Robust Policy Gradient Optimization through Parameter Perturbation in Reinforcement Learning

Md Masudur Rahman
Purdue University
West Lafayette, IN 47906
rahman64@purdue.edu

Juan P. Wachs
Purdue University
West Lafayette, IN 47906
jpwachs@purdue.edu

Yexiang Xue
Purdue University
West Lafayette, IN 47906
yexiang@purdue.edu

Abstract

Policy gradient methods have achieved strong performance in reinforcement learning, yet remain vulnerable to premature convergence and poor generalization, especially in on-policy settings where exploration is limited. Existing solutions typically rely on entropy regularization or action noise, but these approaches either require sensitive hyperparameter tuning or affect the interaction dynamics rather than the optimization process itself. In this paper, we propose Robust Policy Optimization (RPO), a policy gradient method that introduces perturbations to the policy parameters only during optimization. This technique smooths the loss landscape and implicitly regularizes learning, reducing sensitivity to local irregularities without altering policy behavior during data collection. We provide a theoretical analysis showing that RPO adds a Hessian-based regularization term to the objective, biasing updates toward flatter, more robust solutions. RPO significantly improves upon PPO and entropy-regularized variants across diverse continuous control benchmarks, achieving faster convergence, higher returns, and greater robustness—without the need for entropy tuning.

1 Introduction

Reinforcement learning (RL) has demonstrated impressive capabilities across a wide range of complex decision-making tasks, including robotic control [Schulman et al., 2017] and game-playing [Mnih et al., 2015, Silver et al., 2018, 2016]. Despite these successes, achieving robust and generalizable behavior remains a core challenge, particularly for on-policy algorithms such as Proximal Policy Optimization (PPO) [Schulman et al., 2017]. These methods often converge prematurely to overly deterministic policies, exploiting suboptimal strategies that perform well in the short term but fail to generalize across varying conditions.

This tendency is partly due to the local nature of gradient-based policy optimization, which can overfit to short-horizon reward signals and produce deterministic policies that generalize poorly across varying state distributions. Such policies often exhibit low entropy and limited adaptability [Mnih et al., 2016, Ahmed et al., 2019, Nikishin et al., 2022], making them brittle under state distribution shifts. Encouraging smoother optimization dynamics—such as through implicit regularization—has been shown in prior work to improve generalization and robustness in both supervised learning and reinforcement learning.

To address this issue, many existing methods introduce entropy regularization [Mnih et al., 2016, Ahmed et al., 2019], adding an entropy bonus to the optimization objective to encourage stochasticity. While this can be effective, it typically requires manual tuning of entropy coefficients and is sensitive to hyperparameter selection [Andrychowicz et al., 2020]. Other approaches, such as action noise injection [Fujimoto et al., 2018, Haarnoja et al., 2018], attempt to improve exploration by adding randomness to actions during environment interaction. However, these methods affect data collection

2nd Workshop on Aligning Reinforcement Learning Experimentalists and Theorists (ARLET 2025).

rather than the underlying optimization dynamics, often resulting in unstable learning or degraded data efficiency.

We propose **Robust Policy Optimization (RPO)**, an effective alternative that introduces perturbations to the policy parameters during training—but not during environment interaction.

While most reinforcement learning algorithms encourage exploration by injecting noise into actions or adding entropy regularization, these techniques primarily influence behavior during environment interaction rather than the optimization process itself [Fujimoto et al., 2018, Haarnoja et al., 2018, Mnih et al., 2016, Ahmed et al., 2019]. In contrast, RPO perturbs the policy parameters only during optimization, leaving interaction dynamics untouched. This approach smooths the loss landscape by averaging over a neighborhood of policies, implicitly regularizing the learning process. As a result, RPO steers policy updates away from brittle, overfit solutions and toward flatter regions of the loss surface—an effect associated with improved generalization and robustness. Moreover, this strategy avoids the need for tuning entropy coefficients or injecting behavioral stochasticity, making it both sample-efficient and easy to implement. Conceptually, RPO is analogous to techniques like dropout [Srivastava et al., 2014] or data augmentation in supervised learning, where randomness introduced during training helps the model generalize better without altering the input distribution.

