Robust Reinforcement Learning via Adversarial Kernel Approximation

Kaixin Wang[*,†] **Uri Gadot**[*,†]

Navdeep Kumar[†] **Kfir Levy**[†] **Shie Mannor**[†,‡]

Abstract

Robust Markov Decision Processes (RMDPs) provide a framework for sequential decision-making that is robust to perturbations on the transition kernel. However, robust reinforcement learning (RL) approaches in RMDPs do not scale well to realistic online settings with high-dimensional domains. By characterizing the adversarial kernel in RMDPs, we propose a novel approach for online robust RL that approximates the adversarial kernel and uses a standard (non-robust) RL algorithm to learn a robust policy. Notably, our approach can be applied on top of any underlying RL algorithm, enabling easy scaling to high-dimensional domains. Experiments in classic control tasks, MinAtar and DeepMind Control Suite demonstrate the effectiveness and the applicability of our method.

1 Introduction

In reinforcement learning (RL), we are concerned with learning good policies for sequential decision-making problems modeled as Markov Decision Processes (MDPs) [30, 37]. MDPs assume that the transition model of the environment is fixed across training and testing, but this is often violated in practical applications. For example, when deploying a simulator-trained robot in reality, a notable challenge is the substantial disparity between the simulated environment and the intricate complexities of the real world, leading to potential subpar performance upon deployment. Such a mismatch may significantly degrade the performance of the trained policy in testing. To deal with this issue, the robust MDP (RMDP) framework has been introduced in [17, 25, 46], aiming to learn policies that are robust to perturbation of the transition model within an uncertainty set.

Existing works on learning robust policies in RMDPs often suffer from poor scalability when it comes to online learning in high-dimensional domains. Specifically, model-based methods that solve RMDPs [3, 7, 10, 15, 22, 46] require access to the nominal transition probability, making it difficult to scale beyond tabular settings. While some recent works [22, 23, 44, 45] introduce model-free methods that add regularization to the learning process, the effectiveness of their methods are not validated in high-dimensional environments. In addition, these methods are based on particular RL algorithms (e.g., , policy gradient, Q learning), limiting their general applicability. We defer a more detailed discussion on related works to Section 5.

In this work, we tackle the problem of learning robust policies in RMDPs from an alternative direction. As shown in Figure 1, unlike previous works that explicitly regularize the learning process, we propose to approximately simulate the adversarial transition kernel while leaving the RL part untouched. In the context of RMDPs, the adversarial kernel is the one within the uncertainty set that

[*]Equal Contribution
[†]Technion
[‡]Nvidia Research

16th European Workshop on Reinforcement Learning (EWRL 2023).

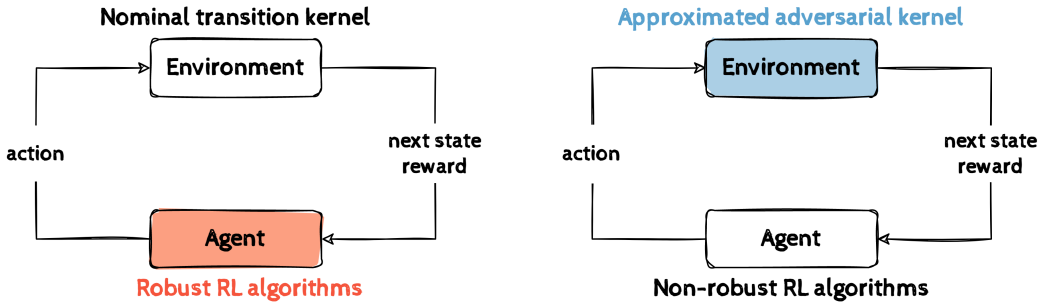

Figure 1: The agent-environment interaction loop during training. **Left**: Existing methods typically regularize how an agent updates its policy to improve robustness. **Right**: Our work approximates the adversarial transition kernel, so the agent essentially learns its policy under the worst scenarios and can use any non-robust RL algorithm.

leads to the worst possible return (see Definition 3.1). Intuitively, the idea is to situate the agent in the approximately worst scenarios for learning policies robust to perturbations. Our method can be applied on top of any (deep) RL algorithm, offering good scalability to high-dimensional domains.

Specifically, our work builds upon recent results on characterizing the adversarial transition kernel in RMDPs [23]. Kumar et al. [23] reveal that for L_p uncertainty set, the adversarial kernel essentially modifies the next-state transition probability of the nominal kernel (e.g., , penalizes the probability of transiting to states with high values). If we are able to simulate such modifications and sample next states according to the adversarial kernel, then the policy would be effectively trained under the approximated adversarial kernel and become more robust. However, for domains with large state spaces, the use of L_p uncertainty imposes some limitations on approximating the adversarial kernel (see Section 3). To circumvent these limitations, we consider another commonly used uncertainty set, the KL uncertainty set [25, 27, 36, 47]. We characterize the adversarial kernel under KL uncertainty and show that it has a form amenable to simulation.

To verify the effectiveness of our method, we conduct experiments on multiple environments ranging from small-scale classic control tasks to high-dimensional MinAtar games [49] and DeepMind Control Suite [40]. The agent is trained in the nominal environment and tested in environments with perturbed transitions. Since our method is agnostic to the underlying RL algorithm, we can easily plug it into a double DQN [41] agent for discrete-action environments or a SAC [12] agent for continuous-action environments. Experiment results demonstrate that with our method, the learned policy suffers from less performance degradation when the transition kernel is perturbed.

In summary, our paper makes the following contributions:

- To learn robust policies in RMDPs, we propose to approximately simulate the adversarial transition kernel, rather than regularizing the learning process. This opens up a new paradigm for learning robust policies in RMDPs.
- We theoretically characterize the adversarial kernel under the KL uncertainty set, which is amenable to approximate simulation for environments with large state spaces.
- Our method is not tied to a particular RL algorithm and can be easily integrated with any deep RL method. This flexibility translates to the good scalability of our method in complex high-dimensional domains such as MinAtar and DeepMind Control Suite. To the best of our knowledge, our work is the first among related works in RMDPs that enjoys such flexibility.

2 Preliminaries

Notations. For a finite set $\mathcal{Z}$, we write the probability simplex over it as $\Delta_{\mathcal{Z}}$. Given two real functions $f, g : \mathcal{Z} \to \mathbb{R}$, their inner product is $\langle f, g \rangle = \sum_{z \in \mathcal{Z}} f(z)g(z)$. The L_p norm is denoted by $\|\cdot\|_p$. For distributions P, Q, we denote the Kullback–Leibler (KL) divergence of P from Q by $D_{\mathrm{KL}}(P \,\|\, Q)$.

2.1 Markov Decision Processes

A Markov decision process (MDP) [30, 37] is a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma, \mu)$, where $\mathcal{S}$ and $\mathcal{A}$ are the state space and the action space respectively, $P : \mathcal{S} \times \mathcal{A} \to \Delta_{\mathcal{S}}$ is the transition kernel, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $\gamma \in [0, 1)$ is the discount factor, and $\mu \in \Delta_{\mathcal{S}}$ is the initial state distribution. A stationary policy $\pi : \mathcal{S} \to \Delta_{\mathcal{A}}$ maps a state to a probability distribution over $\mathcal{A}$. We use $P(\cdot|s, a) \in \Delta_{\mathcal{S}}$ to denote the probabilities of transiting to the next state when the agent takes action a at state s. For a policy π, we denote the expected reward and transition by:

$$R^\pi(s) = \sum_a \pi(a|s) R(s, a), \quad P^\pi(s'|s) = \sum_a \pi(a|s) P(s'|s, a). \tag{1}$$

The value function $v^\pi : \mathcal{S} \to \mathbb{R}$ maps a state to the expected cumulative reward when the agent starts from that state and follows policy π, i.e., ,

$$v^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \,\middle|\, s_0 = s, a_t \sim \pi(\cdot|s_t), s_{t+1} \sim P(\cdot|s_t, a_t) \right]. \tag{2}$$

It is known that v^π is the unique fixed point of the Bellman operator T^π [31],

$$T^\pi v = R^\pi + \gamma P^\pi v. \tag{3}$$

The agent's objective is to obtain a policy π^* that maximizes the discounted return

$$J^\pi = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \,\middle|\, s_0 \sim \mu, a_t \sim \pi(\cdot|s_t), s_{t+1} \sim P(\cdot|s_t, a_t) \right] = \langle \mu, v^\pi \rangle. \tag{4}$$

2.2 Robust Markov Decision Processes

In MDPs, the system dynamic P is usually assumed to be constant over time. However, in real-life scenarios, it is subject to perturbations, which may significantly impact the performance in deployment [24]. Robust MDPs (RMDPs) provide a theoretical framework for taking such uncertainty into consideration, by taking P as not fixed but chosen adversarially from an uncertainty set $\mathcal{P}$ [17, 25]. Since we may consider different dynamics P in the RMDPs context, in the following, we will use subscript P to make the dependency explicit. The objective in RMDPs is to obtain a policy $\pi^*_{\mathcal{P}}$ that maximizes the robust return

$$J^\pi_{\mathcal{P}} = \min_{P \in \mathcal{P}} J^\pi_P. \tag{5}$$

However, solving RMDPs for general uncertainty sets is NP-hard while the optimal policy can be non-stationary [46]. To make RMDPs tractable, we need to make some assumptions about the uncertainty set.

