introduce a novel operator $\underline{\mathcal{T}}^\pi$, namely the *worst-attack Bellman operator*, defined as below.

Definition 4.1 (Worst-attack Bellman Operator)**.** For MDP $\mathcal{M}$, given a fixed policy π and attack radius ϵ, define the worst-attack Bellman operator $\underline{\mathcal{T}}^\pi$ as

$$\left(\underline{\mathcal{T}}^\pi Q\right)(s,a) := \mathbb{E}_{s' \sim P(s,a)}[R(s,a) + \gamma \min_{a' \in \mathcal{A}_{\text{adv}}(s',\pi)} Q\left(s', a'\right)], \tag{1}$$

where $\forall s \in \mathcal{S}$, $\mathcal{A}_{\text{adv}}(s,\pi)$ is defined as

$$\mathcal{A}_{\text{adv}}(s,\pi) := \{a \in \mathcal{A} : \exists \tilde{s} \in \mathcal{B}_\epsilon(s) \text{ s.t. } \pi(\tilde{s}) = a\}. \tag{2}$$

Here $\mathcal{A}_{\text{adv}}(s',\pi)$ denotes the set of actions an adversary can mislead the victim π into selecting by perturbing the state s' into a neighboring state $\tilde{s} \in \mathcal{B}_\epsilon(s')$. This hypothetical perturbation to the *future* state s' is the key for characterizing the worst-case long-term reward under attack. The following theorem associates the worst-attack Bellman operator and the worst-attack action value.

Theorem 4.2 (Worst-attack Bellman Operator and Worst-attack Action Value)**.** *For any given policy π, $\underline{\mathcal{T}}^\pi$ is a contraction whose fixed point is $\underline{Q}^\pi$, the worst-attack action value of π under any ℓ_p observation attacks with radius ϵ.*

Theorem 4.2 proved in Appendix A suggests that the lowest possible cumulative reward of a policy under bounded observation attacks can be computed by worst-attack Bellman operator. The corresponding worst-attack value $\underline{V}^\pi$ can be obtained by $\underline{V}^\pi(s) = \min_{a \in \mathcal{A}_{\text{adv}}(s,\pi)} \underline{Q}^\pi(s,a)$.

How to Compute $\mathcal{A}_{\text{adv}}$. To obtain $\mathcal{A}_{\text{adv}}(s,\pi)$, we need to identify the actions that can be the outputs of the policy π when the input state s is perturbed within $\mathcal{B}_\epsilon(s)$. This can be solved by

commonly-used convex relaxation of neural networks [14, 53, 47, 51, 13], where layer-wise lower and upper bounds of the neural network are derived. That is, we calculate $\overline{\pi}$ and $\underline{\pi}$ such that $\overline{\pi}(s) \geq \pi(\hat{s}) \geq \underline{\pi}(s), \forall \hat{s} \in \mathcal{B}_\epsilon(s)$. With such a relaxation, we can obtain a superset of $\mathcal{A}_{\text{adv}}$, namely $\hat{\mathcal{A}}_{\text{adv}}$. Then, the fixed point of Equation (1) with $\mathcal{A}_{\text{adv}}$ being replaced by $\hat{\mathcal{A}}_{\text{adv}}$ becomes a lower bound of the worst-attack action value. For a continuous action space, $\hat{\mathcal{A}}_{\text{adv}}(s, \pi)$ contains actions bounded by $\overline{\pi}(s)$ and $\underline{\pi}(s)$. For a discrete action space, we can first compute the maximal and minimal probabilities of taking each action, and derive the set of actions that are likely to be selected. The computation of $\hat{\mathcal{A}}_{\text{adv}}$ is not expensive, as there are many efficient convex relaxation methods [30, 51] which compute $\overline{\pi}$ and $\underline{\pi}$ with only constant-factor more computations than directly computing $\pi(s)$. Experiment in Section 5 verifies the efficiency of our approach. More implementation details and explanations are provided in Appendix C.1.

Estimating Worst-attack Value. Note that the worst-attack Bellman operator $\underline{\mathcal{T}}^\pi$ is similar to the optimal Bellman operator $\mathcal{T}^*$, although it uses $\min_{a \in \mathcal{A}_{\text{adv}}}$ instead of $\max_{a \in \mathcal{A}}$. Therefore, once we identify $\mathcal{A}_{\text{adv}}$ as introduced above, it is straightforward to compute the worst-attack action value using Bellman backups. To model the worst-attack action value, we train a network named *worst-attack critic*, denoted by $\underline{Q}^\pi_\phi$, where ϕ is the parameterization. Concretely, for any mini-batch $\{s_t, a_t, r_t, s_{t+1}\}_{t=1}^N$, $\underline{Q}^\pi_\phi$ is optimized by minimizing the following estimation loss:

$$\mathcal{L}_{\text{est}}(\underline{Q}^\pi_\phi) := \frac{1}{N} \sum_{t=1}^{N} (\underline{y}_t - \underline{Q}^\pi_\phi(s_t, a_t))^2, \text{where } \underline{y}_t = r_t + \gamma \min_{\hat{a} \in \mathcal{A}_{\text{adv}}(s_{t+1}, \pi)} \underline{Q}^\pi_\phi(s_{t+1}, \hat{a}). \quad (3)$$

For a discrete action space, $\mathcal{A}_{\text{adv}}$ is a discrete set and solving $\underline{y}_t$ is straightforward. For a continuous action space, we use gradient descent to approximately find the minimizer $\hat{a}$. Since $\mathcal{A}_{\text{adv}}$ is in general small, this minimization is usually easy to solve. In MuJoCo, we find that 50-step gradient descent already converges to a good solution with little computational cost, as detailed in Appendix D.3.3.

Differences with Worst-case Value Estimation in Related Work. Our proposed worst-attack Bellman operator is different from the worst-case Bellman operator in the literature of risk-sensitive RL [17, 10, 42, 9, 2, 45], whose goal is to avoid unsafe trajectories under the intrinsic uncertainties of the MDP. These inherent uncertainties of the environment are independent of the learned policy. In contrast, our focus is to defend against adversarial perturbations created by malicious attackers that can be *adaptive* to the policy. The GWC reward proposed by [32] also estimates the worst-case reward of a policy under state perturbations. But their evaluation is based on a greedy strategy and requires interactions with the environment, which is different from our estimation.

Mechanism 2: Worst-case-aware Policy Optimization

So far we have introduced how to evaluate the worst-attack value of a policy by learning a worst-attack critic. Inspired by the actor-critic framework, where the actor policy network π_θ is optimized towards a direction that the critic value increases the most, we can regard worst-attack critic as a special critic that directs the actor to increase the worst-attack value. That is, we encourage the agent to select an action with a higher worst-attack action value, by minimizing the worst-attack policy loss below:

$$\mathcal{L}_{\text{wst}}(\pi_\theta; \underline{Q}^\pi_\phi) := -\frac{1}{N} \sum_{t=1}^{N} \sum_{a \in \mathcal{A}} \pi_\theta(a|s_t) \underline{Q}^\pi_\phi(s_t, a), \quad (4)$$

where $\underline{Q}^\pi_\phi$ is the worst-attack critic learned via $\mathcal{L}_{\text{est}}$ introduced in Equation (3). Note that $\mathcal{L}_{\text{wst}}$ is a general form, while the detailed implementation of the worst-attack policy optimization can vary depending on the architecture of π_θ in the base RL algorithm (e.g. PPO has a policy network, while DQN acts using the greedy policy induced by a Q network). In Appendix C.2 and Appendix C.3, we illustrate how to implement $\mathcal{L}_{\text{wst}}$ for PPO and DQN as two examples.

The proposed worst-case-aware policy optimization has several **merits** compared to prior ATLA [50] and PA-ATLA [41] methods which alternately train the agent and an RL attacker. **(1)** Learning the optimal attacker h^* requires collecting extra samples using the current policy (on-policy estimation). In contrast, $\underline{Q}^\pi_\phi$ can be learned using off-policy samples, e.g., historical samples in the replay buffer, and thus is more suitable for training where the policy changes over time. ($\underline{Q}^\pi_\phi$ depends on the current policy via the computation of $\mathcal{A}_{\text{adv}}$.) **(2)** We properly exploit the policy function that is being trained by computing the set of possibly selected actions $\hat{\mathcal{A}}_{\text{adv}}$ for any state. In contrast, ATLA [50] learns an attacker by treating the current policy as a black box, ignoring the intrinsic properties of the policy. PA-ATLA [41], although assumes white-box access to the victim policy, also needs to explore and learn from extra on-policy interactions. **(3)** The attacker trained with DRL methods, namely $\hat{h}^*$, is

not guaranteed to converge to an optimal solution, such that the performance of π estimated under $\hat{h}^*$ can be overly optimistic. Our estimation, as mentioned in Mechanism 1, computes a lower bound of $\underline{Q}^\pi$ and thus can better indicate the robustness of a policy.

Mechanism 3: Value-enhanced State Regularization

As discussed in Section 1, the vulnerability of a deep policy comes from both the policy's intrinsic vulnerability with the RL dynamics and the DNN approximator. The first two mechanisms of WocaR-RL mainly focus on the policy's intrinsic vulnerability, i.e., let the policy select actions that are less vulnerable to possible attacks in all future steps. However, if a bounded state perturbation can cause the network to output a very different action, then the $\mathcal{A}_{\text{adv}}$ set will be large and $\underline{Q}^\pi$ can thus be low. Therefore, it is also important to encourage the trained policy to output similar actions for the clean state s and any $\tilde{s} \in \mathcal{B}_\epsilon(s)$, as is done in prior work [52, 39, 8].

But different from these prior methods, we note that different states should be treated differently. Some states are "critical" where selecting a bad action will result in catastrophic consequences. For example, when the agent gets close to the bomb in Figure 1, we should make the network more resistant to adversarial state perturbations. To differentiate states based on their impacts on future reward, we propose to measure the importance of states with Definition 4.3 below.

Definition 4.3 (State Importance Weight)**.** Define state importance weight of $s \in \mathcal{S}$ for policy π as

$$w(s) = \max_{a_1 \in \mathcal{A}} Q^\pi(s, a_1) - \min_{a_2 \in \mathcal{A}} Q^\pi(s, a_2). \tag{5}$$

To justify whether Definition 4.3 can characterize state importance, we train a DQN network in an Atari game Pong, and show the states with the highest weight and the lowest weight in Figure 3, among many state samples. We can see that the state with higher weight in Figure 3(left) is indeed crucial for the game, as the green agent paddle is close to the ball. Conversely, a less-important state in Figure 3(right) does not have significantly different future rewards under different actions. Computing $w(s)$ is easy in a discrete action space, while in a continuous action space, one can use gradient descent to approximately find the maximal and the minimal Q values for a state.

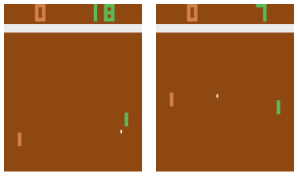


Figure 3: States in Pong with **(left)** high weight $w(s)$ and **(right)** low weight $w(s)$.

By incorporating the state importance weight $w(s)$, we regularize the policy network and let it pay more attention to more crucial states, by minimizing the following loss:

$$\mathcal{L}_{\text{reg}}(\pi_\theta) = \frac{1}{N} \sum_{t=1}^{N} w(s_t) \max_{\tilde{s}_t \in \mathcal{B}_\epsilon(s_t)} \text{Dist}(\pi_\theta(s_t), \pi_\theta(\tilde{s}_t)), \tag{6}$$

where Dist can be any distance measure between two distributions (e.g., KL-divergence). Minimizing $\mathcal{L}_{\text{reg}}$ can result in a smaller $\mathcal{A}_{\text{adv}}$, and thus the worst-attack value will be closer to the natural value.

WocaR-RL: A Generic Robust Training Framework

So far we have introduced three key mechanisms and their loss functions, $\mathcal{L}_{\text{est}}$ in Equation (3), $\mathcal{L}_{\text{wst}}$ in Equation (4) and $\mathcal{L}_{\text{reg}}$ in Equation (6). Then, our robust training framework WocaR-RL combines these losses with any base RL algorithm. To be more specific, as shown in Figure 4, for any base RL algorithm that trains policy π_θ using loss $\mathcal{L}_{\text{RL}}$, we learn an extra worst-attack critic network $\underline{Q}^\pi_\phi$ by minimizing

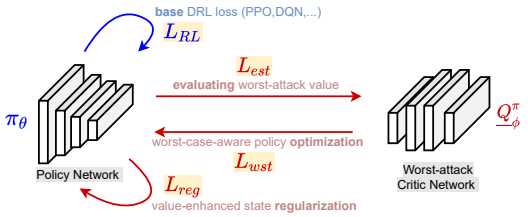


Figure 4: Training architecture of WocaR-RL. (Components proposed in this paper are colored as red.)

$$\mathcal{L}_{\underline{Q}^\pi_\phi} := \mathcal{L}_{\text{est}}(\underline{Q}^\pi_\phi), \tag{7}$$

and combine $\mathcal{L}_{\text{wst}}$ and $\mathcal{L}_{\text{reg}}$ with $\mathcal{L}_{\text{RL}}$ to optimize π_θ by minimizing

$$\mathcal{L}_{\pi_\theta} := \mathcal{L}_{\text{RL}}(\pi_\theta) + \kappa_{\text{wst}} \mathcal{L}_{\text{wst}}(\pi_\theta; \underline{Q}^\pi_\phi) + \kappa_{\text{reg}} \mathcal{L}_{\text{reg}}(\pi_\theta), \tag{8}$$

where κ_{wst} and κ_{reg} are hyperparameters balancing between natural performance and robustness. Note that $\underline{Q}^\pi_\phi$ is trained together but independently with π_θ using historical transition samples, so WocaR-RL does not require extra samples from the environment. WocaR-RL can also be interpreted from a geometric perspective based on prior RL polytope theory [7, 41] as detailed in Appendix B.

