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

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

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,

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

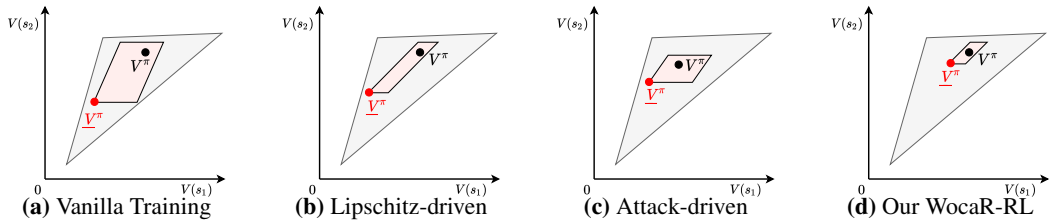
so it is a contraction in the sup-norm.

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 $\mathcal{A}_{\mathrm{adv}}$. 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 8: Geometric understanding of different training methods following the polytope theory by [8] and [42]. 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 [14] in supervised learning. For example, Fischer et al. [9] achieve provable robustness for DQN by applying the DiffAI [31] approach, so that the DQN agent selects the same action for any element inside $\mathcal{B}_\epsilon(s)$. Zhang et al. [54] 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) [28]. 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. [52] 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. [52] and a follow-up work by Sun et al. [42] 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 [27, 49, 23] 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. [8] point out that the value functions of all policies in a finite MDP form a polytope, as shown by the grey area in Figure 8. Sun et al. [42] 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 8), 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 8(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 8 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) [48, 15] 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 [54] 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 [33].

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

In policy-based DRL methods [38, 25, 39] 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 ($\mathcal{L}_{\rm RL}$) and the worst-attack action value $\underline{Q}^\pi_\phi$ ($\mathcal{L}_{\rm wst}$) can be increased ($\mathcal{L}_{\rm RL}$ and $\mathcal{L}_{\rm wst}$). At the same time, π_θ is also regularized by $\mathcal{L}_{\rm reg}$.

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 $\mathcal{L}_{\rm reg}$ is the same as the SA-regularizer [54]. 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 [32, 16, 47] 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 ($\mathcal{L}_{\rm RL}$ and $\mathcal{L}_{\rm wst}$), while enforcing the network to output the same action under bounded state perturbations ($\mathcal{L}_{\rm reg}$).

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 $\hat{\mathcal{A}}_{\rm adv}$ to compute the worst-case TD loss $\mathcal{L}_{\rm est}$ (Line 11 to 15). The implementation of $\mathcal{L}_{\rm reg}$ is the same as the SA-regularizer [54], 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 $\mathcal{L}_{\rm reg}$ 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 $\mathcal{L}_{\rm est}$ (Line 6 to 9). The solutions for state importance weight $w(s_t)$ and regularization $\mathcal{L}_{\rm reg}$ 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 [35, 44, 45]. 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 [54] 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:

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

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 [12]) (from SA-PPO [54]):

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

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

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

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 [54] 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}

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

Initialize replay buffer $\mathcal{B}$

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

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

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

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

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

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

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}

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

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 $Q_u(s_{t+1}, a, \epsilon_t)$.

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 $\hat{\mathcal{A}}_{\text{adv}}$

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

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

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

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

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 $y_i^r = r_i$

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 [54] 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 [54] 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:

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

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 [53] by SGLD (Stochastic gradient Langevin dynamics [12]):

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

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 [33]. 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 [52]. 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 [54] and [33] in hyperparameters and network structures for vanilla DQN training. The implementation of all our baselines applies Double DQN [17] and Prioritized Experience Replay [37]. 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 [54].
RADIAL-DQN Following the original implementation from [33], 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 [19].
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 [54, 52]. 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.