2.3 Rectangular uncertainty set

One commonly used assumption to enable tractability for RMDPs is rectangularity. Specifically, we assume that the uncertainty set $\mathcal{P}$ can be factorized over states-actions:

$$\mathcal{P} = \bigtimes_{(s,a) \in (\mathcal{S} \times \mathcal{A})} \mathcal{P}_{sa}, \tag{sa-rectangularity}$$

where $\mathcal{P}_{sa} \subseteq \Delta_{\mathcal{S}}$. In other words, the uncertainty in one state-action pair is independent of that in another state-action pair.

Under this assumption, RMDPs will admit a deterministic optimal policy as in the standard MDPs [17, 25]. The rectangularity assumption also allows the robust value function to be well-defined:

$$v^\pi_{\mathcal{P}} = \min_{P \in \mathcal{P}} v^\pi_P, \quad \text{and} \quad v^*_{\mathcal{P}} = \max_\pi v^\pi_{\mathcal{P}}. \tag{6}$$

In addition, $v^\pi_{\mathcal{P}}$ and $v^*_{\mathcal{P}}$ are the unique fixed points of the robust Bellman operator $T^\pi_{\mathcal{P}}$ and the optimal robust Bellman operator $T^*_{\mathcal{P}}$ respectively, which are defined as

$$T^\pi_{\mathcal{P}} v = \min_{P \in \mathcal{P}} T^\pi_P v \quad \text{and} \quad T^*_{\mathcal{P}} v = \max_\pi T^\pi_{\mathcal{P}} v. \tag{7}$$

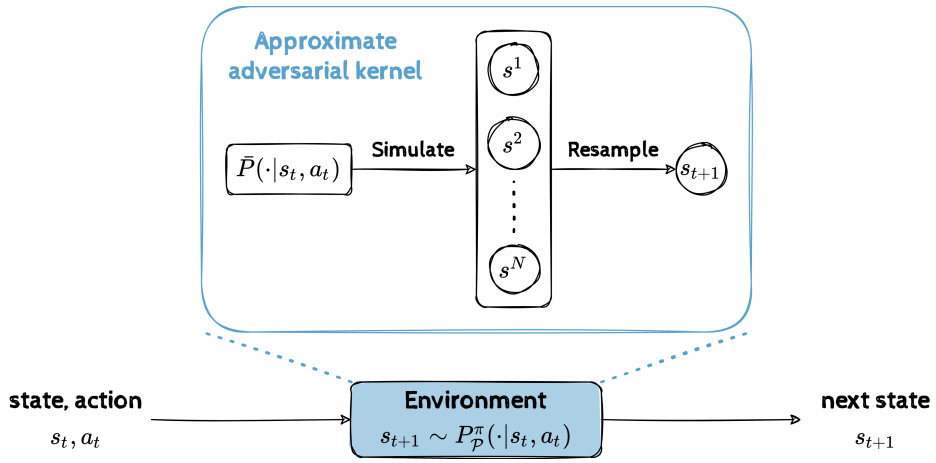

Figure 2: An illustration of how next states are sampled in the approximated adversarial kernel.

To model perturbations on the environment dynamics, the rectangular uncertainty set is often constructed to be centered around a nominal kernel $\bar{P}$, i.e., ,

$$\mathcal{P}_{sa} = \{P_{sa} \mid \text{dist}(P_{sa}, \bar{P}_{sa}) \leq \beta_{sa}\}, \tag{8}$$

where dist is some divergence, β_{sa} is the uncertainty radius, and P_{sa} is a shorthand for $P(\cdot|s, a)$. To measure the deviation from the nominal kernel, L_p norm and KL divergence are often used, i.e., ,

$$\mathcal{P}_{sa} = \{P_{sa} \mid \|P_{sa} - \bar{P}_{sa}\|_p \leq \beta_{sa}\}, \qquad (L_p \text{ uncertainty set})$$

$$\mathcal{P}_{sa} = \{P_{sa} \mid D_{\text{KL}}(P_{sa} \,\|\, \bar{P}_{sa}) \leq \beta_{sa}\}. \qquad (\text{KL uncertainty set})$$

3 Method

As introduced earlier, our work proposes to learn robust policies by approximately simulating the adversarial transition kernel.

Definition 3.1. *For an uncertainty set $\mathcal{P}$, the adversarial kernel for policy π is defined as*

$$P^\pi_\mathcal{P} = \arg\min_{P \in \mathcal{P}} J^\pi_P. \tag{9}$$

Training policies under the adversarial kernel will give us a robust policy with respect to the uncertainty set. The challenge is how to approximate $P^\pi_\mathcal{P}$ in online robust RL. For a general $\mathcal{P}$, it requires an additional minimization process to find the adversarial kernel and it is also unclear how we can parameterize and train $P^\pi_\mathcal{P}$ using gradient-based methods.

Note that all we want from approximating $P^\pi_\mathcal{P}$ is to sample adversarial next states for training, i.e., , simulate the adversarial kernel. Suppose we have access to next states under the nominal kernel $\bar{P}$, if we can draw a connection between $\bar{P}$ and $P^\pi_\mathcal{P}$, then we might be able to obtain next states that are approximately distributed according to $P^\pi_\mathcal{P}(\cdot|s, a)$. Recent work [23] has characterized such a connection for L_p uncertainty set.

Theorem 3.2 (Kumar et al. [23])**.** *For an L_p uncertainty set $\mathcal{P}$, the adversarial kernel is related to the nominal kernel through:*

$$P^\pi_\mathcal{P}(s'|s, a) = \bar{P}(s'|s, a) - \beta_{sa} u^\pi_\mathcal{P}(s'). \tag{10}$$

Here $u^\pi_\mathcal{P}$ is a balanced and normalized version of the robust value function $v^\pi_\mathcal{P}$ (see [23] for details), which has zero mean and unit norm. Intuitively, high $v^\pi_\mathcal{P}(s)$ corresponds to positive $u^\pi_\mathcal{P}(s)$ and low $v^\pi_\mathcal{P}(s)$ corresponds negative $u^\pi_\mathcal{P}(s)$. Thus, the adversarial kernel essentially discourages transitions to states with high values while encouraging transitions to low-value states.

Even with this connection, it is still difficult to sample from the adversarial kernel under L_p uncertainty when the state space is large. First, the adversarial kernel $P^\pi_\mathcal{P}(\cdot|s, a)$ and the nominal kernel $\bar{P}(\cdot|s, a)$

Algorithm 1 Robust RL with Adversarial Kernel Approximation

Input: sample size N, robustness parameter κ

Initialize: initial state s_0, policy π and value function v, data buffer

1: **for** $t = 0, 1, 2, \cdots$ **do**
2: Play action $a_t \sim \pi(\cdot|s_t)$.
3: Simulate next state $s^i \sim \bar{P}(\cdot|s_t, a_t), i = 1, \cdots, N$, with the nominal environment dynamic.
4: Choose $s_{t+1} = s^i$ with probability proportional to $e^{-\frac{v(s^i) - \frac{1}{N}\sum_{i=1}^N v(s^i)}{\kappa}}$.
5: Add (s_t, a_t, s_{t+1}) to the data buffer.
6: Train π and v with data from the buffer using any non-robust RL method.
7: **end for**

Output: a robust policy π

may have different support. In many domains with large state spaces, $\bar{P}(\cdot|s,a)$ may only have support on a small subset of states, whereas $u^\pi_\mathcal{P}$ has support on almost all states since it is normalized from the robust value function $v^\pi_\mathcal{P}$. Thus, even if we can approximate $\bar{P}(s'|s,a)$ with enough samples, it is impossible to evaluate $u^\pi_\mathcal{P}(s')$ on states where $\bar{P}(s'|s,a) = 0$ without the knowing the entire state space. Second, even if we knew $u^\pi_\mathcal{P}$, sampling from $P^\pi_\mathcal{P}(\cdot|s,a)$ requires the ability to "teleport" to states that may not be reachable under the nominal kernel $\bar{P}(\cdot|s,a)$. This is a pretty strong requirement for the environment.

The two difficulties above inherently result from the L_p structure of the uncertainty set. To circumvent those issues, we consider another commonly used uncertainty set, the KL uncertainty set. We characterize its adversarial kernel in Theorem 3.3 and show that it is more amenable to approximation. The proof of Theorem 3.3 is deferred to the appendix.