Our WocaR-RL is a generic robust training framework that can be used to robustify existing DRL algorithms. We provide two case studies: **(1)** combining WocaR-RL with a policy-based algo-

rithm PPO [38], namely *WocaR-PPO*, and **(2)** combining WocaR-RL with a value-based algorithm DQN [31], namely *WocaR-DQN*. The pseudocodes of WocaR-PPO and WocaR-DQN are illustrated in Appendix C.2 and Appendix C.3. The application of WocaR-RL to other DRL methods is then straightforward, since most DRL methods are either policy-based or value-based. Next, we show by experiments that WocaR-PPO and WocaR-DQN achieve state-of-the-art robustness with superior efficiency, in various continuous control tasks and video game environments. We also empirically verify the effectiveness of each of the 3 mechanisms of WocaR-RL in Section 5.2.

5 Experiments and Discussion

In this section, our experimental evaluations on various MuJoCo and Atari environments aim to study the following questions: **(1)** Can WocaR-RL learn policies with better **robustness** under existing strong adversarial attacks? **(2)** Can WocaR-RL maintain **natural performance** when improving robustness? **(3)** Can WocaR-RL learn more **efficiently** during robust training? **(4)** Is each mechanism in WocaR-RL **effective**? Problem (1), (2) and (3) are answered in Section 5.1 with detailed empirical results, and problem (4) is studied in Section 5.2 via ablation experiments.

5.1 Experiments and Evaluations

Environments. Following most prior works [52, 50, 32] and the released implementation, we apply our WocaR-RL to PPO [38] on 4 MuJoCo tasks with continuous action spaces, including Hopper, Walker2d, Halfcheetah and Ant, and to DQN [31] agents on 4 Atari games including Pong, Freeway, BankHeist and RoadRunner, which have high dimensional pixel inputs and discrete action spaces.

Baselines and Implementation. We compare our algorithm with several state-of-the-art robust training methods, including (1) *SA-PPO/SA-DQN* [52]: regularizing policy networks by convex relaxation. (2) *ATLA-PPO* [50]: alternately training an agent and an RL attacker. (3) *PA-ATLA-PPO* [41]: alternately training an agent and a more advanced RL attacker PA-AD. (4) *RADIAL-PPO/RADIAL-DQN* [32]: optimizing policy network by designed adversarial loss functions based on robustness bounds. SA and RADIAL have both PPO and DQN versions, which are compared with our WocaR-PPO and WocaR-DQN. But ATLA and PA-ATLA do not provide DQN versions, since alternately training on DQN can be expensive as explained in the original papers [41]. (PA-ATLA has an A2C version, which we compare in Appendix D.2.) Therefore, we reproduce their ATLA-PPO and PA-ATLA-PPO results and compare them with our WocaR-PPO. More implementation and hyperparameter details are provided in Appendix D.1.

Case I: Robust PPO for MuJoCo Continuous Control

Evaluation Metrics. To reflect both the natural performance and robustness of trained agents, we report the average episodic rewards under no attack and against various attacks. For a comprehensive robustness evaluation, we attack the trained robust models with multiple existing attack methods, including: (1) *MaxDiff* [52] (maximal action difference), (2) *Robust Sarsa (RS)* [52] (attacking with a robust action-value function), (3) *SA-RL* [52] (finding the optimal state adversary) and (4) *PA-AD* [41] (the *existing strongest attack* by learning the optimal policy adversary with RL). For a clear comparison, we use the same attack radius ϵ as in most baselines [52, 50, 41].

Performance and Robustness of WocaR-PPO Figure 5 (left four columns) shows performance curves during training under four different adversarial attacks. Among all four attack algorithms, WocaR-PPO converges much faster than baselines, and often achieves the best asymptotic robust performance, especially under the strongest PA-AD attack. It is worth emphasizing that since we train a robust agent without explicitly learning an RL attacker, our method not only obtains stronger robustness and much higher efficiency, but also a more general defense: WocaR-PPO obtains comprehensively superior performance against a variety of attacks compared against existing SOTA algorithms based on learned attackers (ATLA-PPO, PA-ATLA-PPO). Additionally, in our experiments, WocaR-PPO learns relatively more universal defensive behaviors as shown in Figure 2, which can physically explain why our algorithm can defend against diverse attacks. We provide policy demonstrations in multiple tasks in our supplementary materials.
The comparison of natural performance and the worst-case performance appears in Figure 5 (right). We see that WocaR-PPO maintains competitive natural rewards under no attack compared with other baselines, which demonstrates that our algorithm gains more robustness without losing too much natural performance. The full results of baselines and our algorithm under different attack evaluations are provided by Table 2 in Appendix D.2 (including performance under random attacks).

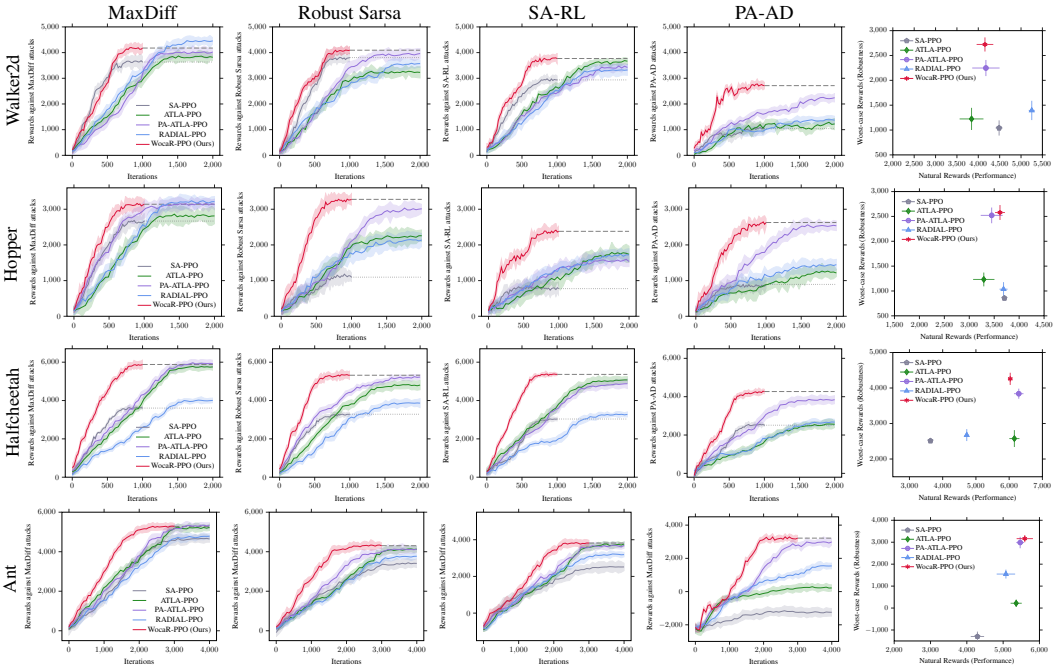


Figure 5: Robustness, Efficiency and High Natural Performance of WocaR-PPO. (Left four columns) Learning curves of rewards under MaxDiff, Robust Sarsa, SA-RL and PA-AD *(the strongest)* attacks during training on four environments. **(Rightmost column)** Average episode natural rewards v.s. average worst rewards under attacks. Each row shows the performance of baselines and WocaR-PPO on one environment. Shaded regions are computed over 20 random seeds. Results under more attack radius ϵ's are in Appendix D.3.1.

Efficiency of Training WocaR-PPO. The learning curves in Figure 5 (left) directly show the sample efficiency of WocaR-PPO. Following the optimal settings provided in [52, 50, 32], our method takes *50% training steps* required by RADIAL-PPO and ATLA methods on Hopper, Walker2d, and Halfcheetah because RADIAL-PPO needs more steps to ensure convergence and ATLA methods require additional adversary training steps. When solving high dimensional environments like Ant, WocaR-PPO only requires *75% steps* compared with all other baselines to converge. We also provide additional results of baselines using the same training steps as WocaR-PPO in Appendix D.3.2.
In terms of time efficiency, WocaR-PPO saves *50% training time* for convergence on Hopper, Walker2d, and Halfcheetah, and *32% time* on Ant compared with the SOTA method. Therefore, *WocaR-PPO achieves both higher computational efficiency and higher sample efficiency than SOTA baselines.* Detailed costs in time and sampling are in Appendix D.3.3.

Case II: Robust DQN for Atari Video Games

Evaluation Metrics. Since Atari games have pixel state spaces and discrete action spaces, the applicable attacking algorithms also differ from those in MuJoCo tasks. We include the following common attacks: (1) 10-step untargeted *PGD* (projected gradient descent) attack, (2) *MinBest* [18], which minimizes the probability of choosing the "best" action, (3) *PA-AD* [41], as the state-of-the-art RL-based adversarial attack algorithm.

Performance and Robustness of WocaR-DQN. Table 1 presents the results on four Atari games under attack radius $\epsilon = 3/255$, while results and analysis under smaller attack radius $1/255$ are in Appendix D.2. We can see that *our WocaR-DQN consistently outperforms baselines under MinBest and PA-AD attacks in all environments, with a significant advance under the strongest (worst-case) PA-AD attacks compared with other robust agents.* Under PGD attacks, WocaR-DQN performs comparably with the state-of-the-art in Freeway and Pong (which are simpler games) and gains higher rewards than other agents in BankHeist and Roadrunner. Since SA-DQN and RADIAL-DQN focus on bounding and smoothing the policy network and do not consider the policy's intrinsic vulnerability, they are robust under the PGD attack but still vulnerable against the stronger PA-AD attack.

Efficiency of Training WocaR-DQN. The total training time for SA-DQN, RADIAL-DQN, and our WocaR-DQN are roughly 35, 17, and 18 hours, respectively. All baselines are trained for 6 million frames on the same hardware. Therefore, WocaR-DQN is 49% faster (and is more robust) than SA-DQN. Compared to the more advanced baseline RADIAL-DQN, although WocaR-DQN is 5% slower, it achieves better robustness (539% higher reward than RADIAL-DQN in RoadRunner).

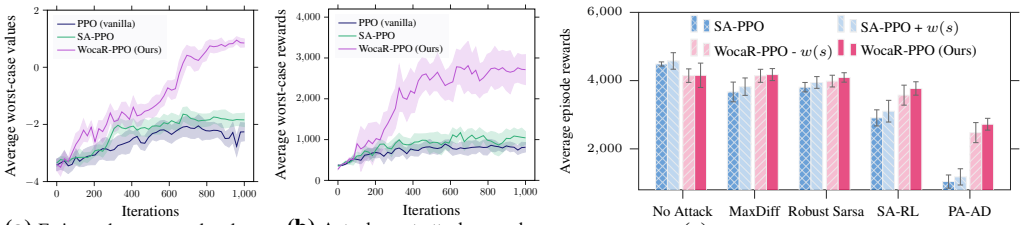
Model	Pong				BankHeist			
	Natural Reward	PGD	MinBest	PA-AD	Natural Reward	PGD	MinBest	PA-AD
		ϵ= 3/255				ϵ= 3/255		
DQN	21.0 ± 0.0	-21.0 ± 0.0	-9.7 ± 4.0	-19.0 ± 2.2	**1308 ± 24**	0 ± 0	119 ± 65	102 ± 92
SA-DQN	21.0 ± 0.0	21.0 ± 0.0	20.6 ± 3.5	18.7 ± 2.6	1245 ± 14	1176 ± 63	1024 ± 31	489 ± 106
RADIAL-DQN	21.0 ± 0.0	21.0 ± 0.0	19.5 ± 2.1	13.2 ± 1.8	1178 ± 4	1176 ± 63	928 ± 113	508 ± 85
WocaR-DQN (Ours)	**21.0 ± 0.0**	**21.0 ± 0.0**	**20.8 ± 3.3**	**19.7 ± 2.4**	1220 ± 12	**1214 ± 7**	**1045 ± 20**	**754 ± 102**

Model	Freeway				RoadRunner			
	Natural Reward	PGD	MinBest	PA-AD	Natural Reward	PGD	MinBest	PA-AD
		ϵ= 3/255				ϵ= 3/255		
DQN	**34.0 ± 0.1**	0.0 ± 0.0	5.5 ± 1.8	4.7 ± 2.9	**45527 ± 4894**	0 ± 0	2985 ± 1440	203 ± 65
SA-DQN	30.0 ± 0.0	30.0 ± 0.0	18.3 ± 3.0	9.5 ± 3.8	44638 ± 2367	20678 ± 1563	4214 ± 2587	5516 ± 4684
RADIAL-DQN	33.1 ± 0.2	**33.2 ± 0.2**	16.4 ± 2.3	10.8 ± 3.6	44675 ± 5854	38576 ± 1960	8476 ± 3964	1290 ± 4015
WocaR-DQN (Ours)	31.2 ± 0.4	31.4 ± 0.3	**19.8 ± 3.8**	**12.3 ± 3.2**	44156 ± 2279	**38720 ± 1765**	**10545 ± 2984**	**8239 ± 2766**

Table 1: Robustness and High Natural Performance of WocaR-DQN. Average episode rewards ± standard deviation over 50 episodes on three baselines and WocaR-DQN on four Atari environments. Best results (natural reward of under attacks for each column) on each environment boldfaced. WocaR-DQN outperforms all the baselines in most cases or gains similar performance in the other metrics. We highlight the most robust agent as gray. Each result is obtained with 10 random seeds.