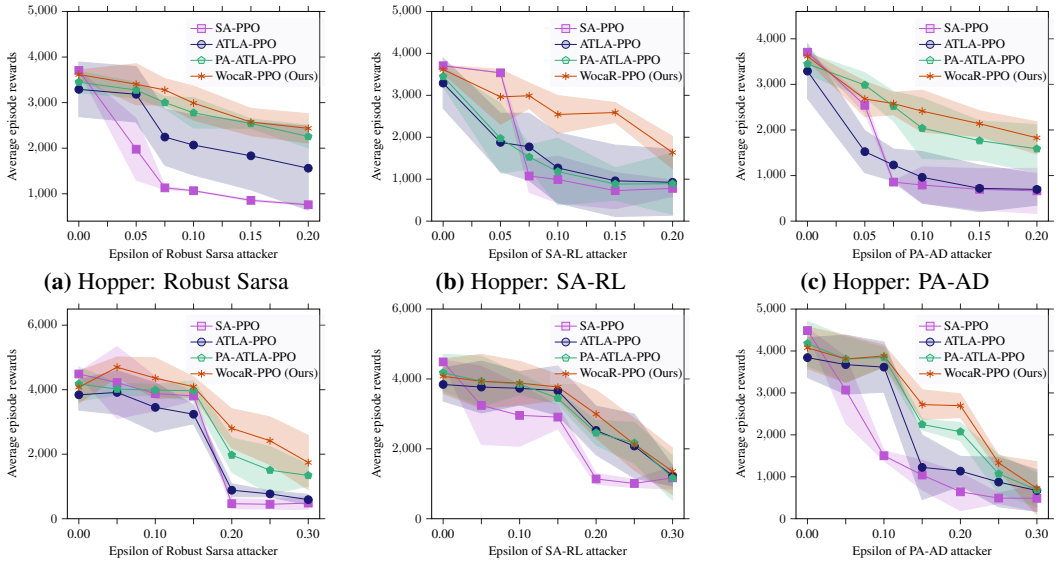
Environment	Model	Natural Reward	Random	MAD	RS	SA-RL	PA-AD
Halfcheetah state-dim: 17 ϵ=0.15	PPO (vanilla)	**7117 ± 98**	5486 ± 1378	1836 ± 866	489 ± 758	-660 ± 218	-356 ± 407
	SA-PPO	3632 ± 20	3619 ± 18	3624 ± 23	3283 ± 20	3028 ± 23	2512 ± 16
	ATLA-PPO	6157 ± 852	6164 ± 603	5790 ± 174	4806 ± 392	5058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

Table 2: Average episode rewards ± 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[54] and PA-ATLA-A2C[42] 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 ± 0.0	-21.0 ± 0.0	-21.0 ± 0.0	-7.4 ± 2.8	-9.7 ± 4.0	-18.2 ± 2.3	-19.0 ± 2.2
	SA-DQN	21.0 ± 0.0	21.0 ± 0.0	21.0 ± 0.0	21.0 ± 0.0	20.6 ± 3.5	20.4 ± 1.8	18.7 ± 2.6
	RADIAL-DQN	21.0 ± 0.0	21.0 ± 0.0	21.0 ± 0.0	21.0 ± 0.0	19.5 ± 2.1	20.3 ± 2.5	13.2 ± 1.8
	WocaR-DQN (Ours)	**21.0 ± 0.0**	**21.0 ± 0.0**	**21.0 ± 0.0**	**21.0 ± 0.0**	**20.8 ± 3.3**	**21.0 ± 0.2**	**19.7 ± 2.4**
Freeway	DQN	**34.0 ± 0.1**	0.0 ± 0.0	0.0 ± 0.0	9.5 ± 3.0	5.5 ± 1.8	9.3 ± 2.7	4.7 ± 2.9
	SA-DQN	30.0 ± 0.0	30.0 ± 0.0	30.0 ± 0.0	27.2 ± 3.4	18.3 ± 3.0	20.1 ± 4.0	9.5 ± 3.8
	RADIAL-DQN	33.1 ± 0.2	**33.1 ± 0.2**	**33.2 ± 0.2**	22.6 ± 3.3	16.4 ± 2.3	18.5 ± 4.2	10.8 ± 3.6
	WocaR-DQN (Ours)	31.2 ± 0.4	31.2 ± 0.5	31.4 ± 0.3	**29.6 ± 2.5**	**19.8 ± 3.8**	**24.9 ± 3.7**	**12.3 ± 3.2**
BankHeist	DQN	**1308 ± 24**	54 ± 20	0 ± 0	210 ± 79	119 ± 65	213 ± 111	102 ± 92
	SA-DQN	1245 ± 14	**1245 ± 10**	1176 ± 63	1148 ± 36	1024 ± 31	1054 ± 11	489 ± 106
	RADIAL-DQN	1178 ± 4	1178 ± 4	1176 ± 63	1049 ± 27	928 ± 113	1035 ± 46	508 ± 85
	WocaR-DQN (Ours)	1220 ± 12	1220 ± 3	**1214 ± 7**	**1192 ± 12**	**1045 ± 20**	**1096 ± 19**	**754 ± 102**
RoadRunner	DQN	**45527 ± 4894**	0 ± 0	0 ± 0	14962 ± 6431	2985 ± 1440	842 ± 41	203 ± 65
	SA-DQN	44638 ± 2367	43970 ± 975	20678 ± 1563	39736 ± 2315	4214 ± 2587	38432 ± 3574	5516 ± 4684
	RADIAL-DQN	44675 ± 5854	**44605 ± 1094**	38576 ± 1960	38060 ± 1799	8476 ± 3964	36310 ± 9149	1290 ± 4015
	WocaR-DQN (Ours)	44156 ± 2279	44079 ± 2154	**38720 ± 1765**	**40758 ± 3369**	**10545 ± 2984**	**38954 ± 3647**	**8239 ± 2766**

Table 3: Average episode rewards ± 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 9: 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 9 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.