Theorem 3.3. *For a KL uncertainty set $\mathcal{P}$, the adversarial kernel is related to the nominal kernel through:*

$$P^\pi_\mathcal{P}(s'|s,a) = \bar{P}^\pi(s'|s,a)e^{-\delta^\pi(s')}, \tag{11}$$

where δ^π is of the form

$$\delta^\pi(s') = \frac{v^\pi_\mathcal{P}(s') - \omega_{sa}}{\kappa_{sa}}, \tag{12}$$

and satisfies

$$\sum_{s'} \bar{P}^\pi(s'|s,a)e^{-\delta^\pi(s')} = 1, \quad \sum_{s'} \bar{P}^\pi(s'|s,a)e^{-\delta^\pi(s')}(-\delta^\pi(s')) = \beta_{sa}. \tag{13}$$

Here, ω_{sa} and κ_{sa} are normalizing constants independent of s'. We can view ω_{sa} as a threshold, in the sense that transiting to states with robust values lower than ω_{sa} (i.e., negative $\delta^\pi(s')$) will be encouraged. κ_{sa} works as a temperature parameter to control how much we discourage/encourage transitions to states with high/low robust values.

Unlike the L_p uncertainty case, here, the adversarial kernel $P^\pi_\mathcal{P}(\cdot|s,a)$ has the same support as the nominal kernel $\bar{P}^\pi(\cdot|s,a)$. To sample from $P^\pi_\mathcal{P}(\cdot|s,a)$, we only need to evaluate $e^{-\delta^\pi(s)}$ for states in the support of the nominal kernel. Based on theoretical results, we introduce a method to approximately simulate the adversarial kernel. As illustrated in Figure 2, we first draw a batch of states from the nominal kernel $\bar{P}(\cdot|s,a)$ and then resample the next state with probability proportional to $e^{-\delta^\pi(s')}$. In this way, next states will be approximately distributed according to $P^\pi_\mathcal{P}(\cdot|s,a)$. Compared to the L_p uncertainty case, our method only requires sampling next states from the nominal kernel, which is feasible in most simulated environments.

In practical implementations, we approximate $\delta^\pi(s')$ by

$$\hat{\delta}^\pi(s') = \frac{v(s') - \frac{1}{N}\sum_{i=1}^N v(s')}{\kappa}, \tag{14}$$

where v is the robust value function approximated with neural networks, and κ is a hyperparameter controlling the robustness level. We implement the threshold ω as the average value, and it is possible to use other quantities such as quantiles. Putting it together, we summarize our method in Algorithm 1.

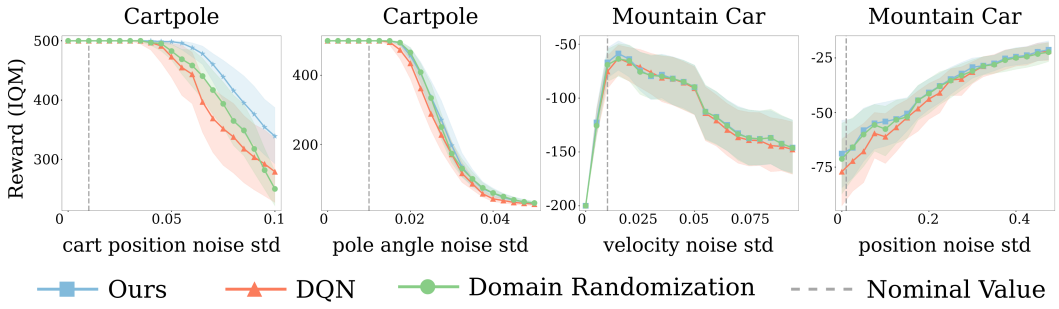

Figure 3: Evaluation results on classic control environments with noise perturbations.

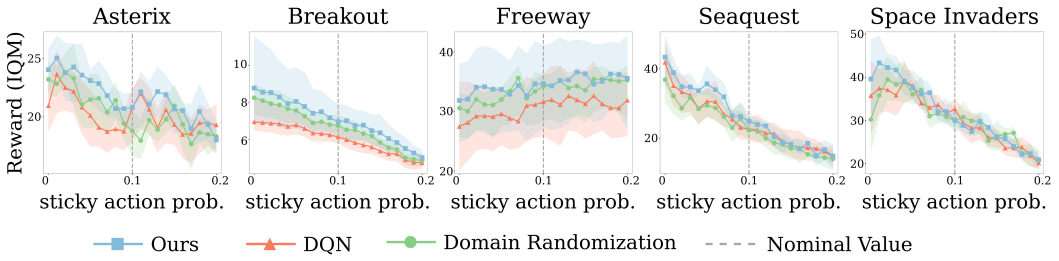

Figure 4: Evaluation results on MinAtar environments with noise perturbations.

4 Experiments

In this section, we first introduce our experimental setting in Section 4.1. Next, we evaluate our method in experiments with perturbed transition dynamics, including noise perturbation (Section 4.2) and environment parameter perturbation (Section 4.3). Finally, we conduct ablation experiments for the hyperparameter τ in Section 4.4.

4.1 Setting

To evaluate the effectiveness of our method in learning robust policies, we conduct experiments in the online robust RL setting: training the agent online under the nominal dynamics and testing its performance under perturbed dynamics. We consider three high-dimensional domains including both discrete and continuous control tasks, to demonstrate our algorithm can be "plugged and played" with any RL method. Specifically, we experiment on 2 classic control environments from OpenAI Gym [5], 5 video games from the MinAtar benchmark [49], and 4 continuous control tasks from DeepMind Control Suite [40]. For the baseline RL algorithm, we use Double DQN [41] for classic control and MinAtar environments, and SAC [12] for continuous control environments. To obtain stable results, we run each experiment with multiple random seeds, and report the interquartile mean (IQM) and 95% stratified bootstrap confidence intervals (CIs) as recommended by [1]. As most existing methods in RMDPs literature do not scale well (Section 5), we do not have "apple-to-apple" comparisons. Hence, we consider another commonly-used robust RL approach: domain randomization [28], and conduct the same set of experiments. Domain randomization [28] trains the agent under diverse scenarios by perturbing the parameter of interest during training, such that the trained agent can be robust to similar perturbations during testing. It is worth noting that domain randomization has an edge on our method, since it has access to different perturbed parameters during training, while our method is completely oblivious to those parameters, and only tests on them. More details about environments, implementations, training and evaluation can be found in the appendix.

4.2 Noise perturbation

In this subsection, we evaluate our method in scenarios where the perturbations on the transition dynamic are implemented as noise perturbations. Specifically, we consider stochastic nominal kernels

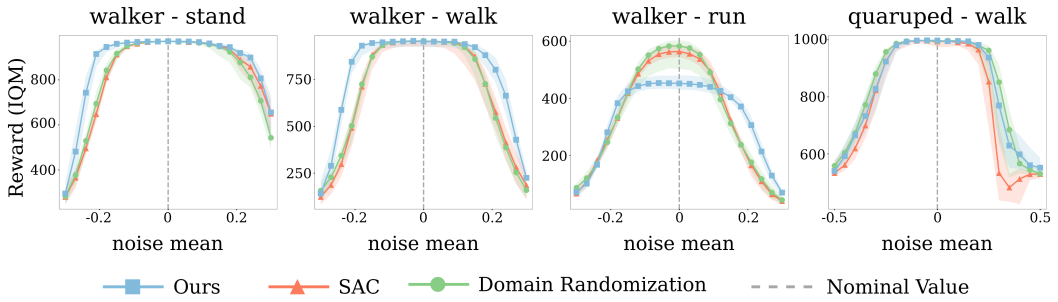

Figure 5: Evaluation results on DeepMind Control environments with noise perturbations.

in which the stochasticity is controlled by some (observation or action) noises. The agent is trained under a fixed noise (i.e., , the nominal kernel) and tested with varying noises (i.e., , perturbed kernels).

On classic control environments, we implement the stochasticity by adding Gaussian noise to the state after applying the original deterministic dynamics of the environments, i.e., , $\tilde{s}_{t+1} = s_{t+1} + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma)$. Then $\tilde{s}_{t+1}$ is considered as the next state output from the stochastic nominal kernel. The noise scale σ is fixed during training and varied during testing. The agent's test performance across different perturbed values is depicted in Figure 3. In the Cartpole environment, when the noise scale deviates from the nominal value, our method achieves better performance than the baseline DQN. The improvement in robustness is particularly significant when we change the noise scale in the cart position. While the improvement in the Mountain Car environment is less significant, our method still performs favorably against the baseline.

Apart from the classic control tasks, we evaluate the performance of our method in discrete control on the more challenging MinAtar environments. Here we take advantage of the existing sticky action and use it as the source of stochasticity. The sticky action probability is fixed at some value during training and perturbed during testing. As the results in Figure 4 show, our method yields better performance than the baseline and domain randomization. Note that the agent's performance sometimes does not follow a decreasing trend when the perturbation parameter moves away from the nominal value in one direction. This might be because the perturbation parameter has an asymmetric effect on the learning.

Next, we evaluate our method on the continuous control tasks in the DeepMind Control Suite. The stochasticity is implemented by adding Gaussian noise to the action since directly adding noise to the state might lead to an invalid physical state. During testing, we perturb the mean of the Gaussian noise. Figure 5 shows the agent's performance across different perturbed values. From the results, we can see that our method suffers less performance degradation as the noise mean deviates from zero (the nominal value), clearly outperforming the baseline SAC and Domain Randomization. In the walker-run task, our method achieves lower reward under the nominal dynamic but performs better under perturbed ones, which indicates a trade-off between the performance under the nominal kernel and robustness under perturbations.