5.2 Verifying Effectiveness of WocaR-RL

Now we dive deeper into the algorithmic design and verify the effectiveness of WocaR-RL by ablation studies on WocaR-PPO.



(a) Estimated worst-attack values **(b)** Actual worst-attack rewards **(c)** Ablation study for $w(s)$

Figure 6: (a)&(b) Comparison of the estimated worst-attack action value $\underline{Q}^{\pi}_{\phi}$ and the worst rewards under strongest attack during training on Walker2d; **(c)** Ablation evaluations for state importance weight $w(s)$ under no attack and four different attacks on Walker2d.

(1) Worst-attack value estimation. We show the learned worst-attack value estimation, $\underline{Q}^{\pi}_{\phi}$, during the training process in Figure 9c, in comparison with the actual reward under the strongest attack (PA-AD [41]) in Figure 9b. The pink curves in both plots suggest that *our worst-attack value estimation matches the trend of actual worst-case reward under attacks,* although the network estimated value and the real reward have different scales due to the commonly-used reward normalization for learning stability. Therefore, the proposed worst-attack value estimation ($\mathcal{L}_{\text{est}}$) is effective in the experiments.

(2) Worst-case-aware policy optimization. Compared to vanilla PPO and SA-PPO, we can see that *WocaR-PPO improves the worst-attack value and the worst-case reward during training*, suggesting the effectiveness of our worst-attack value improvement ($\mathcal{L}_{\text{wst}}$). The comparison of natural rewards, as well as curves in other environments, are provided in Appendix D.3.4. Moreover, the adjustable weight κ_{wst} in Equation (8) controls the trade-off between natural value and worst-attack value in policy optimization. When κ_{wst} is high, the policy pays more attention to its worst-attack value. Appendix D.3.5 verifies that *WocaR-RL, with different values of weight κ_{wst}, produces different robustness and natural performance while consistently dominating other robust agents.*

(3) Value-enhanced state regularization. We compare the performance of the original WocaR-PPO to variants of WocaR-PPO without the state importance weight $w(s)$ in Figure 6c. The results of SA-PPO with $w(s)$ also indicate the effectiveness of the state importance. Detailed ablation studies for more environments are shown in Appendix D.3.6, which verify that the weighted state regularization is beneficial to enhancing the robustness of RL agents. Without $w(s)$, our algorithm also achieves similar or better performance than baselines, but including this inexpensive technique $w(s)$ gives WocaR-RL a greater advantage, especially under learned strong attacks SA-RL and PA-AD.

6 Conclusion

This paper proposes a robust RL training framework, WocaR-RL, that evaluates and improves the long-term robustness of a policy via worst-attack value estimation, worst-case-aware policy optimization, value-enhanced state regularization. Compared with state-of-the-art robust RL approaches, WocaR-RL achieves better robustness, efficiency and interpretability in experiments.

References

[1] Naveed Akhtar and Ajmal Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access*, 6:14410–14430, 2018.

[2] Sarah Bechtle, Yixin Lin, Akshara Rai, Ludovic Righetti, and Franziska Meier. Curious ilqr: Resolving uncertainty in model-based rl. In Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura, editors, *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 162–171. PMLR, 30 Oct–01 Nov 2020.

[3] Vahid Behzadan and Arslan Munir. Vulnerability of deep reinforcement learning to policy induction attacks. In *International Conference on Machine Learning and Data Mining in Pattern Recognition*, pages 262–275. Springer, 2017.

[4] Vahid Behzadan and Arslan Munir. Whatever does not kill deep reinforcement learning, makes it stronger. *arXiv preprint arXiv:1712.09344*, 2017.

[5] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. Adversarial attacks and defences: A survey. *arXiv preprint arXiv:1810.00069*, 2018.

[6] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning*, pages 1310–1320. PMLR, 2019.

[7] Robert Dadashi, Adrien Ali Taiga, Nicolas Le Roux, Dale Schuurmans, and Marc G Bellemare. The value function polytope in reinforcement learning. In *International Conference on Machine Learning*, pages 1486–1495. PMLR, 2019.

[8] Marc Fischer, Matthew Mirman, Steven Stalder, and Martin Vechev. Online robustness training for deep reinforcement learning. *arXiv preprint arXiv:1911.00887*, 2019.

[9] Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[10] Chris Gaskett. Reinforcement learning under circumstances beyond its control. In *International Conference on Computational Intelligence for Modelling Control and Automation*, 2003.

[11] Saul B Gelfand and Sanjoy K Mitter. Recursive stochastic algorithms for global optimization in rˆd. *SIAM Journal on Control and Optimization*, 29(5):999–1018, 1991.

[12] Adam Gleave, Michael Dennis, Cody Wild, Neel Kant, Sergey Levine, and Stuart Russell. Adversarial policies: Attacking deep reinforcement learning. In *International Conference on Learning Representations*, 2020.

[13] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

[14] Sven Gowal, Krishnamurthy Dj Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. Scalable verified training for provably robust image classification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4842–4851, 2019.

[15] Arthur Guez and H Van Hasselt. Deep reinforcement learning with double q-learning. *Association for the Advancement of Artificial Intelligence*, 2015.

[16] David Silver Hado Van Hasselt, Arthur Guez. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[17] Matthias Heger. Consideration of risk in reinforcement learning. In *International Conference on Machine Learning*, 1994.

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

[19] Yunhan Huang and Quanyan Zhu. Deceptive reinforcement learning under adversarial manipulations on cost signals. In *International Conference on Decision and Game Theory for Security*, pages 217–237. Springer, 2019.

[20] Ezgi Korkmaz. Investigating vulnerabilities of deep neural policies. In *Uncertainty in Artificial Intelligence*, pages 1661–1670. PMLR, 2021.

[21] Jernej Kos and Dawn Song. Delving into adversarial attacks on deep policies. *arXiv preprint arXiv:1705.06452*, 2017.

[22] Aounon Kumar, Alexander Levine, and Soheil Feizi. Policy smoothing for provably robust reinforcement learning. *arXiv preprint arXiv:2106.11420*, 2021.

[23] Xian Yeow Lee, Yasaman Esfandiari, Kai Liang Tan, and Soumik Sarkar. Query-based targeted action-space adversarial policies on deep reinforcement learning agents. In *Proceedings of the ACM/IEEE 12th International Conference on Cyber-Physical Systems*, ICCPS '21, page 87–97, New York, NY, USA, 2021. Association for Computing Machinery.

[24] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[25] Shiau Hong Lim, Huan Xu, and Shie Mannor. Reinforcement learning in robust markov decision processes. *Advances in Neural Information Processing Systems*, 26:701–709, 2013.

[26] Björn Lütjens, Michael Everett, and Jonathan P How. Certified adversarial robustness for deep reinforcement learning. In *Conference on Robot Learning*, pages 1328–1337. PMLR, 2020.

[27] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

[28] 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.

[29] Daniel J. Mankowitz, Nir Levine, Rae Jeong, Abbas Abdolmaleki, Jost Tobias Springenberg, Yuanyuan Shi, Jackie Kay, Todd Hester, Timothy Mann, and Martin Riedmiller. Robust reinforcement learning for continuous control with model misspecification. In *International Conference on Learning Representations*, 2020.

[30] Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3578–3586. PMLR, 10–15 Jul 2018.

[31] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[32] Tuomas Oikarinen, Wang Zhang, Alexandre Megretski, Luca Daniel, and Tsui-Wei Weng. Robust deep reinforcement learning through adversarial loss. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

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

[34] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *International Conference on Machine Learning*, pages 2817–2826. PMLR, 2017.

[35] Amin Rakhsha, Goran Radanovic, Rati Devidze, Xiaojin Zhu, and Adish Singla. Policy teaching via environment poisoning: Training-time adversarial attacks against reinforcement learning. In *International Conference on Machine Learning*, pages 7974–7984, 2020.

[36] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[37] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[38] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[39] Qianli Shen, Yan Li, Haoming Jiang, Zhaoran Wang, and Tuo Zhao. Deep reinforcement learning with robust and smooth policy. In *International Conference on Machine Learning*, pages 8707–8718. PMLR, 2020.

[40] Yanchao Sun, Da Huo, and Furong Huang. Vulnerability-aware poisoning mechanism for online rl with unknown dynamics. In *International Conference on Learning Representations*, 2021.

[41] 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.

[42] Aviv Tamar, Huan Xu, and Shie Mannor. Scaling up robust mdps by reinforcement learning. *arXiv preprint arXiv:1306.6189*, 2013.

[43] Kai Liang Tan, Yasaman Esfandiari, Xian Yeow Lee, Soumik Sarkar, et al. Robustifying reinforcement learning agents via action space adversarial training. In *2020 American control conference (ACC)*, pages 3959–3964. IEEE, 2020.

[44] 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.

[45] Garrett Thomas, Yuping Luo, and Tengyu Ma. Safe reinforcement learning by imagining the near future. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[46] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[47] Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pages 5286–5295. PMLR, 2018.

[48] Fan Wu, Linyi Li, Zijian Huang, Yevgeniy Vorobeychik, Ding Zhao, and Bo Li. Crop: Certifying robust policies for reinforcement learning through functional smoothing. *arXiv preprint arXiv:2106.09292*, 2021.

[49] Chaowei Xiao, Xinlei Pan, Warren He, Jian Peng, Mingjie Sun, Jinfeng Yi, Mingyan Liu, Bo Li, and Dawn Song. Characterizing attacks on deep reinforcement learning. *arXiv preprint arXiv:1907.09470*, 2019.

[50] 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.

[51] Huan Zhang, Hongge Chen, Chaowei Xiao, Sven Gowal, Robert Stanforth, Bo Li, Duane Boning, and Cho-Jui Hsieh. Towards stable and efficient training of verifiably robust neural networks. In *International Conference on Learning Representations*, 2020.

[52] 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.

[53] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 4944–4953, Red Hook, NY, USA, 2018. Curran Associates Inc.

[54] Xuezhou Zhang, Yuzhe Ma, Adish Singla, and Xiaojin Zhu. Adaptive reward-poisoning attacks against reinforcement learning. In *International Conference on Machine Learning*, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See our abstract and Section 1.
   (b) Did you describe the limitations of your work? [Yes] We discuss the limitations in Appendix E.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] We discuss the potential negative societal impacts in Appendix E.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] Our paper conforms to the guidelines.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Appendix A.
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix A.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix D.1
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] For each empirical result, we report error bars.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See efficiency analysis in Appendix D.3.3.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Supplementary Material:

Efficient Adversarial Training without Attacking: Worst-Case-Aware Robust Reinforcement Learning

A Theoretical Analysis

Similar to the worst-attack action value, we can define the worst-attack value as below:

Definition A.1 (Worst-attack Value)**.** For a given policy π, define the worst-attack value of π as

$$\underline{V}^{\pi}(s) := \mathbb{E}_P[\sum_{t=0}^{\infty} \gamma^t R\left(s_t, \pi(h^*(s_t))\right) \mid s_0 = s], \tag{9}$$

where h^* is the optimal attacker which minimizes the victim's cumulative reward under the ϵ constraint.

Proof of Theorem 4.2. First, we show that $\underline{\mathcal{T}}^{\pi}$ is a contraction.

For any two Q functions $Q_1 : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and $Q_2 : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, we have

$$\begin{aligned}
&\|\underline{\mathcal{T}}^{\pi} Q_1 - \underline{\mathcal{T}}^{\pi} Q_2\|_{\infty} \\
&= \max_{s,a} \left| \sum_{s' \in \mathcal{S}} P\left(s' \mid s, a\right) \left[R(s,a) + \gamma \min_{a' \in \mathcal{A}_{\text{adv}}(s',\pi)} Q_1\left(s', a'\right) - R(s,a) + \gamma \min_{a' \in \mathcal{A}_{\text{adv}}(s',\pi)} Q_2\left(s', a'\right) \right] \right| \\
&= \gamma \max_{s,a} \left| \sum_{s' \in \mathcal{S}} P\left(s' \mid s, a\right) \left[\min_{a' \in \mathcal{A}_{\text{adv}}(s',\pi)} Q_1\left(s', a'\right) - \min_{a' \in \mathcal{A}_{\text{adv}}(s',\pi)} Q_2\left(s', a'\right) \right] \right| \\
&\leq \gamma \max_{s,a} \sum_{s' \in \mathcal{S}} P\left(s' \mid s, a\right) \left| \min_{a' \in \mathcal{A}_{\text{adv}}(s',\pi)} Q_1\left(s', a'\right) - \min_{a' \in \mathcal{A}_{\text{adv}}(s',\pi)} Q_2\left(s', a'\right) \right| \\
&\leq \gamma \max_{s,a} \sum_{s' \in \mathcal{S}} P\left(s' \mid s, a\right) \max_{a' \in \mathcal{A}_{\text{adv}}(s',\pi)} \left| Q_1\left(s', a'\right) - Q_2\left(s', a'\right) \right| \\
&= \gamma \max_{s,a} \sum_{s' \in \mathcal{S}} P\left(s' \mid s, a\right) \|Q_1 - Q_2\|_{\infty} \\
&= \gamma \|Q_1 - Q_2\|_{\infty}
\end{aligned}$$

The second inequality comes from the fact that,

$$\left| \min_{x_1} f(x_1) - \min_{x_2} g(x_2) \right| \leq \max_{x} |f(x) - g(x)|$$

The operator $\underline{\mathcal{T}}^{\pi}$ satisfies,

$$\|\underline{\mathcal{T}}^{\pi} Q_1 - \underline{\mathcal{T}}^{\pi} Q_2\|_{\infty} \leq \gamma \|Q_1 - Q_2\|_{\infty}$$

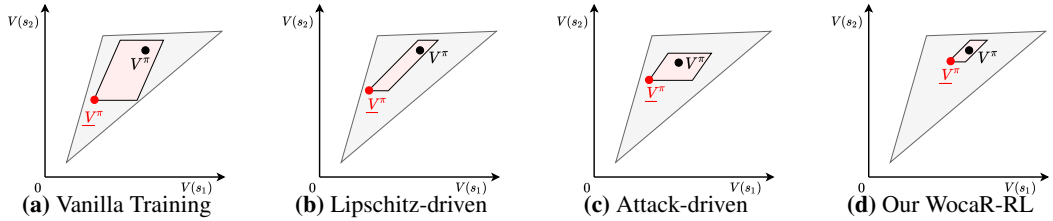
so it is a contraction in the sup-norm.