Formally, RPO modifies the standard policy gradient objective [Sutton et al., 1999] by optimizing an expectation over a perturbed neighborhood of policies: $\tilde{L}(\theta) = \mathbb{E}_z[L(\theta + z)]$, where $z \sim \mathcal{U}(-\alpha, \alpha)$ is a bounded perturbation applied to the policy parameters. This induces a convolutional smoothing effect on the loss surface, reducing sensitivity to sharp local minima and promoting convergence to flatter, more robust solutions.

Our theoretical analysis shows that RPO introduces an implicit regularization effect by penalizing curvature in the optimization landscape. Specifically, we derive that the perturbed objective approximates the original loss plus a regularization term proportional to the trace of the Hessian: $\tilde{L}(\theta) = L(\theta) + \frac{\alpha^2}{6}\text{Tr}(HL(\theta))$. This formulation reveals that RPO encourages convergence to low-curvature regions in parameter space. Importantly, RPO averages the loss over a local neighborhood of policies, providing a smoothing effect that stabilizes learning without biasing the policy gradient estimator.

Empirically, we evaluate RPO across continuous control tasks from DeepMind Control [Tunyasuvunakool et al., 2020], OpenAI Gym [Brockman et al., 2016], PyBullet [Coumans and Bai, 2016–2021], and Nvidia IsaacGym [Makoviychuk et al., 2021]. Across these diverse benchmarks, RPO consistently outperforms PPO and entropy-regularized PPO, achieving faster convergence, higher returns, and greater robustness to varying action distributions and task settings.

Contributions.

- We introduce **Robust Policy Optimization (RPO)**, a novel policy gradient method that applies optimization-time perturbations to the policy parameters. This perturbation acts as a form of loss smoothing, encouraging robust and generalizable policy learning without modifying environment interaction.
- We provide a **theoretical analysis** showing that RPO induces an implicit regularization effect by penalizing the trace of the Hessian, thereby biasing optimization toward flatter regions of the loss landscape.
- We conduct extensive experiments on a range of **continuous control tasks**, demonstrating that RPO consistently outperforms baselines in different settings in terms of sample efficiency, and robustness.

2 Preliminaries: Policy Optimization and Implicit Regularization

We consider a standard Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P(s' \mid s, a)$ is the transition probability function, $r(s, a)$ is the reward function, and $\gamma \in (0, 1)$ is the discount factor. The agent's objective is to learn a policy $\pi(a \mid s; \theta)$, parameterized by $\theta \in \mathbb{R}^d$, that maximizes the expected cumulative return:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right], \tag{1}$$

where $\tau = (s_0, a_0, s_1, a_1, \dots)$ denotes a trajectory generated by following policy π_θ.

2.1 Policy Gradient Methods

Policy gradient methods directly optimize $J(\theta)$ by estimating its gradient using the policy gradient theorem [Sutton et al., 1999]:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a \mid s) \, A^{\pi_\theta}(s, a) \right], \tag{2}$$

where $A^{\pi_\theta}(s, a) = Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s)$ is the advantage function, representing how much better an action is compared to the expected value of the current policy in state s. In practice, the advantage is estimated using temporal-difference methods or generalized advantage estimation (GAE) [Schulman et al., 2016], which balances bias and variance by incorporating multi-step value targets.

This advantage-weighted objective enables actor-critic methods to scale to complex environments by providing lower-variance gradient estimates. To further stabilize training [Kakade, 2001], Trust Region Policy Optimization (TRPO) [Schulman et al., 2015] and Proximal Policy Optimization (PPO) [Schulman et al., 2017] constrain the size of policy updates using trust region constraints or clipped surrogate objectives. Nonetheless, these methods can still suffer from premature convergence to low-entropy or brittle policies [Andrychowicz et al., 2020, Nikishin et al., 2022].