4.3 Perturbing environment parameters

To further validate the effectiveness of our method, we consider a more realistic scenario where some physical/logical parameters in the environment (e.g., , pole length in Cartpole) are perturbed. Similarly, the agent is trained with a fixed environment parameter, and tested under perturbed parameters.

For classic control environments, we perturb cart mass, pole mass, pole length, and gravity in Cartpole, and perturb force, gravity, and max speed in Mountain Car. Figure 6 summarizes the testing results of the agents trained under the nominal dynamics. Again, we can see that our method achieves better performance than the baseline when the environment parameters deviate from the nominal value, especially in the Cartpole environment.

For DeepMind control tasks, we implement the perturbations on the environment parameters using the Real-World Reinforcement Learning Suite [8]. Specifically, we perturb joint damping, thigh length, and torso length in walker tasks, and perturb joint damping, shin length, and torso density

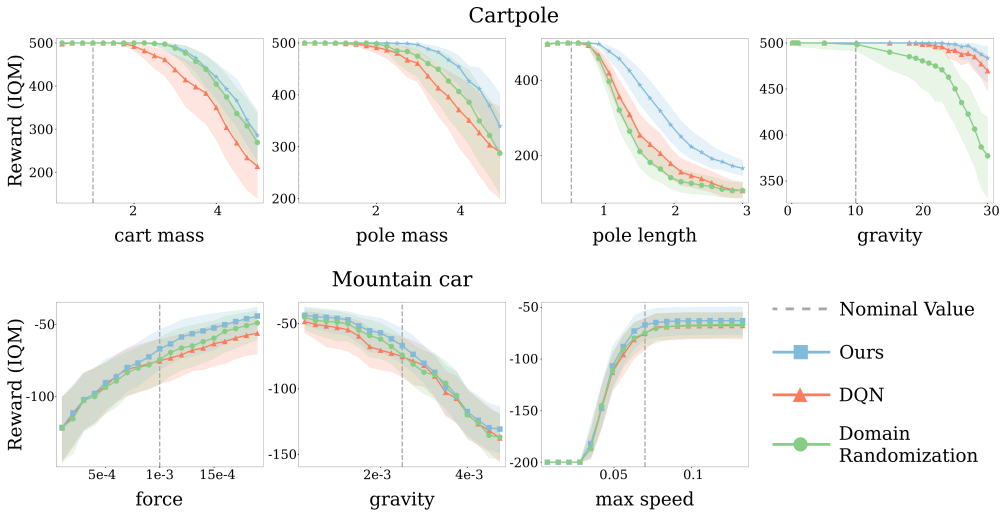

Figure 6: Evaluation results on classic control environments with perturbed environment parameters.

for quadruped tasks. As shown in Figure 7, our method generally works better than the baseline under model mismatch, improving the robustness of the learned policy. Similar to our observations in the previous section, the walker-run task emphasizes the inherent trade-off of solving RMDPs: optimizing the worst-case scenario can lead to suboptimal performance under the nominal model. While the performance improvement is less obvious in the quadruped-walk task, the results of our method have lower variance than the baseline.

4.4 Ablation studies

Two important hyperparameter in our algorithm are κ, which controls the skewness of the distribution for re-sampling, and N which controls the number of samples for each time-step. In this subsection, we conduct ablation experiments to investigate how those parameters affects performance. Intuitively, when we decrease κ, we are essentially considering a higher level of robustness. If κ is very small, then with a high probability the environment dynamic will transit to the "worst" state (i.e., , one with the lowest value). In addition, by increasing N our approximation of the the nominal kernel is better, which might also create a better approximation of the "adversarial" one.

We experiment on the DeepMind Control tasks under noise perturbation setting, using different κ and N, when we train the agent. For clarity, we plot the performance difference between our method and the baseline instead of the absolute performance and defer the original results with CIs (shaded areas) to the appendix. Figure 8 shows the results of changing κ, In the walker domain, decreasing κ makes our algorithm perform better in perturbed environments, which aligns with our expectations.

It is worth mentioning that the influence of κ has a dependency on the environment. Decreasing it too much can lead to too conservative policies and may not always work well. For example, on the quadruped domain, using a small κ does not yield the best performance. In addition, we observe the robustness-performance trade-off in the walker-run task once again. While large κ achieves high performance under the nominal kernel, it significantly underperforms when the kernel is perturbed.

Figure 9 shows the results of different number of samples N, in principle, increasing the number of next state samples can help us better approximate the worst kernel, which might lead to better performance. From the empirical results, we can see that small sample size will result in limited performance gain compared to the baseline, but increasing the sample size may not bring monotonic improvements.

5 Related works

Early works in RMDPs lay the theoretical foundations for solving RMDPs with robust dynamic programming [2, 17, 19, 25, 46]. Recent works attempt to reduce the time complexity for certain

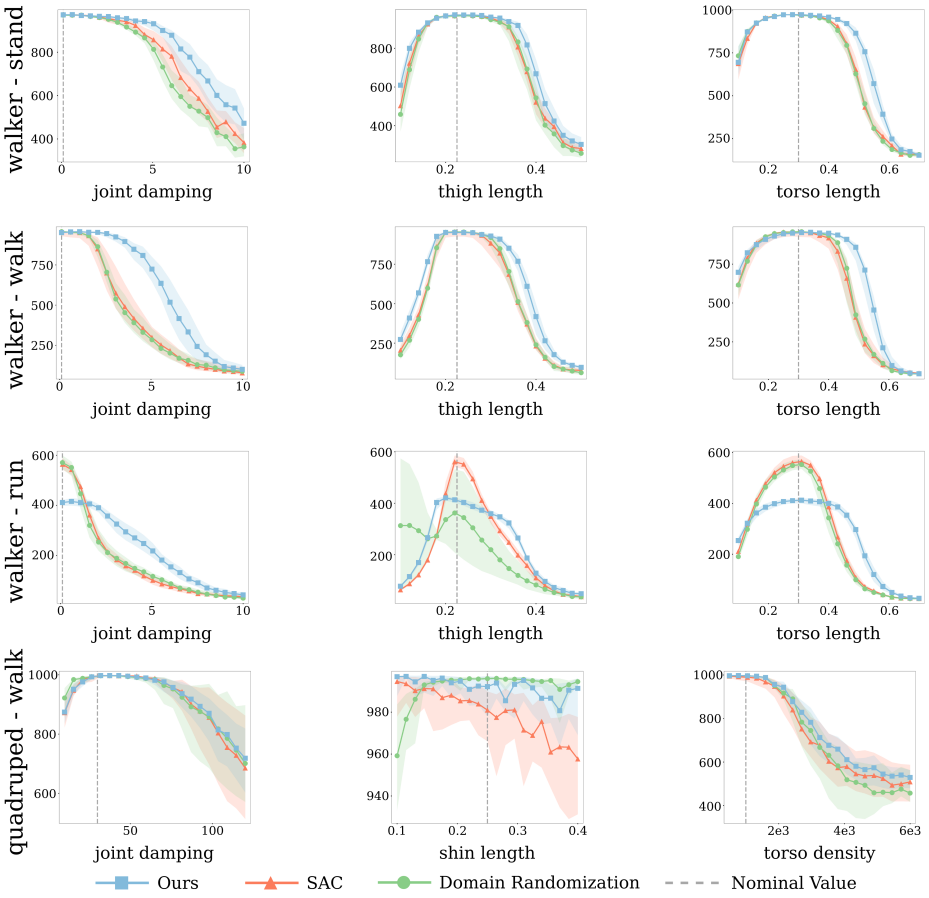

Figure 7: Evaluation results on DeepMind Control tasks with perturbed environment parameters.

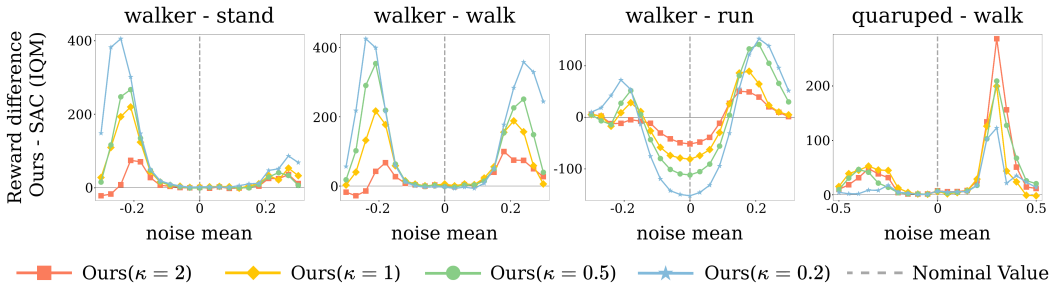

Figure 8: Evaluation results on DeepMind Control tasks with noise perturbations for different κ.

s-rectangular uncertainties, such as L_1 uncertainty [14, 16] and KL uncertainty [10]. However, all those methods require the full knowledge of the nominal model.