Recall the definition of worst-attack action value:

$$\underline{Q}^{\pi}(s,a) := \mathbb{E}_P[\sum_{t=0}^{\infty} \gamma^t R\left(s_t, \pi(h^*(s_t))\right) \mid s_0 = s, a_0 = a], \tag{10}$$

where h^* is the optimal attacker which minimizes the victim's cumulative reward under the ϵ constraint. That is, the optimal attacker h^* lets the agent select the worst possible action among all achievable actions in $\mathcal{A}_{\text{adv}}$. Hence, we have $\underline{Q}^{\pi}(s,a) = \underline{\mathcal{T}}^{\pi} \underline{Q}^{\pi}(s,a)$. Therefore, $\underline{Q}^{\pi}(s,a)$ is the fixed point of the Bellman operator $\underline{\mathcal{T}}^{\pi}$.

□



(a) Vanilla Training **(b)** Lipschitz-driven **(c)** Attack-driven **(d)** Our WocaR-RL

Figure 7: Geometric understanding of different training methods following the polytope theory by [7] and [41]. x, y axes represent the policy value for $s_1 \in \mathcal{S}$ and $s_2 \in \mathcal{S}$. The grey polytope depicts the value space of all policies, while the pink polytope (referred to as value perturbation polytope) contains the values of policy π under all attacks with given constraint (ϵ-radius ℓ_p perturbations on the state input to the policy). V^π denotes the value of a learned policy, and $\underline{V}^\pi$ stands for the worst-attack value of this policy π (located at the bottom leftmost vertex of the value perturbation polytope).
Two relations between the value perturbation polytope and policy robustness: The more distant the pink value perturbation polytope's bottom leftmost vertex is from the origin, the higher worst-attack value π has. The smaller the pink value perturbation polytope is, the less vulnerable the policy is (i.e., an ϵ-bounded state perturbation can not lead to a drastic change of the policy value).
Our method: WocaR-RL makes a policy more robust via worst-attack value estimation, worst-case-aware policy optimization and value-enhanced state regularization, which shrink the value perturbation polytope and move the value perturbation polytope's bottom leftmost vertex away from the origin.

B Geometric Understanding of WocaR-RL

B.1 A Closer Look at Robust RL

In real-world applications where observations may be noisy or perturbed, it is important to ensure that the agent not only makes good decisions, but also makes safe decisions.

Existing Robust RL Approaches. There are many existing robust training methods for RL, and we summarize the common ideas as the following two categories.
(1) *Lipschitz-driven methods:* encourage the policy to output similar actions for any pair of clean state and perturbed state, i.e., $\min_\theta \max_{s\in\mathcal{S},\tilde{s}\in\mathcal{B}_\epsilon(s)} \mathsf{Dist}(\pi_\theta(s), \pi_\theta(\tilde{s}))$, where Dist can be any distance metric. Therefore, the policy function (network) has small local Lipschitz constant at each clean state. Note that this idea is similar to many certifiable robust training methods [13] in supervised learning. For example, Fischer et al. [8] achieve provable robustness for DQN by applying the DiffAI [30] approach, so that the DQN agent selects the same action for any element inside $\mathcal{B}_\epsilon(s)$. Zhang et al. [52] propose to minimize the total variance between $\pi(s)$ and $\pi(\tilde{s})$ using convex relaxations of NNs. Although Lipschitz-driven methods are relatively efficient in training, they usually treat all states equally, and do not explicitly consider long-term rewards. Therefore, it is hard to obtain a non-vacuous reward certification, especially in continuous-action environments.
(2) *Attack-driven methods:* train the agent under adversarial attacks, which is analogous to Adversarial Training (AT) [27]. However, different from AT, a PGD attacker may not induce a robust policy in an RL problem due to the uncertainty and complexity of the environment. Zhang et al. [50] propose to alternately train an agent and an RL-based "optimal" adversary, so that the agent can adapt to the worst-case input perturbation. Therefore, attack-driven method can be formulated as $\max_\theta \underline{V}^{\pi_\theta}$. Zhang et al. [50] and a follow-up work by Sun et al. [41] apply the alternate training approach and obtain state-of-the-art robust performance. However, learning the optimal attacker using RL algorithms doubles the learning complexity and the required samples, making it hard to apply these methods to large-scale problems. Moreover, although these attack-driven methods improve the worst-case performance of an agent, the natural reward can be sacrificed.
Note that we discuss methods that improve the robustness of deep policies during training. Therefore, the focus is different from some important works [26, 48, 22] that directly use non-robust policies and execute them in a robust way.

Our Motivation: Geometric Understanding of Robust RL. The robustness of a learned RL policy can be understood from a geometric perspective. Dadashi et al. [7] point out that the value functions of all policies in a finite MDP form a polytope, as shown by the grey area in Figure 7. Sun et al. [41] further find that $V^{\tilde{\pi}}$, possible values of a policy π under all ϵ-constrained ℓ_p perturbations, also form a polytope (pink area in Figure 7), which we refer to as the *value perturbation polytope*.

Recall that in robust RL, we pursue a high natural value V^π, and a high worst-case value $\underline{V}^\pi$ which is the lower leftmost vertex of the value perturbation polytope. A vulnerable policy that outputs a different action for a perturbed state as a larger value perturbation polytope. Lipschitz-driven methods, as Figure 7(a) shows, attempts to shrink the size of the value perturbation polytope, but does not necessarily result in a high $\underline{V}^\pi$. Attack-driven methods, as Figure 7 shows, improves $\underline{V}^\pi$, but have no control over the size of the value perturbation polytope, and may not obtain a high natural value V^π.

Our Proposed Robust RL Principle. In contrast to prior Lipschitz-driven methods and Attack-driven methods, we propose to both "lift the position" and "shrink the size" of the value perturbation polytope. To achieve the above principle in an efficient way, we propose to (1) directly estimate and optimize the worst-case value of a policy without training the optimal attacker (worst-attack value estimation and worst-case-aware policy optimization mechanisms of WocaR-RL), and (2) regularize the local Lipschitz constants of the policy with value-enhanced weights (value-enhanced state regularization mechanism of WocaR-RL). See Section 4 for more details of the proposed algorithm.

C Algorithm Details

C.1 Computing $\mathcal{A}_{\text{adv}}$ by Network Bounding Techniques

Recall that $\mathcal{A}_{\text{adv}}(s,\pi) = \{a \in \mathcal{A} : \exists \tilde{s} \in \mathcal{B}_\epsilon(s) \text{ s.t. } \pi(\tilde{s}) = a\}$ is the set of actions that π may be misled to select in state s. Computing the exact $\mathcal{A}_{\text{adv}}$ is difficult due to the complexity of neural networks, so we use relaxations of network such as Interval Bound Propagation (IBP) [47, 14] to approximately calculate $\mathcal{A}_{\text{adv}}$.

A Brief Introduction to Convex Relaxation Methods. Convex relaxation methods are techniques to bound a neural network that provide the upper and lower bound of the neural network output given a bounded l_p perturbation to the input. In particular, we take l_∞ as an example, which has been studied extensively in prior works. Formally, let f_θ be a real-valued function parameterized by a neural network θ, and let $f_\theta(s)$ denote the output of the neural network with the input s. Given an l_∞ perturbation budget ϵ, convex relaxation method outputs $(\underline{f_\theta(s)}, \overline{f_\theta(s)})$ such that

$$\underline{f_\theta(s)} \leq \min_{\|s'-s\|_\infty \leq \epsilon} f_\theta(s') \leq \max_{\|s'-s\|_\infty \leq \epsilon} f_\theta(s') \leq \overline{f_\theta(s)}$$

Recall that we use π_θ to denote the parameterized policy being trained that maps a state observation to a distribution over the action space, and π denotes the deterministic policy refined from π_θ with $\pi(s) = \text{argmax}_{a\in\mathcal{A}} \pi_\theta(a|s)$. $\mathcal{A}_{\text{adv}}(s,\pi)$ contains actions that could be selected by π (with the highest probability in π_θ's output) when s is perturbed within a ϵ-radius ball. Our goal is to approximately identify a superset of $\mathcal{A}_{\text{adv}}(s,\pi)$, i.e., $\hat{\mathcal{A}}_{\text{adv}}(s,\pi)$, via the convex relaxation of networks introduced above.

Computing $\mathcal{A}_{\text{adv}}$ in Continuous Action Space. The most common policy parameterization in a continuous action space is through a Gaussian distribution. Let $\mu_\theta(s)$ be the mean of Gaussian computed by $\pi_\theta(s)$, then $\pi = \mu(s)$. Therefore, we can use network relaxation to compute an upper bound and a lower bound of μ_θ with input $\mathcal{B}_\epsilon(s)$. Then, $\hat{\mathcal{A}}_{\text{adv}}(s,\pi) = [\underline{\mu_\theta(s)}, \overline{\mu_\theta(s)}]$, i.e., a set of actions that are coordinate-wise bounded by $\underline{\mu_\theta(s)}$ and $\overline{\mu_\theta(s)}$. For other continuous distributions, e.g., Beta distribution, the computation is similar, as we only need to find the largest and smallest actions. In summary, we can compute $\hat{\mathcal{A}}_{\text{adv}}(s,\pi) = [\underline{\pi_\theta(s)}, \overline{\pi_\theta(s)}]$.

Computing $\mathcal{A}_{\text{adv}}$ in Discrete Action Space. For a discrete action space, the output of π_θ is a categorical distribution, and π selects the action with the highest probability. Or equivalently, in value-based algorithms like DQN, the Q network (can be regarded as π_θ) outputs the Q estimates for each action, and π selects the action with the highest Q value. In this case, we can compute the upper and lower bound of π_θ in every dimension (corresponding to an action), denoted as $\overline{a}_i, \underline{a}_i$, $\forall 1 \leq i \leq |\mathcal{A}|$. Then, an action $a_i \in \mathcal{A}$ is in $\hat{\mathcal{A}}_{\text{adv}}$ if for all $1 \leq j \leq |\mathcal{A}|, j \neq i$, we have $\overline{a}_i > \underline{a}_j$.

Implementation details of $\mathcal{A}_{\text{adv}}$ For a continuous action space, interval bound propagation (IBP) is the cheapest method to implement convex relaxation. We use IBP+Backward relaxation provided by *auto_LiRPA* library, following [52] to efficiently produce tighter bounds $\mathcal{A}_{\text{adv}}$ for the policy networks

π_{theta}. For a discrete action space, we compute the layer-wise output bounds for the Q-network by applying robustness verification algorithms from [32].

C.2 Worst-case-aware Robust PPO (WocaR-PPO)

In policy-based DRL methods [37, 24, 38] such as PPO, the actor policy π_θ is optimized so that it increases the probability of selecting actions with higher critic values. Therefore, we combine our worst-attack critic and the original critic function, and optimize π_θ such that both the natural value ($\mathcal{L}_{\text{RL}}$) and the worst-attack action value $\underline{Q}^{\pi}_{\phi}$ ($\mathcal{L}_{\text{wst}}$) can be increased ($\mathcal{L}_{\text{RL}}$ and $\mathcal{L}_{\text{wst}}$). At the same time, π_θ is also regularized by $\mathcal{L}_{\text{reg}}$.

We provide the full algorithm of WocaR-PPO in Algorithm 1 and highlight the differences with the prior method SA-PPO. WocaR-PPO needs to train an additional worst-attack critic $\underline{Q}^{\pi}_{\phi}$ to provide the robust-PPO-clip objective. The perturbation budget ϵ_t increases slowly during training. The implementation of $\mathcal{L}_{\text{reg}}$ is the same as the SA-regularizer [52]. For computing the state importance weight w_{s_t}, because there is no Q-value network in PPO, we provide a different formula to measure the state importance without extra calculation (Line 11 in Algorithm 1).

C.3 Worst-case-aware Robust DQN (WocaR-DQN)

For value-based DRL methods [31, 15, 46] such as DQN, a Q network is learned to evaluate the natural action value. Although the policy is not directly modeled by a network, the Q network induces a greedy policy by $\pi(s) = \text{argmax}_a Q(s, a)$. To distinguish the acting policy and the natural action value, we keep the original Q network, and learn a new Q network that serves as a robust policy. This new Q network is called a *robust Q network*, denoted by Q_r, which is used to take greedy actions $a = \pi(s) := \text{argmax}_a Q_r(s, a)$ In addition to the original vanilla Q network Q_v and the robust Q network Q_r, we learn the worst-attack critic network $\underline{Q}^{\pi}_{\phi}$, which evaluates the worst-attack action value of the greedy policy induced by Q_r. Then, we update Q_r by assigning higher values for actions with both high natural Q value and high worst-attack action value ($\mathcal{L}_{\text{RL}}$ and $\mathcal{L}_{\text{wst}}$), while enforcing the network to output the same action under bounded state perturbations ($\mathcal{L}_{\text{reg}}$).