2.2 Implicit Regularization through Optimization-Time Perturbation

Recent work in deep learning has shown that flatter regions of the loss landscape are associated with improved generalization [Keskar et al., 2017]. This insight has motivated the use of implicit regularization techniques such as dropout [Srivastava et al., 2014], data augmentation, batch normalization in supervised [Ioffe and Szegedy, 2015] and off-policy reinforcement learning [Bhatt et al., 2024]. These methods introduce stochasticity during training, which helps models converge to broader optima that generalize better to unseen data.

In reinforcement learning setting issues of generalization and stability arise due to the non-stationary, high-variance nature of policy updates. We are particularly interested in the effect of introducing noise into the policy *parameters during optimization*—not during data collection or interaction with the environment. This idea can be understood through the lens of optimization-time smoothing, a general technique that modifies the training loss by averaging it over a neighborhood in parameter space.

Specifically, we define a smoothed training objective of the form:

$$\tilde{L}(\theta) = \mathbb{E}_{z \sim \mathcal{U}(-\alpha, \alpha)} \left[L(\theta + z) \right], \tag{3}$$

where $z \in \mathbb{R}^d$ is a uniformly sampled perturbation vector and $\alpha > 0$ controls the magnitude of the perturbation. This formulation corresponds to a convolution of the original loss with a uniform kernel, resulting in a smoothed version of the loss landscape.

Although no explicit regularization term is added to the objective, this smoothing induces an implicit bias toward flatter solutions. Prior work [Dziugaite and Roy, 2018, Foret et al., 2021] has shown that this effect can be understood through a second-order Taylor expansion:

$$\tilde{L}(\theta) \approx L(\theta) + \frac{\alpha^2}{6} \mathrm{Tr}(HL(\theta)), \tag{4}$$

where $HL(\theta)$ denotes the Hessian of the loss with respect to θ. The trace term penalizes curvature, discouraging convergence to sharp local minima and instead favoring broader, more stable regions of the loss surface.

This theoretical connection between curvature and generalization has been extensively validated in supervised learning [Dziugaite and Roy, 2018, Foret et al., 2021]. Sharp minima lead to high sensitivity to small changes in model parameters and poor robustness under distribution shifts. Implicit regularization via loss smoothing helps avoid such solutions, yielding models that are more robust and generalizable.

In the following section, we apply this principle to policy gradient methods by introducing optimization-time parameter perturbations. Our approach leverages this implicit regularization effect to stabilize training and improve generalization in reinforcement learning.

3 Methodology: Robust Policy Optimization (RPO)

3.1 Algorithmic Description

Robust Policy Optimization (RPO) is a modified policy gradient method that introduces perturbations to the policy parameters during the optimization phase, while keeping the data collection policy unchanged. This separation aims to ensure the exploration behavior remains stable during environment interaction, while optimization benefits from smoother gradient estimates. Unlike traditional entropy regularization, which requires manually tuned coefficients and modifies the reward objective, RPO regularizes implicitly by averaging gradient updates over a neighborhood in parameter space. The details of the RPO method are presented in Algorithm 1.

Algorithm 1 Robust Policy Optimization (RPO)

Initialize: Policy parameters θ, experience buffer $\mathcal{D}$
for each iteration **do**
  $\mathcal{D} \leftarrow \emptyset$ ▷ Initialize batch storage
  for each environment step t **do**
    $(\mu_t, \sigma_t) \leftarrow \pi_\theta(s_t)$ ▷ Compute action distribution
    $a_t \sim \mathcal{N}(\mu_t, \sigma_t)$ ▷ Sample action
    Execute a_t, observe s_{t+1}, r_t
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
  end for
  Compute advantages A_t for each $(s_t, a_t) \in \mathcal{D}$
  for each transition $(s_t, a_t, A_t) \in \mathcal{D}$ **do**
    $(\mu_t, \sigma_t) \leftarrow \pi_\theta(s_t)$ ▷ Compute policy mean and std
    Sample perturbation $z \sim \mathcal{U}(-\alpha, \alpha)$
    Set perturbed mean $\mu'_t = \mu_t + z$
    Compute log-probability $\log \pi_\theta(a_t \mid s_t; \mu'_t, \sigma_t)$
    $g \leftarrow g + \nabla_\theta \log \pi_\theta(a_t \mid s_t; \mu'_t, \sigma_t) A_t$ ▷ Accumulate policy gradient
  end for
  Update θ using accumulated gradients
end for