One line of work aims to bridge the gap between theory and practice and design methods that can be applied in the online robust RL setting where we do not have full knowledge about the transition model. Derman et al. [7] point out the connection between regularized MDPs and Robust MDPs. They define new regularized robust Bellman operators that suggest a possible online sample-based method. However, the contraction of the Bellman operators implicitly assumes that the state space can not be very large. On regularizing the learning process, Kumar et al. [21, 23] introduce Q-learning and policy gradient methods for L_p uncertainty sets, but do not experimentally evaluate their methods with experiments. Another type of uncertainty set considered in online robust RL is the R-contamination uncertainty, for which previous works have derived a robust Q-learning algorithm [42]

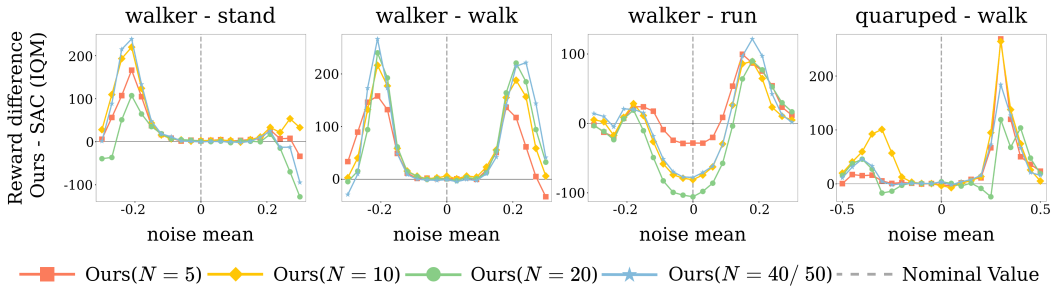

Figure 9: Evaluation results on DeepMind Control tasks with noise perturbations for different N.

and a regularized policy gradient algorithms [43]. R-contamination uncertainty assumes that the adversary can take the agent to any state, which is too conservative in practice. In addition, all of those methods are tied to a particular type of RL algorithm. Our work, however, aims to tackle the problem from a different perspective by approximating the adversarial kernel and can adopt any non-robust RL algorithm to learn the optimal robust policy.

Perturbing the training environment was previously discussed in unsupervised environment design [6, 18], domain randomization [28, 38], robust adversarial RL [29, 34] and risk aversion [11, 26]. However, their focus on robustness differs from our perspective. They primarily investigate how to design a practical method to deal with changes in the environment. Our method is under the RMDPs framework and based on the theoretical derivation of the worst kernel within the KL uncertainty set.

Most closely related to our work is [23], which characterizes the adversarial kernel for L_p uncertainty set. While they use the adversarial kernel to motivate some regularization during training, we propose to approximately simulate the adversarial kernel, opening a new paradigm for learning robust policies in RMDPs. In addition, we extend their characterization of the adversarial kernel to the KL uncertainty set, giving rise to a practical method scalable to high-dimensional domains.

6 Conclusions and discussions

In this paper, we introduce an approach that tackles the RMDPs problem from a new perspective, by approximately simulating the adversarial transition kernel while leaving the RL part untouched. The highlight of our method is that it can be applied on top of existing non-robust deep RL algorithms to learn robust policies, exhibiting attractive scalability to high-dimensional domains. We believe this new perspective will offer some insights for future works on RMDPs.

One limitation of our work is that we require the ability to sample next states from the transition model for multiple times. In future works, we will study how to combine our method with a learned transition model where sampling the next states would not be a problem. Another interesting direction is to explore the application of this method in an offline setting where we have the logged data instead of the nominal environment.

References

[1] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 2021.

[2] J. Andrew Bagnell, Andrew Y. Ng, and Jeff G. Schneider. Solving Uncertain Markov Decision Processes. *Technical Report*, 1 2001. doi: 10.1184/R1/6560927.v1. URL `https://kilthub.cmu.edu/articles/journal_contribution/Solving_Uncertain_Markov_Decision_Processes/6560927`.

[3] Bahram Behzadian, Marek Petrik, and Chin Pang Ho. Fast algorithms for l_∞-constrained s-rectangular robust MDPs. *Advances in Neural Information Processing Systems*, 34:25982–25992, 2021.

[4] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *ArXiv preprint*, abs/1606.01540, 2016. URL https://arxiv.org/abs/1606.01540.

[6] Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. *Advances in neural information processing systems*, 33:13049–13061, 2020.

[7] Esther Derman, Matthieu Geist, and Shie Mannor. Twice regularized mdps and the equivalence between robustness and regularization. *Advances in Neural Information Processing Systems*, 34:22274–22287, 2021.

[8] Gabriel Dulac-Arnold, Nir Levine, Daniel J. Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. An empirical investigation of the challenges of real-world reinforcement learning. *CoRR*, abs/2003.11881, 2020. URL https://arxiv.org/abs/2003.11881.

[9] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[10] Julien Grand-Clément and Christian Kroer. Scalable first-order methods for robust mdps. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 12086–12094. AAAI Press, 2021. URL https://ojs.aaai.org/index.php/AAAI/article/view/17435.

[11] Ido Greenberg, Yinlam Chow, Mohammad Ghavamzadeh, and Shie Mannor. Efficient risk-averse reinforcement learning. *arXiv preprint arXiv:2205.05138*, 2022.

[12] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1856–1865. PMLR, 2018. URL http://proceedings.mlr.press/v80/haarnoja18b.html.

[13] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1856–1865. PMLR, 2018. URL http://proceedings.mlr.press/v80/haarnoja18b.html.

[14] Chin Pang Ho, Marek Petrik, and Wolfram Wiesemann. Fast bellman updates for robust mdps. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1984–1993. PMLR, 2018. URL http://proceedings.mlr.press/v80/ho18a.html.

[15] Chin Pang Ho, Marek Petrik, and Wolfram Wiesemann. Partial policy iteration for l1-robust markov decision processes. *ArXiv preprint*, abs/2006.09484, 2020. URL https://arxiv.org/abs/2006.09484.

[16] Chin Pang Ho, Marek Petrik, and Wolfram Wiesemann. Partial policy iteration for l1-robust markov decision processes. *The Journal of Machine Learning Research*, 22(1):12612–12657, 2021.

[17] Garud N Iyengar. Robust dynamic programming. *Mathematics of Operations Research*, 30(2): 257–280, 2005.

[18] Minqi Jiang, Michael Dennis, Jack Parker-Holder, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Replay-guided adversarial environment design. *Advances in Neural Information Processing Systems*, 34:1884–1897, 2021.

[19] David L. Kaufman and Andrew J. Schaefer. Robust modified policy iteration. *INFORMS J. Comput.*, 25:396–410, 2013.

[20] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

[21] Navdeep Kumar, Kfir Levy, Kaixin Wang, and Shie Mannor. Efficient policy iteration for robust markov decision processes via regularization. *ArXiv preprint*, abs/2205.14327, 2022. URL https://arxiv.org/abs/2205.14327.

[22] Navdeep Kumar, Kfir Levy, Kaixin Wang, and Shie Mannor. Efficient policy iteration for robust Markov decision processes via regularization. *ArXiv preprint*, abs/2205.14327, 2022. URL https://arxiv.org/abs/2205.14327.

[23] Navdeep Kumar, Esther Derman, Matthieu Geist, Kfir Levy, and Shie Mannor. Policy gradient for s-rectangular robust markov decision processes. *ArXiv preprint*, abs/2301.13589, 2023. URL https://arxiv.org/abs/2301.13589.

[24] Shie Mannor, Duncan Simester, Peng Sun, and John N Tsitsiklis. Bias and variance approximation in value function estimates. *Management Science*, 53(2):308–322, 2007.

[25] Arnab Nilim and Laurent El Ghaoui. Robust control of Markov decision processes with uncertain transition matrices. *Operations Research*, 53(5):780–798, 2005.

[26] Xinlei Pan, Daniel Seita, Yang Gao, and John Canny. Risk averse robust adversarial reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8522–8528. IEEE, 2019.

[27] Kishan Panaganti and Dileep Kalathil. Sample complexity of robust reinforcement learning with a generative model. In *International Conference on Artificial Intelligence and Statistics*, pages 9582–9602. PMLR, 2022.

[28] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE, 2018.

[29] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 2817–2826. PMLR, 2017. URL http://proceedings.mlr.press/v70/pinto17a.html.

[30] Martin L. Puterman. Markov decision processes: Discrete stochastic dynamic programming. In *Wiley Series in Probability and Statistics*, 1994.

[31] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[32] Antonin Raffin. Rl baselines3 zoo. https://github.com/DLR-RM/rl-baselines3-zoo, 2020.

[33] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

[34] Marc Rigter, Bruno Lacerda, and Nick Hawes. Rambo-rl: Robust adversarial model-based offline reinforcement learning. *arXiv preprint arXiv:2204.12581*, 2022.

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

[36] Laixi Shi and Yuejie Chi. Distributionally robust model-based offline reinforcement learning with near-optimal sample complexity. *arXiv preprint arXiv:2208.05767*, 2022.

[37] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL `http://incompleteideas.net/book/the-book-2nd.html`.

[38] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[39] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

[40] Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020. ISSN 2665-9638. doi: https://doi.org/10.1016/j.simpa.2020.100022. URL `https://www.sciencedirect.com/science/article/pii/S2665963820300099`.