WocaR-DQN is presented in Algorithm 2. WocaR-DQN trains three Q-value functions including a vanilla Q network, a worst-case Q network, and a robust Q network. The worst-case Q $\underline{Q}^{\pi}_{\phi}$ is learned to estimate the worst-case performance and the robust Q is updated using the vanilla value and worst-case value together. Moreover, a target Q network is used as the original DQN implementation, to compute the target value when updating the vanilla Q network (Line 8 to 10 in Algorithm 2). To learn the worst-case critic $\underline{Q}^{\pi}_{\phi}$, we select the worst-attack action from the estimated possible perturbed action set $\hat{\mathcal{A}}_{\text{adv}}$ to compute the worst-case TD loss $\mathcal{L}_{\text{est}}$ (Line 11 to 15). The implementation of $\mathcal{L}_{\text{reg}}$ is the same as the SA-regularizer [52], where the robust Q network is regularized. To update the robust Q, we use a special y_i^r which combines the target Q $Q_{v'}$ and Q_r for the next state to compute the TD loss, and minimize the $\mathcal{L}_{\text{reg}}$ weighted by the state importance $w(s_i)$ (Line 16 to 17). In WocaR-DQN, we use an increasing ϵ_t schedule and a more slowly increasing worst-case schedule $\kappa_{wst}(t)$ for robust Q training.

C.4 Worst-case-aware Robust A2C (WocaR-A2C)

We also provide WocaR-A2C based on A2C implementation in Algorithm 3. Differ from the original A2C, WocaR-A2C needs to learn an additional $\underline{Q}^{\pi}_{\phi}$ similar to WocaR-PPO. To learn $\underline{Q}^{\pi}_{\phi}$, we compute the output bounds for the policy network π_{θ_π} under ϵ-bounded perturbations and then select the worst action $\hat{a}_{t+1}$ to calculate the TD-loss $\mathcal{L}_{\text{est}}$ (Line 6 to 9). The solutions for state importance weight $w(s_t)$ and regularization $\mathcal{L}_{\text{reg}}$ are same as WocaR-PPO (Line 10-11). To learn the policy network π_{θ_π}, we minimize the $\underline{Q}^{\pi}_{\phi}$ value together with the original actor loss (Line 12).

C.5 Extension to Action Attacks

Although our paper mainly focuses on state attack, our proposed techniques and algorithms based on the worst-attack Bellman operator can be easily extended to action attack, which is another threat model studied in previous works [34, 43, 44]. In fact, for action attack, we even do not need to apply

Algorithm 1 Worst-case-aware Robust PPO (WocaR-PPO). We highlight the difference compares with SA-PPO [52] in blue.

Input: Number of iterations T, a schedule ϵ_t for the perturbation radius ϵ, weights $\kappa_{\text{wst}}, \kappa_{\text{reg}}$

1: Initialize policy network $\pi_{\theta_\pi}(a \mid s)$, value network $V_{\theta_V}(s)$ and worst-attack critic network $\underline{Q}^{\pi}_{\phi}(s,a)$ with parameters θ_π, θ_V and ϕ

2: **for** $k = 0, 1, ..., T$ **do**

3: Collect a set of trajectories $\mathcal{D} = \{\tau_k\}$ by running π_{θ_π} in the environment, each trajectory τ_k contains $\tau_k := \{(s_t, a_t, r_t, s_{t+1})\}, t \in [|\tau_k|]$

4: Compute rewards-to-go $\hat{R}_t$ for each step t in every trajectory k with discount factor γ

5: Compute advantage estimation $\hat{A}_t$ based on the current value function $V_{\theta_V}(s_t)$ and cumulative reward $\hat{R}_t$ for each step t

6: Update parameters of value function θ_V by regression on mean-squared error:

$$\theta_V \leftarrow \arg\min_{\theta_V} \frac{1}{|\mathcal{D}|\,|\tau_k|} \sum_{\tau_k \in \mathcal{D}} \sum_{t=0}^{|\tau_k|} \left(V_{\theta_V}(s_t) - \hat{R}_t \right)^2$$

7: Use IBP to compute bounds of current policy network π:
Find the upper bound $\overline{\pi}(s_{t+1}, \epsilon; \theta)$ and lower bound $\underline{\pi}(s_{t+1}, \epsilon; \theta)$ of the policy network π_{θ_π}

8: Select the worst action for next states:
Calculate the action satisfied $\hat{a}_{t+1} = \arg\min_{a \in [\underline{\pi}, \overline{\pi}]} \underline{Q}^{\pi}_{\phi}(s_{t+1}, a)$ with the worst-attack critic network $\underline{Q}^{\pi}_{\phi}$ using gradient descent.

9: Compute next worst-case value:
Set $\underline{y}_t = \begin{cases} r_t & \text{for terminal } s_{t+1} \\ r_t + \gamma \underline{Q}^{\pi}_{\phi}(s_{t+1}, \hat{a}_{t+1}) & \text{for non-terminal } s_{t+1} \end{cases}$

10: Update parameters of worst-attack critic network ϕ by minimizing the TD-error ($\mathcal{L}_{\text{est}}$):

$$\phi \leftarrow \arg\min_{\phi} \frac{1}{|\mathcal{D}|\,|\tau_k|} \sum_{\tau_k \in \mathcal{D}} \sum_{t=0}^{|\tau_k|} (\underline{y}_t - \underline{Q}^{\pi}_{\phi}(s_t, a_t))^2$$

11: For each state s_t, calculate a state importance weight w_{s_t} by $V_{\theta_V}(s_t) - \min_a \underline{Q}^{\pi}_{\phi}(s_t, a)$ for s_t

12: Solve the value-enhanced state regularization loss by SGLD (Stochastic gradient Langevin dynamics [11]) (from SA-PPO [52]):

$$\mathcal{L}_{\text{reg}}(\pi_\theta) = \frac{1}{N} \sum_{t=1}^{N} w(s_t) \max_{\tilde{s}_t \in \mathcal{B}_\epsilon(s_t)} \mathsf{Dist}(\pi_\theta(s_t), \pi_\theta(\tilde{s}_t))$$

13: Update the policy network by minimizing the Robust-PPO-Clip objective (via ADAM):

$$\theta_\pi \leftarrow \arg\min_{\theta'_\pi} \frac{1}{|\mathcal{D}||\tau_k|} \left[\sum_{\tau_k \in \mathcal{D}} \sum_{t=0}^{|\tau_k|} \min\left(\rho_{\theta'_\pi}(a_t \mid s_t)(\hat{A}_t + \kappa_{\text{wst}} \underline{Q}^{\pi}_{\phi}(s_t, a_t)), g(\rho_{\theta'_\pi}(a_t \mid s_t)) \right)(\hat{A}_t + \kappa_{\text{wst}} \underline{Q}^{\pi}_{\phi}(s_t, a_t))) + \kappa_{\text{reg}} w(s_i) \mathcal{L}_{\text{reg}}(\pi_\theta) \right]$$

where $\rho_{\theta'_\pi}(a_t \mid s_t) := \frac{\pi_{\theta'_\pi}(a_t|s_t)}{\pi_{\theta_\pi}(a_t|s_t)}, g(\rho) := \text{clip}\left(\rho_{\theta'_\pi}(a_t \mid s_t), 1 - \epsilon_{\text{clip}}, 1 + \epsilon_{\text{clip}}\right)$

14: **end for**

IBP for the worst-attack Bellman backup. We could just simply replace $\mathcal{A}_{\text{adv}}$ with the set of actions that the agent could take under attack, then the rest of the algorithms will follow the exact same as the ones presented here.

Algorithm 2 Worst-case-aware Robust DQN (WocaR-DQN). We highlight the difference compares with SA-DQN [52] in blue.

Input: Number of iterations T, target network update coefficient τ, a schedule ϵ_t for the perturbation radius ϵ, a worst-case schedule $\kappa_{\text{wst}}(t)$ for weight κ_{wst}, regularization weight κ_{reg}

1: Initialize a vanilla Q network $Q_v(s,a)$, target Q network $Q_{v'}(s,a)$, a robust Q network $Q_r(s,a)$, and a worst-attack critic $\underline{Q}^{\pi}_{\phi}(s,a)$ with parameters θ_{Q_v}, $\theta_{Q_{v'}}$, θ_{Q_r}, and ϕ

2: Initialize replay buffer $\mathcal{B}$

3: **for** $k = 0, 1, ..., T$ **do**

4: With probability β select random action a_t, otherwise select $a_t = \arg\max_a Q_r(s_t, a|\theta_{Q_r})$

5: Execute action a_t in environment and observe reward r_t and the next state s_{t+1}.

6: Store transition $\{s_t, a_t, r_t, s_{t+1}\}$ in $\mathcal{B}$

7: Sample random a minibatch of N transitions $\{s_i, a_i, r_i, s_{i+1}\}$ from $\mathcal{B}$

8: Set $y_i = \begin{cases} r_i & \text{for terminal } s_{i+1} \\ r_i + \gamma \max_{a'} Q_{v'}(s_{i+1}, a'; \theta) & \text{for non-terminal } s_{i+1} \end{cases}$

9: Compute TD-loss for the vanilla Q network: $L(s_i, a_i, s_{i+1}; \theta) = (y_i - Q_v(s_i, a_i; \theta))^2$ and optimize θ_{Q_v}

10: Soft update the target action-value network: $\theta_{Q_{v'}} \leftarrow \tau\theta_{Q_v} + (1-\tau)\theta_{Q_{v'}}$

11: Computing bounds of robust action-value function:
For each action a in action space $\mathcal{A}$, calculate the output bounds of robust action-value function Q_r under ϵ_t-bounded perturbations using IBP to input s_{i+1}: $Q_l(s_{t+1}, a, \epsilon_t)$ and $Q_u(s_{t+1}, a, \epsilon_t)$.

12: Find the possible perturbed action set:
For every action $a \in \mathcal{A}$, if $Q_u(s_{t+1}, a, \epsilon_t) > Q_l(s_{t+1}, a', \epsilon_t), \forall a' \in \mathcal{A}$, then add a in the perturbed action set $\hat{\mathcal{A}}_{\text{adv}}$

13: Calculate the worst-attack action: $\hat{a}_{i+1} = \arg\min_{a \in \hat{\mathcal{A}}_{\text{adv}}} \underline{Q}^{\pi}_{\phi}(s_{i+1}, a)$.

14: Set $\underline{y}_i = \begin{cases} r_i & \text{for terminal } s_{i+1} \\ r_i + \gamma \underline{Q}^{\pi}_{\phi}(s_{i+1}, \hat{a}_{i+1}; \theta) & \text{for non-terminal } s_{i+1} \end{cases}$

15: Compute TD-loss for worst-attack critic: $\mathcal{L}_{\text{est}} = (\underline{y}_i - \underline{Q}^{\pi}_{\phi}(s_i, a_i; \phi))^2$ and perform a gradient descent step with respect to the parameters ϕ

16: Calculate the state importance w_{s_i} for each s_i by normalizing $\max_a Q_v(s_t, a) - \min_a Q_v(s_t, a)$

17: Update the robust Q function Q_r based on the modified TD-Loss and value-enhanced state regularization:

$$L(s_i, a_i, s_{i+1}; \theta_{Q_r}) = (y_i^r - Q_r(s_i, a_i; \theta))^2 + \kappa_{\text{reg}} w(s_i) \mathcal{L}_{\text{reg}}(\theta_{Q_r})$$

where $y_i^r = r_i + \gamma \max_{a'} \left[\kappa_{\text{wst}}(t) Q_{v'}(s_{i+1}, a'; \theta) + (1 - \kappa_{\text{wst}}(t)) \underline{Q}^{\pi}_{\phi}(s_{i+1}, a'; \theta) \right]$ if s_{i+1} is a non-terminal state, otherwise $y_i^r = r_i$

18: **end for**

D Experiment Details and Additional Results

D.1 Implementation Details

For reproducibility, the reported results are selected from 30 agents for different training methods with medium performance due to the high variance in RL training.

D.1.1 PPO in MuJoCo

(a) PPO Baselines

Vanilla PPO We use the optimal hyperparameters from [52] with the original fully connected (MLP) structure as the policy network for vanilla PPO training on all environments. On Hopper, Walker2d and Halfcheetah, we train for 2 million steps (976 iterations) , and 10 million steps (4882 iterations) on Ant to ensure convergence, which are consistent with other baselines (except ATLA methods).

Algorithm 3 Worst-case-aware Robust A2C (WocaR-A2C). We highlight the difference compares with SA-A2C [52] in blue.