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

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

Table 5: Average episode rewards ± 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.

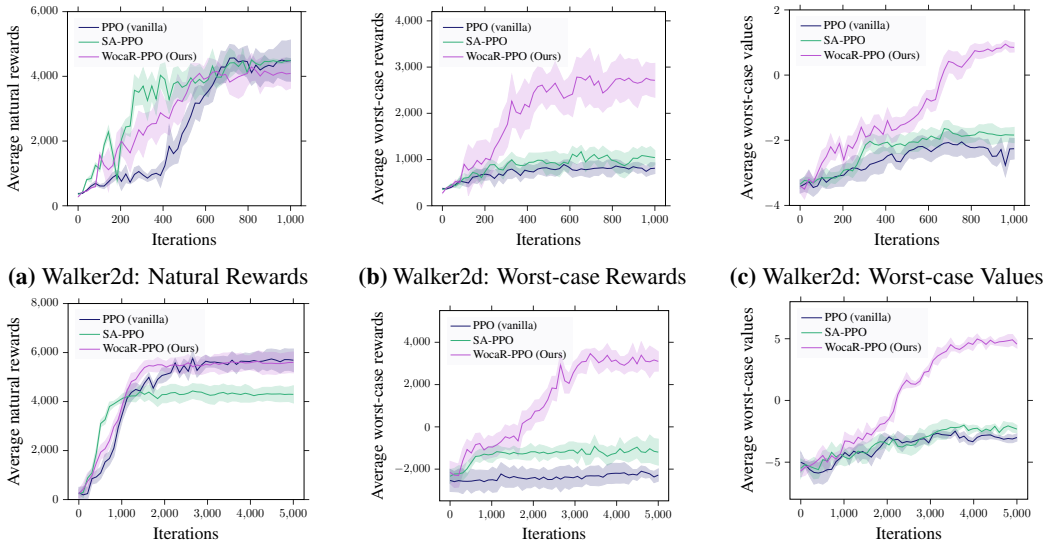
gradient descent is about **1.68 seconds** per batch with batch size 128. In total, this gradient descent computation takes 18% of the total training time, thus it is not the computation bottleneck.

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, we show the learning curves in Walker2d and Ant in Figure 10 to verify the effectiveness of worst-attack value estimation and worst-case-aware policy optimization. Figure 10(a) and (d) show the natural rewards of agents during training without attacks. The actual worst-attack rewards in Figure 10(b) and (e) refer to the the reward obtained by the agents under PA-AD attack [42] 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 10(c) and (f), which are tested by the trained worst-attack value functions $\underline{Q}^{\pi}_{\phi}$.

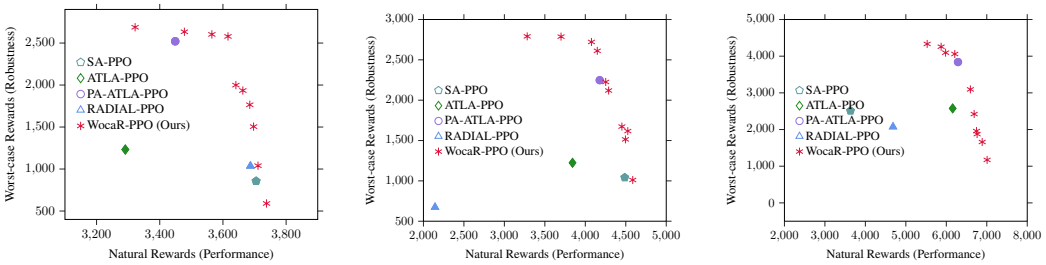
We observe from the curves that our worst-attack critic estimation matches the trend of actual worst-attack rewards. Also, the increases of estimated worst-attack values and actual worst-attack rewards of WocaR-PPO show that our WocaR-RL significantly improves the robustness of agents by enhancing worst-attack values.



(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 10: 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 11: Average natural rewards and worst-case rewards of WocaR-PPO with different κ_{wst} and other baselines on Hopper, Walker2d, and Halfcheetah.

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 11 plots the worst-case performance and natural performance of robust training baselines and 10 agents trained by WocaR-PPO with various values of κ_{wst}. We can see that when reward under worst-case perturbations increases, it leads to a reduction of the natural reward.