[41] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 2094–2100. AAAI Press, 2016. URL `http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12389`.

[42] Yue Wang and Shaofeng Zou. Online robust reinforcement learning with model uncertainty. *ArXiv preprint*, abs/2109.14523, 2021. URL `https://arxiv.org/abs/2109.14523`.

[43] Yue Wang and Shaofeng Zou. Policy gradient method for robust reinforcement learning, 2022.

[44] Yue Wang and Shaofeng Zou. Policy gradient method for robust reinforcement learning. *International Conference on Machine Learning*, 162:23484–23526, 2022.

[45] Yue Wang, Fei Miao, and Shaofeng Zou. Robust constrained reinforcement learning. *ArXiv preprint*, abs/2209.06866, 2022. URL `https://arxiv.org/abs/2209.06866`.

[46] Wolfram Wiesemann, Daniel Kuhn, and Berç Rustem. Robust markov decision processes. *Mathematics of Operations Research*, 38(1):153–183, 2013.

[47] Zaiyan Xu, Kishan Panaganti, and Dileep Kalathil. Improved sample complexity bounds for distributionally robust reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, pages 9728–9754. PMLR, 2023.

[48] Denis Yarats and Ilya Kostrikov. Soft actor-critic (sac) implementation in pytorch. `https://github.com/denisyarats/pytorch_sac`, 2020.

[49] Kenny Young and Tian Tian. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. *arXiv preprint arXiv:1903.03176*, 2019.

A Proof of Theorem 3.3

Recall, the worst values are defined as

$$P^\pi_\mathcal{P} \in \arg\min_{P\in\mathcal{P}} J^\pi_P$$

for any general uncertainty set $\mathcal{P}$. Further, for sa-rectangular uncertainty set $\mathcal{P} = \times_{s\in\mathcal{S},a\in\mathcal{A}}\mathcal{P}_{sa}$, the robust value function exists, that is, the following is well defined [17, 25]

$$v^\pi_\mathcal{P} = \min_{P\in\mathcal{P}} v^\pi_P.$$

This implies,

$$v^\pi_\mathcal{P} = \Big(I - \gamma(P^\pi_\mathcal{P})^\pi \Big)^{-1} R^\pi$$

is the fixed point of robust Bellman operator $\mathcal{T}^\pi_\mathcal{P}$[17, 25], defined as

$$\mathcal{T}^\pi_\mathcal{P} v := \min_{P\in\mathcal{P}} \mathcal{T}^\pi_P v.$$

Proposition A.1. *The worst values can be computed from robust value function. That is*

$$\arg\min_{P\in\mathcal{P}} \mathcal{T}^\pi_P v^\pi_\mathcal{P} \subseteq \min_{P\in\mathcal{P}} v^\pi_P \subseteq \arg\min_{P\in\mathcal{P}} J^\pi_P.$$

Proof. Let

$$P^* \in \arg\min_{P\in\mathcal{P}} \mathcal{T}^\pi_P v^\pi_\mathcal{P}.$$

Now, from fixed point of robust Bellman operator, we have

$$\begin{aligned} v^\pi_\mathcal{P} =& \mathcal{T}^\pi_\mathcal{P} v^\pi_\mathcal{P} = \min_{P\in\mathcal{P}} \mathcal{T}^\pi_P v^\pi_\mathcal{P}, \\ &= \mathcal{T}^\pi_{P^*} v^\pi_\mathcal{P}, \qquad \text{(by construction)}, \\ &= R^\pi + \gamma(P^*)^\pi v^\pi_\mathcal{P}, \qquad \text{(by definition)}. \end{aligned}$$

The above implies,

$$v^\pi_\mathcal{P} = \Big(I - \gamma(P^*)^\pi \Big)^{-1} R^\pi.$$

This implies,

$$P^* \in arg\min_{P\in\mathcal{P}} v^\pi_P.$$

The last inclusion is trivial, that is, every minimizer of value function is a mininizer of robust return. □

Theorem 3.3. *For a KL uncertainty set $\mathcal{P}$, the adversarial kernel is related to the nominal kernel through:*

$$P^\pi_\mathcal{P}(s'|s,a) = \bar{P}^\pi(s'|s,a)e^{-\delta^\pi(s')}, \tag{11}$$

where δ^π is of the form

$$\delta^\pi(s') = \frac{v^\pi_\mathcal{P}(s') - \omega_{sa}}{\kappa_{sa}}, \tag{12}$$

and satisfies

$$\sum_{s'} \bar{P}^\pi(s'|s,a)e^{-\delta^\pi(s')} = 1, \quad \sum_{s'} \bar{P}^\pi(s'|s,a)e^{-\delta^\pi(s')}(-\delta^\pi(s')) = \beta_{sa}. \tag{13}$$

Proof. Recall Definition 3.1

$$P^\pi_\mathcal{P} \in \arg\min_{P\in\mathcal{P}} J^\pi_P. \tag{15}$$

From Proposition A.1, for sa-rectangular uncertainty set $\mathcal{P}$, a worst kernel can be computed using robust value function as

$$P^\pi_\mathcal{P} \in arg\min_{P\in\mathcal{P}} \mathcal{T}^\pi_P v^\pi_\mathcal{P}.$$

Recall, our KL-constrained uncertainty $\mathcal{P}$ is defined as

$$\mathcal{P} := \{P \mid P \in (\Delta_{\mathcal{S}})^{\mathcal{S} \times \mathcal{A}}, D_{KL}(\bar{P}_{s,a}, P_{sa}) \leq \beta_{sa}, \forall s, a\}.$$

where D_{KL} is KL norm that is defined as

$$D_{KL}(P, Q) = \sum_s P(s) \log \left(\frac{P(s)}{Q(s)} \right).$$

Using Proposition A.1 and definition of uncertainty set $\mathcal{P}$, the worst kernel can be extracted as

$$P^{\pi}_{\mathcal{P}}(\cdot|s, a) \in \underset{D_{KL}(p, P_0(\cdot|s,a)) \leq \beta_{sa}, \sum_s p(s)=1, p \succeq 0}{\arg\min} \langle p, v^{\pi}_{\mathcal{U}} \rangle.$$

Using the Lemma A.2, we get the desired solution. □

Lemma A.2. *For $q \in \Delta_{\mathcal{S}}, v \in \mathbb{R}^{\mathcal{S}}, \beta \geq 0$, a solution to*

$$\min_{p \ln(\frac{p}{q}) \leq \beta, 1^T p = 1, p \succeq 0} \langle p, v \rangle.$$

is given by

$$p = q e^{-\frac{v-\omega}{\lambda}},$$

where

$$p \log(\frac{p}{q}) = \left\langle q e^{-\frac{v-\omega}{\lambda}}, \frac{v-\omega}{\lambda} \right\rangle = -\beta$$

and

$$\sum_s q(s) e^{-\frac{v(s)-\omega}{\lambda}}$$

Proof. We have the following optimization problem,

$$\min_{p \ln(\frac{p}{q}) \leq \beta, 1^T p = 1, p \succeq 0} \langle p, v \rangle. \tag{16}$$

We ignore the constraint $p \succeq 0$ for the moment (as we see later, this constrained is automatically satisfied), and focus on

$$\min_{p \ln(\frac{p}{q}) \leq \beta, 1^T p = 1} \langle p, v \rangle. \tag{17}$$

We define Lagrange multiplier as

$$L(p, \lambda, \mu) = \langle p, v \rangle + \lambda \left(p \ln(\frac{p}{q}) - \beta \right) + \mu \left(1^T p - 1 \right).$$

We now put the stationarity condition:

$$\begin{aligned} \frac{\partial L}{\partial p} &= v + \lambda \left(\ln(\frac{p}{q}) + 1 \right) + \mu 1 = 0 \\ \implies p &= q e^{-1} e^{-\frac{v+\mu}{\lambda}}. \end{aligned}$$

With appropriate change of variable $\mu \to \omega$, we have

$$p = q e^{-\frac{v-\omega}{\lambda}}.$$

We have to find the constants ω and λ, using the constraints

$$p \log(\frac{p}{q}) = \left\langle q e^{-\frac{v-\omega}{\lambda}}, \frac{v-\omega}{\lambda} \right\rangle = -\beta$$

and

$$\sum_s p(s) = \sum_s q(s) e^{-\frac{v(s)-\omega}{\lambda}} = 1.$$

We further note that the constraint $1 \geq p(s) \geq 0$ is automatically satisfied as

$$p(s) = q(s) e^{-\frac{v(s)-\omega}{\lambda}} \geq 0$$

and $\sum_s p(s) = 1$, ensures $p(s) \leq 1 \quad \forall s$. □

B Experiment details

B.1 Environments

B.1.1 Classic control tasks

Cartpole [4] is one of the classic control tasks in OpenAI Gym [5]. The task is to balance a pendulum on a moving cart, by moving the cart either left or right. The state consists of the location and velocity of the cart, as well as the angle and angular velocity of the pendulum. To make the transition dynamic stochastic, we add Gaussian noises to the cart position or the pole angle.