Input: Number of iterations T, a schedule ϵ_t for the perturbation radius ϵ, weights $\kappa_{\text{wst}}, \kappa_{\text{reg}}$

1: Initialize policy network $\pi_{\theta_\pi}(a \mid s)$, value network $V_{\theta_V}(s)$ and worst-attack critic network $\underline{Q}^{\pi}_{\phi}(s,a)$ with parameters θ_π, θ_V and ϕ

2: **for** $k = 0, 1, ..., T$ **do**

3: Collect a set of trajectories $\mathcal{D} = \{\tau_k\}$ by running π_{θ_π} in the environment, each trajectory τ_k contains $\tau_k := \{(s_t, a_t, r_t, s_{t+1})\}, t \in [|\tau_k|]$

4: Compute advantage function A_t by

$$A_t = r_t + \gamma V_{\theta_V}(s_{t+1}) - V_{\theta_V}(s_t)$$

5: Update parameters of value function θ_V by regression on mean-squared error:

$$\theta_V \leftarrow \arg\min_{\theta_V} \frac{1}{|\mathcal{D}|\,|\tau_k|} \sum_{\tau_k \in \mathcal{D}} \sum_{t=0}^{|\tau_k|} A_t^2$$

6: Use IBP to compute bounds of current policy network π:
Find the upper bound $\overline{\pi}(s_{t+1}, \epsilon; \theta)$ and lower bound $\underline{\pi}(s_{t+1}, \epsilon; \theta)$ of the policy network π_{θ_π}

7: Select the worst action for next states:
Calculate the action satisfied $\hat{a}_{t+1} = \arg\min_{a \in [\underline{\pi}, \overline{\pi}]} \underline{Q}^{\pi}_{\phi}(s_{t+1}, a)$ with the worst-attack critic network $\underline{Q}^{\pi}_{\phi}$ using gradient descent.

8: Compute next worst-case value:
Set $\underline{y}_t = \begin{cases} r_t & \text{for terminal } s_{t+1} \\ r_t + \gamma \underline{Q}^{\pi}_{\phi}(s_{t+1}, \hat{a}_{t+1}) & \text{for non-terminal } s_{t+1} \end{cases}$

9: Update parameters of worst-attack critic network ϕ by minimizing the TD-error ($\mathcal{L}_{\text{est}}$):

$$\phi \leftarrow \arg\min_{\phi} \frac{1}{|\mathcal{D}|\,|\tau_k|} \sum_{\tau_k \in \mathcal{D}} \sum_{t=0}^{|\tau_k|} (\underline{y}_t - \underline{Q}^{\pi}_{\phi}(s_t, a_t))^2$$

10: For each state s_t, calculate a state importance weight $w(s_t)$ by $V_{\theta_V}(s_t) - \min_a \underline{Q}^{\pi}_{\phi}(s_t, a)$ for s_t

11: Solve the value-enhanced state regularization loss [51] by SGLD (Stochastic gradient Langevin dynamics [11]):

$$\mathcal{L}_{\text{reg}}(\pi_{\theta_\pi}) = \frac{1}{N} \sum_{t=1}^{N} w(s_t) \max_{\tilde{s}_t \in \mathcal{B}_\epsilon(s_t)} \mathsf{Dist}(\pi_{\theta_\pi}(s_t), \pi_{\theta_\pi}(\tilde{s}_t))$$

12: Update the policy network by (via ADAM)

$$\theta_\pi \leftarrow \arg\min_{\theta'_\pi} \frac{1}{|\mathcal{D}||\tau_k|} \left[\sum_{\tau_k \in \mathcal{D}} \sum_{t=0}^{|\tau_k|} (A_t log \pi_{\theta_\pi}(s_t) + \kappa_{\text{wst}} \underline{Q}^{\pi}_{\phi}(s_t, a_t)) \right]$$

13: **end for**

SA-PPO We use the hyperparameters using a grid search and solve the regularizer using convex relaxation with the IBP+Backward scheme to solve the regularizer. The regularization parameter $kappa$ is chosen in $\{0.01, 0.03, 0.1, 0.3, 1.0\}$.

ATLA-PPO The hyperparameters for both policy and adversary are tuned for vanilla PPO with LSTM models. A larger entropy bonus coefficient is set to allow sufficient exploration. We set $N_v = N_\pi = 1$ for all experiments. We train 2441 iterations for Hopper, Walker2d, and Halfcheetah as well as 4882 iterations for Ant.

PA-ATLA-PPO We use the hyperparameters similar to ATLA-PPO and conduct a grid search for a part of adversary hyperparameters including the learning rate and the entropy bonus coefficient.

RADIAL-PPO RADIAL-PPO applies the same value of hyperparameters from [32]. We train agents with the same iterations aligning vanilla PPO for fair comparison.

(b) PPO Attackers

For **Random** and **MaxDiff** attack, we directly use the implementation from [50]. The reported rewards under RS attack are from 30 trained robust value function, which is used to attack agents.
For **SA-RL** attack, a grid search of the optimal hyperparameters for each robust agents is conducted to find the strongest attacker. The strength of the regularization κ is set as 1×10^{-6} to 1.
For **PA-AD** attack, the adversaries are trained by PPO with a grid search of hyperparameters to obtain the strongest adversary.
For different types of RL-based attacks, we respectively train 100 adversaries and report the worst rewards among all trained adversaries.

(c) WocaR-PPO We use the same LSTM structure (single layer with 64 hidden neurons as in vanilla PPO agents. With a grid search experiment, we find the optimal hyperparameters for WocaR-PPO. Specially, we use PGD to compute bounds for the policy network and convex relaxation to solve the state regularization. The number of WocaR-PPO training steps in all environments are the same as those in vanilla PPO. We tune the adjustable weight κ_{wst} and increase κ_{wst} from 0 to the target value. For Hopper, Walker2d and Halfcheetah, κ_{wst} is linearly increasing and we set the target value as 0.8. For Ant, we choose the exponential increase and the target value as 0.5.

D.1.2 DQN in Atari

(a) DQN Baselines

Vanilla DQN We follow [52] and [32] in hyperparameters and network structures for vanilla DQN training. The implementation of all our baselines applies Double DQN [16] and Prioritized Experience Replay [36]. For each Atari environment without framestack, we normalize the pixel values to $[0, 1]$ and clip rewards to $[-1, +1]$. For reliably convergence, we run 6×10^6 steps for all baselines on all environments. Additionally, we use a replay buffer with a capacity of 5×10. During testing, we evaluate agents without epsilon greedy exploration for 1000 episodes.
SA-DQN SA-DQN use the same settings of network structures and hyperparameters as in vanilla DQN. The regularization parameter κ is chosen from $0.005, 0.01, 0.02$ and the schedule of ϵ during training also follows [52].
RADIAL-DQN Following the original implementation from [32], we reproduce the results of RADIAL-DQN with our environment settings.

(b) DQN Attackers

For **PGD** attacks, we apply 10-step untargeted PGD attacks. We also try 50-step PGD attacks, but we find that the rewards of robust agents do not further reduce.
For **MinBest** attacks, we use FGSM to compute state perturbations following [18].
For **PA-AD** attacks, the PA-AD attackers are learned with the ACKTR algorithm. We use a learning rate 0.0001 and train the attackers for 5 million frames.

(c) WocaR-DQN For WocaR-DQN, we keep the same network architectures and hyperparameters as in vanilla DQN agents. During training, we set the adjustable weight κ_{wst} as 0 for the first 2×10^6 steps, and then exponentially increase it from 0 to 0.5 for 4×10^6 steps.

D.2 Additional Experiment Results on Robustness Performance

MuJoCo Experiments We reported all results in Table 2 including episode rewards of well-trained robust models under various adversarial attacks. Under this full adversarial evaluation, we provide a robustness comparison between baselines and our algorithm from a comprehensive angle. We report the attack performance under a common chosen perturbation budget ϵ following [52, 50]. Results in all four MuJoCo environments show that our WocaR-PPO is the most robust method. We emphasize that Table 2 reports the final performance of all robust training baselines after convergence, but some baselines takes much more steps than our WocaR-PPO. Table 5 in Appendix D.3.2 compares all methods under the same number of training steps, where WocaR-PPO outperforms baselines more significantly.

Environment	Model	Natural Reward	Random	MAD	RS	SA-RL	PA-AD
Halfcheetah state-dim: 17 ϵ=0.15	PPO (vanilla)	**7117** $\pm$ **98**	5486 $\pm$ 1378	1836 $\pm$ 866	489 $\pm$ 758	-660 $\pm$ 218	-356 $\pm$ 407
	SA-PPO	3632 $\pm$ 20	3619 $\pm$ 18	3624 $\pm$ 23	3283 $\pm$ 20	3028 $\pm$ 23	2512 $\pm$ 16
	ATLA-PPO	6157 $\pm$ 852	6164 $\pm$ 603	5790 $\pm$ 174	4806 $\pm$ 392	5058 $\pm$ 418	2576 $\pm$ 548
	PA-ATLA-PPO	6289 $\pm$ 342	**6215** $\pm$ **346**	**5961** $\pm$ **253**	5226 $\pm$ 114	4872 $\pm$ 379	3840 $\pm$ 273
	RADIAL-PPO	4724 $\pm$ 14	4731 $\pm$ 42	3994 $\pm$ 156	3864 $\pm$ 232	3253 $\pm$ 131	2674 $\pm$ 168
	WocaR-PPO (Ours)	6032 $\pm$ 68	5969 $\pm$ 149	5850 $\pm$ 228	**5319** $\pm$ **220**	**5365** $\pm$ **54**	**4269** $\pm$ **172**
Hopper state-dim: 11 ϵ=0.075	PPO (vanilla)	3167 $\pm$ 542	2101 $\pm$ 793	1410 $\pm$ 655	794 $\pm$ 238	636 $\pm$ 9	160 $\pm$ 136
	SA-PPO	3705 $\pm$ 2	2710 $\pm$ 801	2652 $\pm$ 835	1130 $\pm$ 42	1076 $\pm$ 791	856 $\pm$ 21
	ATLA-PPO	3291 $\pm$ 600	3165 $\pm$ 576	2814 $\pm$ 725	2244 $\pm$ 618	1772 $\pm$ 802	1232 $\pm$ 350
	PA-ATLA-PPO	3449 $\pm$ 237	3325 $\pm$ 239	3145 $\pm$ 546	3002 $\pm$ 329	1529 $\pm$ 284	2521 $\pm$ 325
	RADIAL-PPO	**3740** $\pm$ **44**	**3729** $\pm$ **100**	3214 $\pm$ 142	2141 $\pm$ 232	1722 $\pm$ 186	1439 $\pm$ 204
	WocaR-PPO (Ours)	3616 $\pm$ 99	3633 $\pm$ 30	**3541** $\pm$ **207**	**3277** $\pm$ **159**	**2390** $\pm$ **145**	**2579** $\pm$ **229**
Walker2d state-dim: 17 ϵ=0.05	PPO (vanilla)	4472 $\pm$ 635	3007 $\pm$ 1200	2869 $\pm$ 1271	1336 $\pm$ 654	1086 $\pm$ 516	804 $\pm$ 130
	SA-PPO	4487 $\pm$ 61	4465 $\pm$ 39	3668 $\pm$ 689	3808 $\pm$ 138	2908 $\pm$ 336	1042 $\pm$ 353
	ATLA-PPO	3842 $\pm$ 475	3927 $\pm$ 368	3836 $\pm$ 492	3239 $\pm$ 294	3663 $\pm$ 707	1224 $\pm$ 770
	PA-ATLA-PPO	4178 $\pm$ 529	4129 $\pm$ 78	4024 $\pm$ 272	3966 $\pm$ 307	3450 $\pm$ 178	2248 $\pm$ 131
	RADIAL-PPO	**5251** $\pm$ **12**	**5184** $\pm$ **42**	**4494** $\pm$ **150**	3572 $\pm$ 239	3320 $\pm$ 245	1395 $\pm$ 194
	WocaR-PPO (Ours)	4156 $\pm$ 495	4244 $\pm$ 157	4177 $\pm$ 176	**4093** $\pm$ **138**	**3770** $\pm$ **196**	**2722** $\pm$ **173**
Ant state-dim: 111 ϵ=0.15	PPO (vanilla)	**5687** $\pm$ **758**	5261 $\pm$ 1005	1759 $\pm$ 828	268 $\pm$ 227	-872 $\pm$ 436	-2580 $\pm$ 872
	SA-PPO	4292 $\pm$ 384	4986 $\pm$ 452	4662 $\pm$ 522	3412 $\pm$ 1755	2511 $\pm$ 1117	-1296 $\pm$ 923
	ATLA-PPO	5359 $\pm$ 153	5366 $\pm$ 104	5240 $\pm$ 170	4136 $\pm$ 149	3765 $\pm$ 101	220 $\pm$ 338
	PA-ATLA-PPO	5469 $\pm$ 106	5496 $\pm$ 158	**5328** $\pm$ **196**	4124 $\pm$ 291	3694 $\pm$ 188	2986 $\pm$ 364
	RADIAL-PPO	5076 $\pm$ 254	5031 $\pm$ 142	4777 $\pm$ 156	3731 $\pm$ 177	3188 $\pm$ 115	1544 $\pm$ 194
	WocaR-PPO (Ours)	5596 $\pm$ 225	**5558** $\pm$ **241**	5284 $\pm$ 182	**4339** $\pm$ **160**	**3822** $\pm$ **185**	**3164** $\pm$ **163**

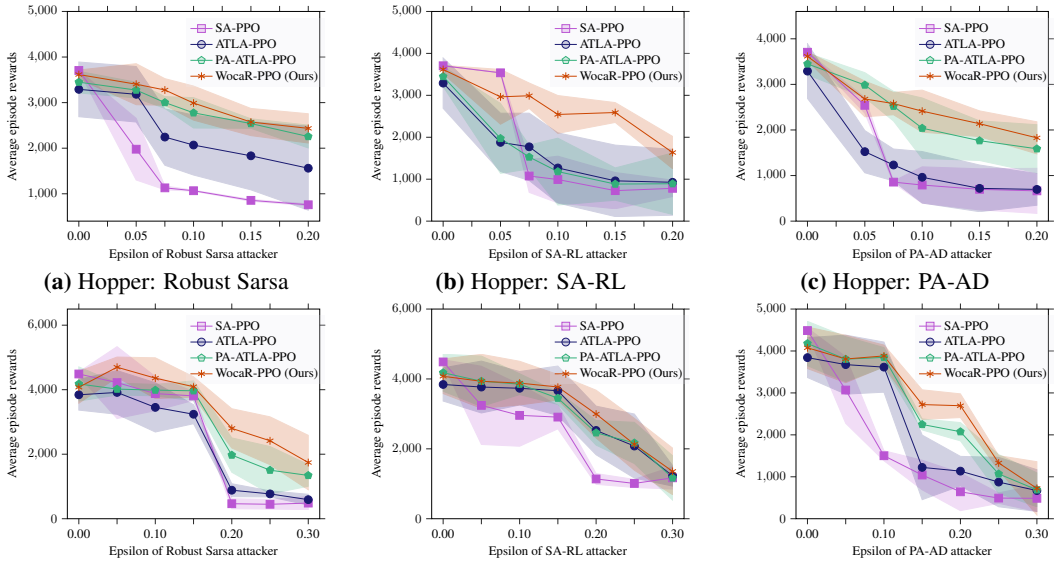
Table 2: Average episode rewards $\pm$ standard deviation over 50 episodes on five baselines and WocaR-PPO on Hopper, Walker2d, Halfcheetah, and Ant. Natural reward and rewards under five types of attacks are reported. Under each column corresponding to an evaluation metric, we bold the best results. And the row for the most robust agent is highlighted as gray. Note that *ATLA-PPO, PA-ATLA-PPO and RADIAL-PPO are trained with more than* $2\times$ *steps than WocaR-PPO*, as reported in Table 6.