The choice of the worst-case value's weight κ_{wst} is to control the trade-off between the final natural performance and robustness. It does not affect the convergence of the algorithm. When we increase the weight of worst-case values κ_{wst}, the reward under worst-case perturbations increases, but it leads to a reduction of the natural reward. Equally, when κ_{wst} is set close to 0, the algorithm is similar to standard training, where the policy achieves high reward under no attack, but extremely low reward under attacks. Hence, κ_{wst} is necessary for our algorithm to balance these two kinds of performance. In practice, one can adjust κ_{wst} according to their preferences to robustness and natural performance.

We 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.

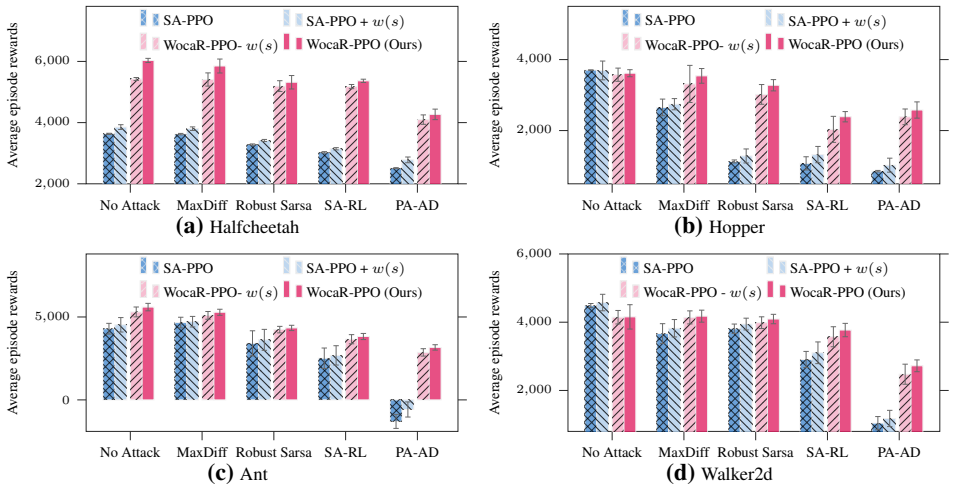


Figure 12: Ablation performance for the state importance weight $w(s)$ under no attack and different attacks on Hopper, Walker2d, Halfcheetah, and Ant.

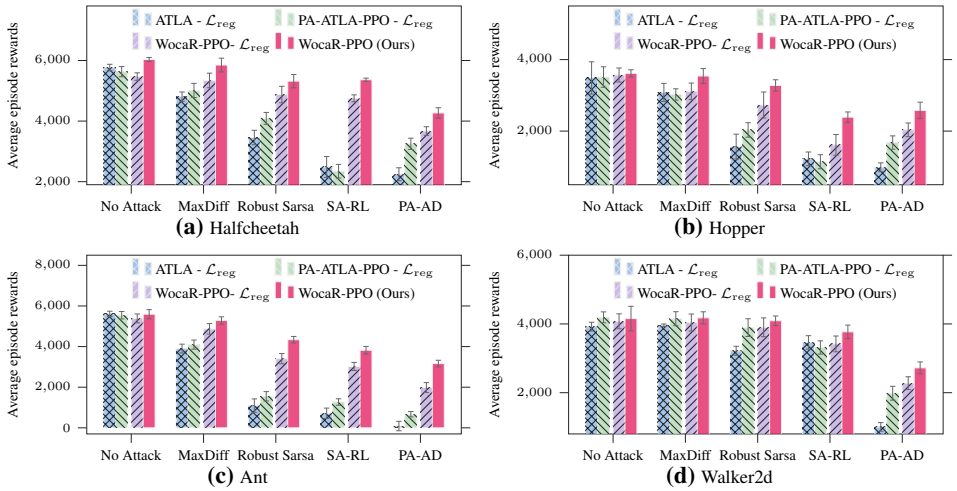


Figure 13: Ablation performance for the state regularization loss $\mathcal{L}_{\text{reg}}$ under no attack and different attacks on Hopper, Walker2d, Halfcheetah, and Ant.

D.3.6 Additional Ablation Studies

We provide full ablation experimental results for the state importance weight $w(s)$ and the regularization loss $\mathcal{L}_{\text{reg}}$ [54] on four MuJoCo environments.

For the state importance weight $w(s)$, we compare the performance between the original WocaR-PPO and WocaR-PPO without $w(s)$ in Figure 12. Additionally, we also equip SA-PPO with $w(s)$ to show 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 13 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 13 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 Potential Societal Impacts

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 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.