Another classic control task we use is Mountain Car [5]. The task is to accelerate a car at the bottom of a valley to reach the goal state on top of the right hill. The state consists of the location and velocity of the car. Similarly, we add Gaussian noises to the cart position or the cart velocity to make stochastic transition dynamic.

The detailed configurations for the nominal value and the perturbation range are summarized in Table 1 and Table 2.

Table 1: Perturbation configurations for Cartpole environment.

	PARAMETER	NOMINAL VALUE	PERTURBATION RANGE
NOISE PERTUBRATION	Cart position noise (std)	0.01	$[0, 0.1]$
	Pole angle noise (std)	0.01	$[0, 0.05]$
ENV. PARAM. PERTUBRATION	Pole mass	0.1	$[0.15, 3.0]$
	Pole length	0.5	$[0.25, 5.0]$
	Cart mass	1	$[0.25, 5.0]$
	Gravity	9.8	$[0.1, 30]$

Table 2: Perturbation configurations for Mountain Car environment.

	PARAMETER	NOMINAL VALUE	PERTURBATION RANGE
NOISE PERTUBRATION	Position noise (std)	0.01	$[0, 0.5]$
	Velocity noise (std)	0.01	$[0, 0.1]$
ENV. PARAM. PERTUBRATION	max speed	0.07	$[0.007, 0.14]$
	force	0.001	$[0.0001, 0.002]$
	gravity	0.0025	$[0.00025, 0.005]$

B.1.2 MinAtar

The MinAtar benchmark[6] is a simplification of the widely-used Atari benchmark [4], which eliminates the representation complexity while preserving the game mechanism. MinAtar consists of 5 games: `Asterix`, `Breakout`, `Freeway`, `Seaquest` and `SpaceInvaders`. The observation is a $10 \times 10 \times n$ grid image, where each channel corresponds to a game-specific object. We use the minimal action space for each game. As mentioned in the main text, the stochasticity of the transition dynamic comes from *sticky actions*. That is, at each step, the agent would repeat previous action with some probability instead of executing the chosen action. The nominal value for sticky action probability is 0.1 and the perturbation range is $[0.0, 0.2]$.

B.1.3 DeepMind Control Suite

The DeepMind Control Suite [40] is a set of continuous control tasks powered by the MuJoCo physics engine [39]. It is widely used to benchmark reinforcement learning agents. As mentioned

[4] `https://gymnasium.farama.org/environments/classic_control/cart_pole/`

[5] `https://gymnasium.farama.org/environments/classic_control/mountain_car/`

[6] `https://github.com/kenjyoung/MinAtar`

in the main text, we consider 4 tasks in our paper: `walker-stand`, `walker-walk`, `walker-run`, `quadruped-walk`. For `walker` tasks, the observations are 24-dimensional vectors and the actions are 6-dimensional vectors. For `quadruped`, the observations are 78-dimensional vectors and the actions are 12-dimensional vectors. For noise perturbation, we fix the standard deviation of the Gaussian noise to 0.2 for `walker` and 0.1 for `quadruped`. The nominal value and the perturbation range are summarized in Table 3 and Table 4.

Table 3: Perturbation configurations for walker tasks.

	PARAMETER	NOMINAL VALUE	PERTURBATION RANGE
NOISE PERTUBRATION	action noise (mean)	0.0	$[-0.3, 0.3]$
ENV. PARAM. PERTUBRATION	thigh length	0.225	$[0.1, 0.5]$
	torso length	0.3	$[0.1, 0.7]$
	joint damping	0.1	$[0.1, 10]$
	contact friction	0.7	$[0.01, 0.7]$

Table 4: Perturbation configurations for quadruped tasks.

	PARAMETER	NOMINAL VALUE	PERTURBATION RANGE
NOISE PERTUBRATION	action noise (mean)	0.0	$[-0.5, 0.5]$
ENV. PARAM. PERTUBRATION	shin length	0.25	$[0.1, 0.4]$
	torso density	1000	$[500, 6000]$
	joint damping	30	$[10, 120]$
	contact friction	1.5	$[0.1, 2.5]$

B.2 Training and evaluation

For both our method and the baseline, we first train the agent under the nominal environment, then for each perturbed environment during testing, we calculate the average reward from 30 episodes. We repeat this process with 40 random seeds in the classic control environments and 10 seeds in MinAtar and DeepMind Control environments. Following the recommended practice in [1], we report the Interquartile Mean (IQM) and the 95% stratified bootstrap confidence intervals (CIs), using The IQM metric is measured by discarding the top and bottom 25% of the results, and averaging across the remaining middle 50%. IQM has the benefit of being more robust to outliers than a regular mean, and being a better estimator of the overall performance than the median. We use the rliable library[7] to calculate IQM and CIs.

As mentioned earlier, we use double DQN [41] as the vanilla non-robust RL algorithm for environments with discrete action spaces. Specifically, we follow the implementation in Stable-Baselines3 [33]. For the classic control environments, we use the DQN's hyperparameters suggested in RL Baselines3 Zoo [32], and for the MinAtar environments, we use the hyperparameters described by [49]. For the classic control environments, we use a two-layer MLP neural network with 256 hidden units per layer. For the MinAtar environments, we use a CNN consisting of a single convolutional layer (16 output channels, 3×3 kernel, stride$= 1$, and padding$= 0$) and another fully connected layer with 128 hidden units. The detailed configurations are summarized in Table 5.

For environments with continuous action spaces, we choose the SAC algorithm [13] as the vanilla non-robust RL algorithm, and follow the implementations and hyperparameter choices in [48]. Both the actor and critic use a two-layer MLP neural network with 1024 hidden units per layer. Table 6 lists the hyperparameters.

The configurations for the sample size N and the robustness parameter κ used in our experiments are summarized in Table 7.

[7] `https://github.com/google-research/rliable`

Table 5: Hyperparameters for double DQN used in the experiments.

PARAMETER	CARTPOLE	MOUNTAIN CAR	MINATAR
batch size	64	128	32
buffer size	100000	10000	100000
exploration final epsilon	0.04	0.07	0.01
exploration fraction	0.16	0.2	0.1
gamma	0.99	0.98	0.99
gradient steps	128	8	1
learning rate	0.0023	0.004	0.00025
learning starts	1000	1000	5000
target update interval	10	600	1000
train frequency	256	16	4
total time-steps	50000	120000	5000000

Table 6: Hyperparameters for SAC used in the experiments.

PARAMETER	VALUE
Total steps	1e6
Warmup steps	5000
Replay size	1e6
Batch size	1024
Discount factor γ	0.99
Optimizer	Adam [20]
Learning rate	1e-4
Target smoothing coefficient	0.005
Target update interval	2
Initial temperature	0.1
Learnable temperature	Yes

Table 7: Hyperparameters specific to our method used in the experiments.

	ENVIRONMENT	SAMPLE SIZE N	ROBUSTNESS PARAMETER κ
CLASSIC CONTROL	Cartpole	15	1
	Mountain car	4	5
MINATAR	Asterix	5	5
	Breakout	50	5
	Freeway	5	1
	Seaquest	10	10
	SpaceInvaders	5	0.5
DEEPMIND CONTROL SUITE	walker-stand	10	0.2
	walker-walk	10	0.2
	walker-run	10	0.2
	quadruped-walk	40	0.5

B.3 Computational resources and costs

We used the following resources in our experiments:

- **CPU:** AMD EPYC 7742 64-Core Processor
- **GPU:** NVIDIA GeForce RTX 2080 Ti

Table 8 lists the training time.

Table 8: Training time per run of our experiments on a single GPU.

	Environment	Baseline	Ours
Classic Control	Cartpole	$\sim$ 4 minutes	$\sim$ 5 minutes
	Mountain car	$\sim$ 9 minutes	$\sim$ 10 minutes
MinAtar	Asterix	$\sim$ 4 hours	$\sim$ 5 hours
	Breakout	$\sim$ 3 hours	$\sim$ 4 hours
	Freeway	$\sim$ 9.5 hours	$\sim$ 12 hours
	Seaquest	$\sim$ 8 hours	$\sim$ 4.5 hours
	SpaceInvaders	$\sim$ 4 hours	$\sim$ 6 hours
DeepMind Control Suite	all tasks	$\sim$ 5 hours	$\sim$ 6 hours

C Additional results

In order to further show how our method can be viewed as a meta-algorithm. We conducted the same experiments with two more algorithms, PPO [35] and TD3 [9]. In Figure 10 we can see the results of noise perturbation on classic control environments, and in Figure 11 we can see the results on noise perturbation on deepmind control suite tasks. Both show the same trends as explained in Section 4.2.

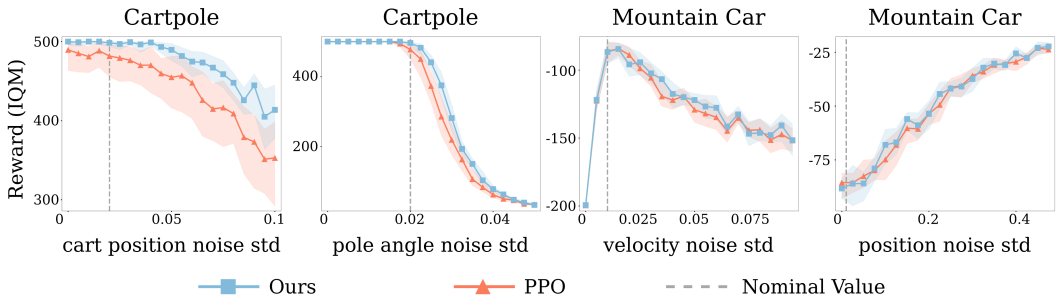

Figure 10: Evaluation results on classic control environments with noise perturbations.