Atari Experiments In Table 3, we present performance based on DQN on four Atari environments under 1/255 and 3/255 ϵ attack. Under ϵ of 1/255, our WocaR-DQN achieves competitive performance under PGD attacks and outperforms all baselines under MinBest and PA-AD attacks, which shows better robustness of WocaR-DQN under weaker attacks.

Based on vanilla A2C, we implement SA-A2C[52] and PA-ATLA-A2C[41] as robust baselines. We implement WocaR-A2C to compare with ATLA methods on Atari. In Table 4, under any ϵ value, our WocaR-A2C outperforms other robust baselines across different attacks. We can conclude that our method considerably enhance more robustness than ATLA methods on Atari.

Environment	Model	Natural Reward	PGD (10 steps)		MinBest		PA-AD	
			ϵ=1/255	ϵ=3/255	ϵ=1/255	ϵ=3/255	ϵ=1/255	ϵ=3/255
Pong	DQN	21.0 $\pm$ 0.0	-21.0 $\pm$ 0.0	-21.0 $\pm$ 0.0	-7.4 $\pm$ 2.8	-9.7 $\pm$ 4.0	-18.2 $\pm$ 2.3	-19.0 $\pm$ 2.2
	SA-DQN	21.0 $\pm$ 0.0	21.0 $\pm$ 0.0	21.0 $\pm$ 0.0	21.0 $\pm$ 0.0	20.6 $\pm$ 3.5	20.4 $\pm$ 1.8	18.7 $\pm$ 2.6
	RADIAL-DQN	21.0 $\pm$ 0.0	21.0 $\pm$ 0.0	21.0 $\pm$ 0.0	21.0 $\pm$ 0.0	19.5 $\pm$ 2.1	20.3 $\pm$ 2.5	13.2 $\pm$ 1.8
	WocaR-DQN (Ours)	**21.0** $\pm$ **0.0**	**21.0** $\pm$ **0.0**	**21.0** $\pm$ **0.0**	**21.0** $\pm$ **0.0**	**20.8** $\pm$ **3.3**	**21.0** $\pm$ **0.2**	**19.7** $\pm$ **2.4**
Freeway	DQN	**34.0** $\pm$ **0.1**	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	9.5 $\pm$ 3.0	5.5 $\pm$ 1.8	9.3 $\pm$ 2.7	4.7 $\pm$ 2.9
	SA-DQN	30.0 $\pm$ 0.0	30.0 $\pm$ 0.0	30.0 $\pm$ 0.0	27.2 $\pm$ 3.4	18.3 $\pm$ 3.0	20.1 $\pm$ 4.0	9.5 $\pm$ 3.8
	RADIAL-DQN	33.1 $\pm$ 0.2	**33.1** $\pm$ **0.2**	**33.2** $\pm$ **0.2**	22.6 $\pm$ 3.3	16.4 $\pm$ 2.3	18.5 $\pm$ 4.2	10.8 $\pm$ 3.6
	WocaR-DQN (Ours)	31.2 $\pm$ 0.4	31.2 $\pm$ 0.5	31.4 $\pm$ 0.3	**29.6** $\pm$ **2.5**	**19.8** $\pm$ **3.8**	**24.9** $\pm$ **3.7**	**12.3** $\pm$ **3.2**
BankHeist	DQN	**1308** $\pm$ **24**	54 $\pm$ 20	0 $\pm$ 0	210 $\pm$ 79	119 $\pm$ 65	213 $\pm$ 111	102 $\pm$ 92
	SA-DQN	1245 $\pm$ 14	**1245** $\pm$ **10**	1176 $\pm$ 63	1148 $\pm$ 36	1024 $\pm$ 31	1054 $\pm$ 11	489 $\pm$ 106
	RADIAL-DQN	1178 $\pm$ 4	1178 $\pm$ 4	1176 $\pm$ 63	1049 $\pm$ 27	928 $\pm$ 113	1035 $\pm$ 46	508 $\pm$ 85
	WocaR-DQN (Ours)	1220 $\pm$ 12	1220 $\pm$ 3	**1214** $\pm$ **7**	**1192** $\pm$ **12**	**1045** $\pm$ **20**	**1096** $\pm$ **19**	**754** $\pm$ **102**
RoadRunner	DQN	**45527** $\pm$ **4894**	0 $\pm$ 0	0 $\pm$ 0	14962 $\pm$ 6431	2985 $\pm$ 1440	842 $\pm$ 41	203 $\pm$ 65
	SA-DQN	44638 $\pm$ 2367	43970 $\pm$ 975	20678 $\pm$ 1563	39736 $\pm$ 2315	4214 $\pm$ 2587	38432 $\pm$ 3574	5516 $\pm$ 4684
	RADIAL-DQN	44675 $\pm$ 5854	**44605** $\pm$ **1094**	38576 $\pm$ 1960	38060 $\pm$ 1799	8476 $\pm$ 3964	36310 $\pm$ 9149	1290 $\pm$ 4015
	WocaR-DQN (Ours)	44156 $\pm$ 2279	44079 $\pm$ 2154	**38720** $\pm$ **1765**	**40758** $\pm$ **3369**	**10545** $\pm$ **2984**	**38954** $\pm$ **3647**	**8239** $\pm$ **2766**

Table 3: Average episode rewards $\pm$ standard deviation over 1000 episodes on baselines and WocaR-DQN on Pong, Freeway, BankHeist, and RoadRunner. Natural reward and rewards under different attacks with ϵ of 1/255 and 3/255 are reported. We bold the best results for each evaluation metric. And the row for the most robust agents on all environments are highlighted by gray.



(a) Hopper: Robust Sarsa **(b)** Hopper: SA-RL **(c)** Hopper: PA-AD

(d) Walker2d: Robust Sarsa **(e)** Walker2d: SA-RL **(f)** Walker2d: PA-AD

Figure 8: Comparisons under different attacks w.r.t. different budget ϵ's on Hopper and Walker2d.

Environment	Model	Natural Reward	PGD (10 steps) ϵ=1/255	PGD (10 steps) ϵ=3/255	MinBest ϵ=1/255	MinBest ϵ=3/255	PA-AD ϵ=1/255	PA-AD ϵ=3/255
BankHeist	A2C	**1228 ± 93**	67 ± 14	0 ± 0	972 ± 99	697 ± 153	636 ± 74	314 ± 116
	SA-A2C	1029 ± 152	1029 ± 156	976 ± 54	902 ± 89	786 ± 52	836 ± 70	644 ± 153
	PA-ATLA-A2C	1076 ± 56	1075 ± 79	1013 ± 69	957 ± 78	842 ± 154	862 ± 106	757 ± 132
	WocaR-A2C (Ours)	1089 ± 34	**1089 ± 78**	**1035 ± 102**	**1043 ± 29**	**937 ± 65**	**1004 ± 94**	**879 ± 128**

Table 4: Average episode rewards ± standard deviation over 1000 episodes on baselines and VaR-A2C on BankHeist. Natural reward and rewards under different attacks with ϵ of 1/255 and 3/255 are reported. We bold the best results for each evaluation metric. And the row for the most robust agents on all environments are highlighted by gray.

D.3 Additional Evaluation and Ablation Studies

D.3.1 Robustness Evaluation Using Multiple ϵ

To study how WocaR-PPO performs under attacks with different value of ϵ, Figure 8 shows the evaluation of our algorithms under different ϵ attacks compared with the baselines in Hopper and Walker2d. We can conclude that our robustly trained model universally and significantly outperforms other robust agents considering various attack budget ϵ.

D.3.2 Additional Evaluation on Sample Efficiency

In Table 5, we report the performance of WocaR-PPO and all robust PPO baselines using the same training steps. We find that under limited training steps, ATLA-PPO, PA-ATLA-PPO and RADIAL-PPO obtain sub-optimal robustness, which suggests that these methods are more sample-hungry. In contrast, WocaR-PPO converges under fewer steps and achieves best performance with a large advantage, which shows the higher efficiency of WocaR-PPO.

D.3.3 Additional Results of Time Efficiency

We show the training efficiency of WocaR-PPO from three aspects including time, training iterations, and sampling in MuJoCo environments by comparing with SA-PPO and state-of-the-art methods ATLA-PPO, PA-ATLA-PPO, and RADIAL-PPO in Table 6. For a fair comparison, we use the same *GeForce RTX 1080 Ti GPUs* to train all the robust agents.

Environment	Model	Natural Reward	Random	MAD	RS	SA-RL	PA-AD
Halfcheetah state-dim: 17 ϵ=0.15	ATLA-PPO	4817 $\pm$ 277	4809 $\pm$ 186	4584 $\pm$ 100	4074 $\pm$ 285	4129 $\pm$ 348	1856 $\pm$ 294
	PA-ATLA-PPO	5023 $\pm$ 282	5076 $\pm$ 149	4720 $\pm$ 334	4392 $\pm$ 158	4159 $\pm$ 248	3085 $\pm$ 295
	RADIAL-PPO	4683 $\pm$ 97	4625 $\pm$ 190	3674 $\pm$ 222	3529 $\pm$ 173	2893 $\pm$ 165	2197 $\pm$ 251
	WocaR-PPO (Ours)	**6032 $\pm$ 68**	**5969 $\pm$ 149**	**5850 $\pm$ 228**	**5319 $\pm$ 220**	**5365 $\pm$ 54**	**4269 $\pm$ 172**
Hopper state-dim: 11 ϵ=0.075	ATLA-PPO	3265 $\pm$ 342	3195 $\pm$ 275	2675 $\pm$ 332	2098 $\pm$ 398	1542 $\pm$ 639	1135 $\pm$ 289
	PA-ATLA-PPO	3429 $\pm$ 196	3455 $\pm$ 315	3072 $\pm$ 478	2889 $\pm$ 258	1458 $\pm$ 274	2032 $\pm$ 244
	RADIAL-PPO	**3687 $\pm$ 80**	3627 $\pm$ 106	2952 $\pm$ 126	1094 $\pm$ 248	1243 $\pm$ 187	1036 $\pm$ 142
	WocaR-PPO (Ours)	3616 $\pm$ 99	**3633 $\pm$ 30**	**3541 $\pm$ 207**	**3277 $\pm$ 159**	**2390 $\pm$ 145**	**2579 $\pm$ 229**
Walker2d state-dim: 17 ϵ=0.05	ATLA-PPO	2664 $\pm$ 366	2695 $\pm$ 320	2547 $\pm$ 210	2439 $\pm$ 174	2092 $\pm$ 144	1544 $\pm$ 280
	PA-ATLA-PPO	3047 $\pm$ 223	3112 $\pm$ 111	2865 $\pm$ 230	2742 $\pm$ 177	2450 $\pm$ 229	1987 $\pm$ 246
	RADIAL-PPO	2143 $\pm$ 153	2231 $\pm$ 89	2095 $\pm$ 121	1680 $\pm$ 193	1078 $\pm$ 115	1274 $\pm$ 117
	WocaR-PPO (Ours)	**4156 $\pm$ 495**	**4244 $\pm$ 157**	**4177 $\pm$ 176**	**4093 $\pm$ 138**	**3770 $\pm$ 196**	**2722 $\pm$ 173**
Ant state-dim: 111 ϵ=0.15	ATLA-PPO	4249 $\pm$ 243	4218 $\pm$ 161	4036 $\pm$ 173	3391 $\pm$ 158	2045 $\pm$ 203	-349 $\pm$ 175
	PA-ATLA-PPO	4533 $\pm$ 238	4492 $\pm$ 190	4232 $\pm$ 203	3579 $\pm$ 261	2762 $\pm$ 152	1765 $\pm$ 185
	RADIAL-PPO	4379 $\pm$ 230	4194 $\pm$ 52	3278 $\pm$ 138	2348 $\pm$ 232	1380 $\pm$ 145	157 $\pm$ 124
	WocaR-PPO (Ours)	**5596 $\pm$ 225**	**5558 $\pm$ 241**	**5284 $\pm$ 182**	**4339 $\pm$ 160**	**3822 $\pm$ 185**	**3164 $\pm$ 163**

Table 5: Average episode rewards $\pm$ standard deviation over 50 episodes on baselines and WocaR-PPO trained for 2 million steps on Hopper, Walker2d, Halfcheetah and 7.5 million steps on Ant (less than the best settings). **Bold** numbers indicate the best results under each attack. The gray rows are the most robust agents.

Model	Hopper		Ant	
	Time (h)	Steps(m)	Time (h)	Steps (m)
SA-PPO	3.0	2.0	8.9	10.0
ATLA-PPO	5.6	5.0	12.8	10.0
PA-ATLA-PPO	5.2	5.0	12.3	10.0
RADIAL-PPO	3.2	4.0	10.2	10.0
WocaR-PPO (Ours)	**2.3**	**2.0**	**8.7**	**7.5**

Table 6: Efficiency comparison of state-of-the-art robust training methods and WocaR-PPO in Hopper and Ant. For Walker2d and Halfcheetah, the sampling steps are same as for Hopper and the training time is also extremely similar. We highlight the most efficient method as gray.

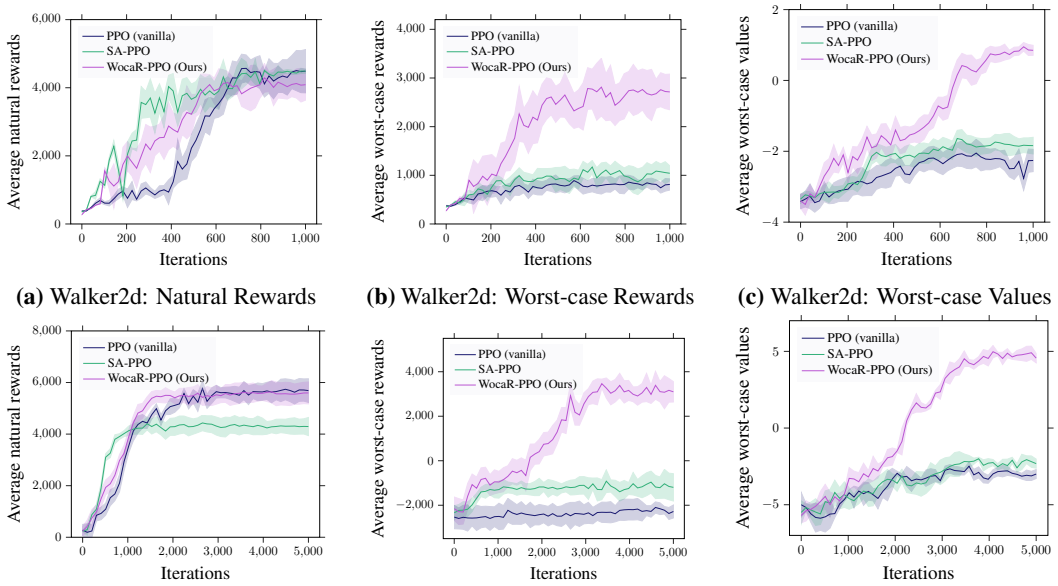
It needs to mention that in continuous action spaces when estimating the worst-case value, we solve $\min_{\hat{a}\in\hat{\mathcal{A}}_{\text{adv}}} \underline{Q}^{\pi}_{\phi}(s_{t+1}, \hat{a})$ using 50-step gradient descent. The running time of this 50-step gradient descent is about **1.68 seconds** per batch with batch size 128. In total, this gradient descent computation takes 18% of the total training time, thus it is not the computation bottleneck.

Without training with an adversary, *our algorithm requires much less (only 50% or 75%) steps to reliably converge.* WocaR-PPO only takes less than half of time for low-dimensional environments to converge compared to ATLA methods and RADIAL-PPO. In high-dimensional environments like Ant, we only need 4 hours for training, while ATLA methods require at least 7 hours. When solving harder tasks, the efficiency advantage of WocaR-PPO is more obvious.

D.3.4 Effectiveness of Worst-attack Policy Optimization

In addition to Figure 6(a) and (b), we show the learning curves in Walker2d and Ant in Figure 9 to verify the effectiveness of worst-attack value estimation and worst-case-aware policy optimization. Figure 9(a) and (d) show the natural rewards of agents during training without attacks. The actual worst-attack rewards in Figure 9(b) and (e) refer to the the reward obtained by the agents under PA-AD attack [41] which is the existing strongest attacking algorithm. To study the worst-case performance during training, We evaluate PPO, SA-PPO and WocaR-PPO agents after every 20 iterations using all types of attacks and report the worst-case rewards for each checkpoint. We also present the trend of the estimated worst-case values during training in Figure 9(c) and (f), which are tested by the trained worst-attack value functions $\underline{Q}^{\pi}_{\phi}$.

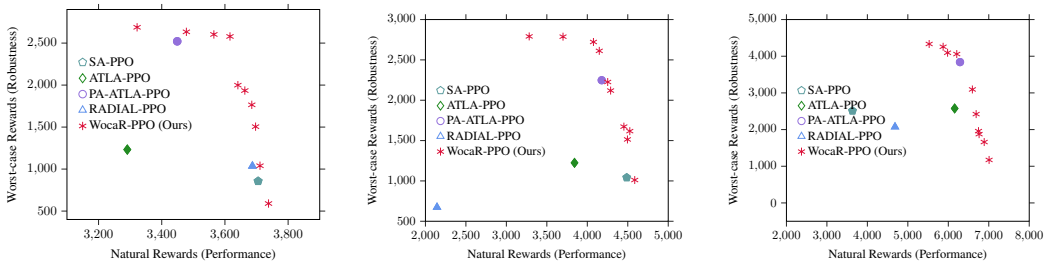
We observe from the curves that our worst-attack critic estimation matches the trend of actual worst-attack rewards. Also, the increases of estimated worst-attack values and actual worst-attack rewards of



(a) Walker2d: Natural Rewards **(b)** Walker2d: Worst-case Rewards **(c)** Walker2d: Worst-case Values

(d) Ant: Natural Rewards **(e)** Ant: Worst-case Rewards **(f)** Ant: Worst-case Values

Figure 9: Learning curves (mean ± standard deviation) of natural rewards, worst-case rewards under attacks and estimated worst-case values during training on Walker2d and Ant for vanilla PPO (blue), SA-PPO (green) and WocaR-PPO (purple).



(a) Hopper **(b)** Walker2d **(c)** Halfcheetah

Figure 10: Average natural rewards and worst-case rewards of WocaR-PPO with different κ_{wst} and other baselines on Hopper, Walker2d, and Halfcheetah.

WocaR-PPO show that our WocaR-RL significantly improves the robustness of agents by enhancing worst-attack values.

D.3.5 Trade-off between Natural Performance and Robustness

As mentioned in Section 5.2, the adjustable weight κ_{wst} controls the trade-off between natural performance and robustness. To discuss the effect of κ_{wst}, we train agents using WocaR-PPO in Hopper, Walker2d, and Halfcheetah with uniformly sampled 40 different values of weight κ_{wst} in range $(0, 1]$.

Figure 10 plots the worst-case performance and natural performance of robust training baselines and 10 agents trained by WocaR-PPO. We can see that when reward under worst-case perturbations increases, it leads to a reduction of the natural reward. Using different κ_{wst}'s, we balance the robustness and performance and report the results in Table 2 with significant better worst-case robustness and comparable natural performance compared with baselines. WocaR-PPO can always find policies which dominate other robust agents.

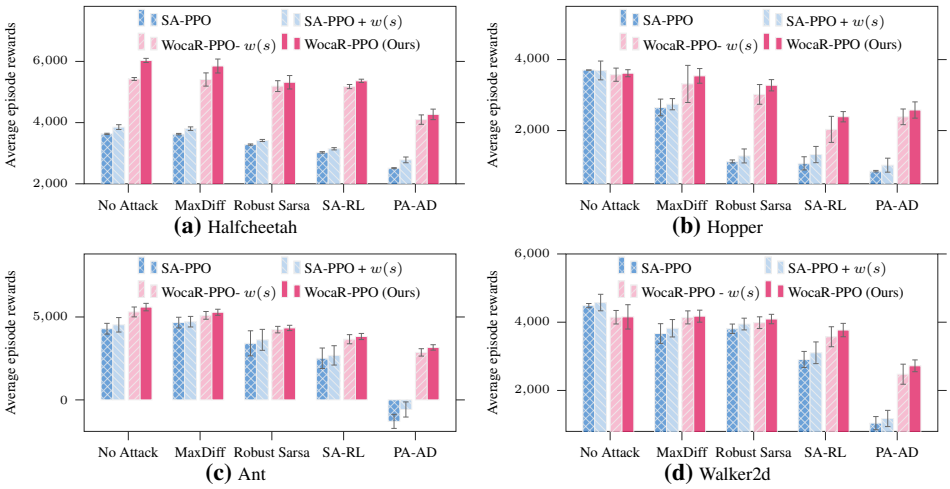


Figure 11: Ablation performance for the state importance weight $w(s)$ under no attack and different attacks on Hopper, Walker2d, Halfcheetah, and Ant.

D.3.6 Additional Ablation Studies

We conduct ablation experiments to analyze the effect of two techniques: our proposed state importance weight $w(s)$ and the state regularization loss $\mathcal{L}_{\text{reg}}$ [52].
For the state importance weight $w(s)$, we compare the performance of the original WocaR-PPO and WocaR-PPO without $w(s)$ in Figure 11. In addition, since SA-PPO [52] also uses a state regularization technique, we also equip SA-PPO with the state importance with $w(s)$, to demonstrate the universal applicability of this design. In all four MuJoCo environments, we can see that with $w(s)$, both WocaR-PPO and SA-PPO get boosted robustness, verifying the effectiveness of the state importance weight.
For the state regularization loss $\mathcal{L}_{\text{reg}}$, Figure 12 verifies that $\mathcal{L}_{\text{reg}}$ enhances the robustness of WocaR-PPO, since the performance of WocaR-PPO drops without $\mathcal{L}_{\text{reg}}$.

On the other hand, Figure 12 also compares the performance of ATLA methods and our algorithm without $\mathcal{L}_{\text{reg}}$ (note that ATLA methods also regularizes the PPO policies during training). The results indicate that *the decisive contribution of WocaR-PPO to robustness improving comes from the worst-attack-aware policy optimization.*

These ablation studies demonstrate that all the techniques are beneficial for robustness improvement and further show that our worst-case-aware training performs better than training with attackers.

E Limitations and Potential Societal Impacts

This paper proposes an efficient and effective robust training framework for RL. But there are several aspects to improve or extend the current approach. The proposed worst-attack Bellman operator in theory gives the exact worst-case value of a policy under ℓ_p bounded attacks. But in practice, it is hard to compute the set $\mathcal{A}_{\text{adv}}$ directly, so we use convex relaxation to obtain a superset of it, $\hat{\mathcal{A}}_{\text{adv}}$. As a result, the fixed point of worst-attack Bellman operator with $\mathcal{A}_{\text{adv}}$ being replaced by $\hat{\mathcal{A}}_{\text{adv}}$ is a lower bound of the worst-case value. Generally, the tighter the convex relaxation is, the tighter the lower bound could be. Then, our algorithm increases the worst-case value by improving its lower bound. In addition, as is common in deep RL and adversarial training works, the convergence of the algorithm with several inter-dependent networks is not well understood yet. Our future work will focus on the above limitations, and develop a deeper theoretical understanding of robust trainingh. In addition, this paper only considers the ℓ_p threat model as is common in most related works. But in real-world applications, other attack models could exist (e.g. patch attack), and improving the robustness of RL agents in these scenarios is another important research direction.

This work focuses on improving the robustness of deep RL agents, which can make RL models more reliable in high-stakes applications. Although it is generally positive for the community to build more

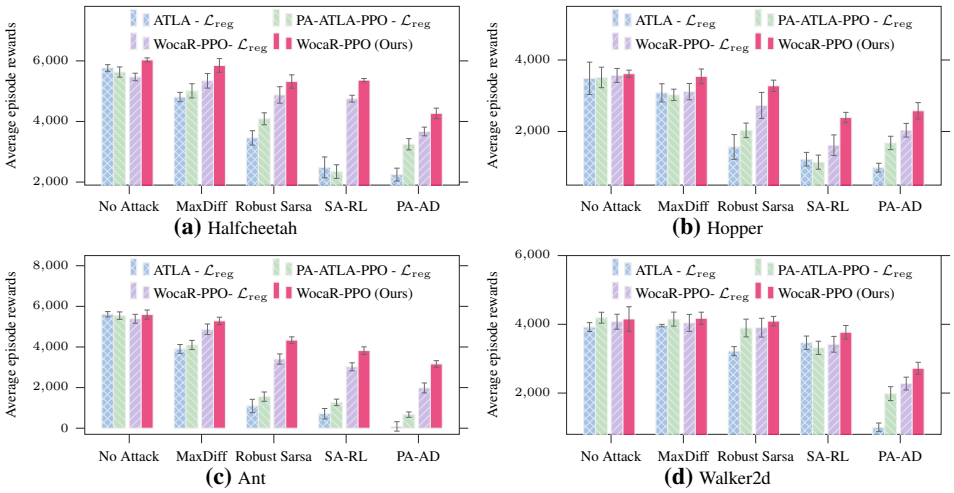


Figure 12: Ablation performance for the state regularization loss $\mathcal{L}_{\text{reg}}$ under no attack and different attacks on Hopper, Walker2d, Halfcheetah, and Ant.

robust agents, such robust agents may also bring some potentially negative impacts, including the possibility of robust robots replacing some occupations and causing mass unemployment.