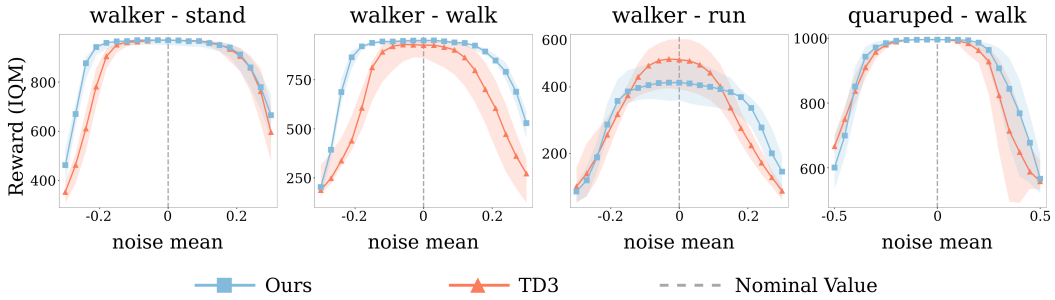

Figure 11: Evaluation results on DeepMind Control environments with noise perturbations.

In Figure 12 we can see the results of environment parameters perturbation on classic control environments, and in Figure 13 we can see the results on environment parameters perturbation on deepmind control suite tasks. Both show the same trends as explained in Section 4.3.

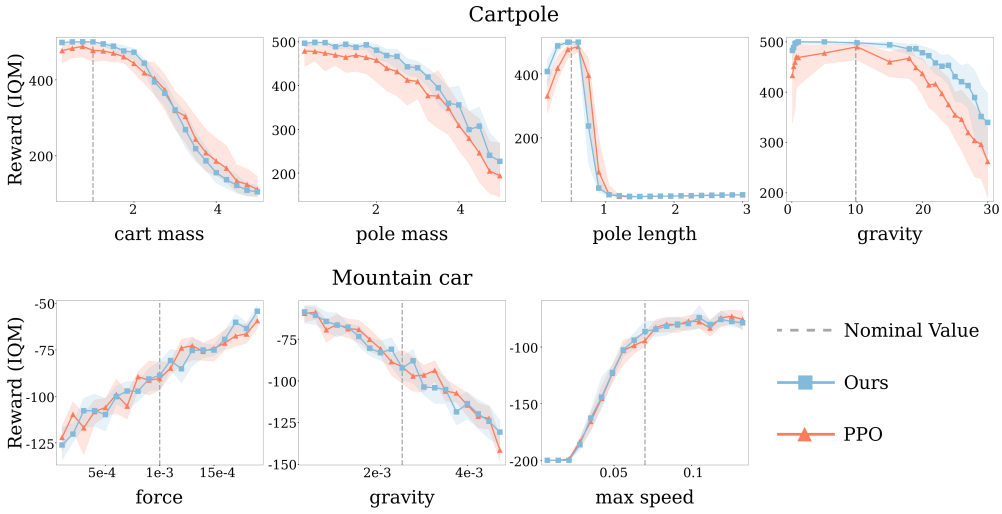

Figure 12: Evaluation results on classic control environments with perturbed environment parameters.

In Section 4.4, we show the relative performance for the ablation study on parameter κ. Here we include the absolute results in Figures 14 and 15.

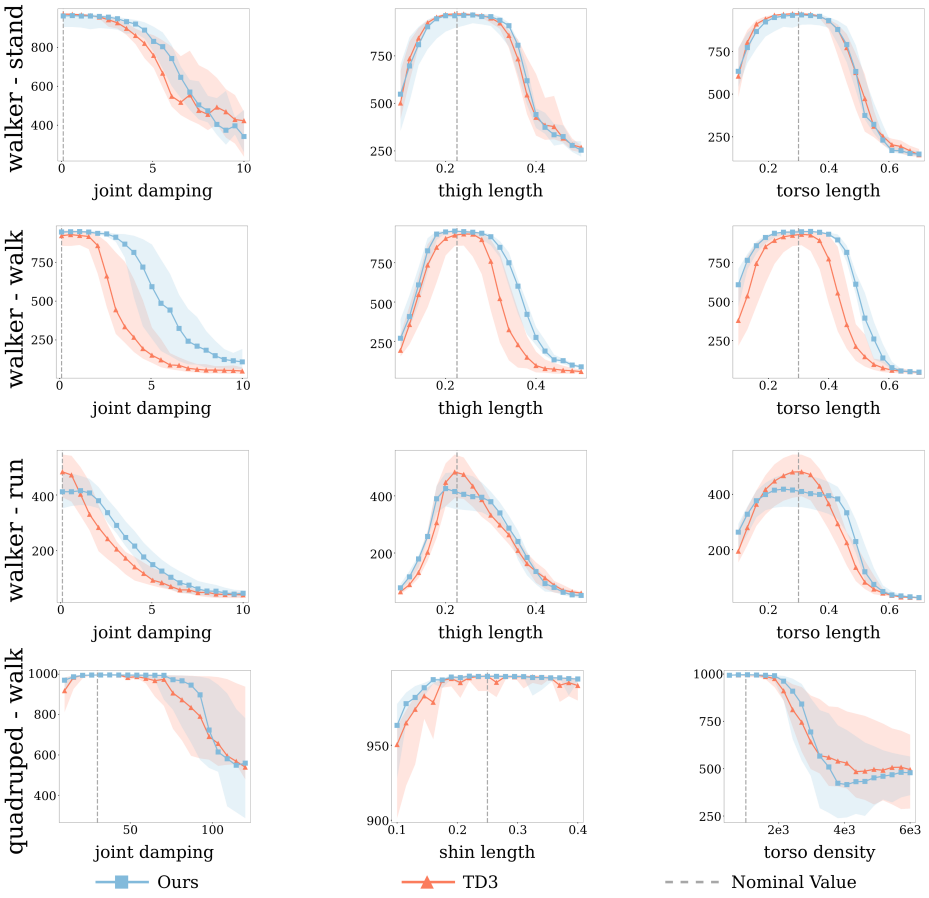

Figure 13: Evaluation results on DeepMind Control tasks with perturbed environment parameters.

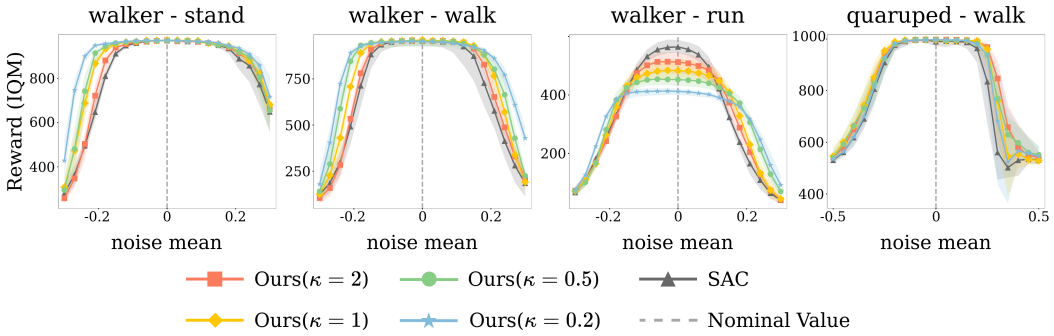

Figure 14: Evaluation results on DeepMind Control tasks with noise perturbations for different κ.

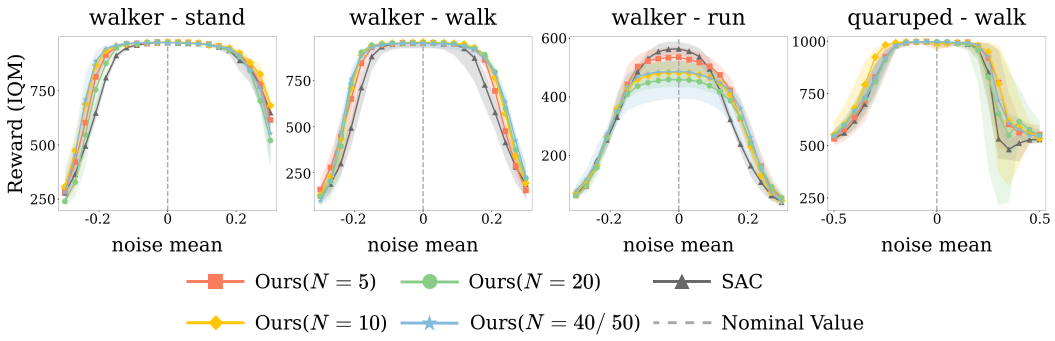

Figure 15: Evaluation results on DeepMind Control tasks with noise perturbations for different N.