3.2 Theoretical Analysis

The key intuition behind Robust Policy Optimization (RPO) is that perturbing the policy parameters during the optimization phase introduces a controlled stochasticity. By decoupling perturbation from data collection, RPO aims to avoid the instability associated with action-space noise, while still encouraging diversity in gradient updates. Unlike traditional entropy-regularized methods, which require manual tuning of entropy coefficients and modify the reward function, RPO implicitly maintains policy stochasticity through optimization-time noise, without altering the objective.

In the continuous control setting, the policy is typically modeled as a Gaussian distribution:

$$\pi_\theta(a \mid s) = \mathcal{N}(\mu_\theta(s), \sigma_\theta(s)),$$

where $\mu_\theta(s)$ and $\sigma_\theta(s)$ are the outputs of a neural network parameterized by θ. Gaussian policies are widely used in continuous control due to their analytical tractability and compatibility with reparameterization-based gradient estimation.

During optimization, RPO perturbs the mean as:

$$\mu' = \mu + z, \quad z \sim \mathcal{U}(-\alpha, \alpha),$$

resulting in a modified policy distribution:

$$\tilde{\pi}_\theta(a \mid s) = \mathbb{E}_z \left[\mathcal{N}(a \mid \mu + z, \sigma)\right].$$

This formulation ensures that policy gradient updates are influenced by a neighborhood of parameter values rather than a single deterministic policy. As a result, RPO naturally prevents premature convergence to deterministic, low-entropy solutions and encourages robust, diverse behavior.

We next provide a formal analysis of how this perturbation affects gradient estimation and induces an implicit regularization effect that improves generalization and optimization stability.

3.3 Unbiasedness of the Gradient Estimator

A desirable property of any modification to the policy gradient method is that it does not introduce bias in the gradient estimate. In this subsection, we show that RPO preserves unbiasedness despite applying perturbations to the policy parameters during optimization.

The standard policy gradient is:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(a \mid s) A(s, a)\right].$$

In RPO, the policy distribution is perturbed during optimization:

$$\pi_\theta^{\text{pert}}(a \mid s; z) = \mathcal{N}(\mu_\theta(s) + z, \sigma_\theta(s)), \quad z \sim \mathcal{U}(-\alpha, \alpha),$$

but the data used for gradient estimation (states and actions) is collected using the unperturbed policy π_θ.

The RPO update computes:

$$\tilde{g}(\theta) = \mathbb{E}_z \left[\nabla_\theta \log \pi_\theta(a \mid s; \mu + z, \sigma) A(s, a)\right].$$

Because the perturbation z is sampled independently from a symmetric zero-mean distribution and does not affect the data distribution, we can apply linearity of expectation:

$$\mathbb{E}_z \left[\nabla_\theta \log \pi_\theta(a \mid s; \mu + z, \sigma)\right] = \nabla_\theta \mathbb{E}_z \left[\log \pi_\theta(a \mid s; \mu + z, \sigma)\right].$$

Since the policy is Gaussian and smooth, this expectation can be viewed as convolution with a uniform kernel, which preserves the mean gradient direction. Hence:

$$\mathbb{E}_z \left[\nabla_\theta \log \pi_\theta(a \mid s; \mu + z, \sigma)\right] = \nabla_\theta \log \pi_\theta(a \mid s),$$

and the full RPO estimator becomes:

$$\mathbb{E}_z[\tilde{g}(\theta)] = \nabla_\theta \log \pi_\theta(a \mid s) A(s, a).$$

Thus, RPO does not introduce any bias into the policy gradient estimator [Sutton et al., 1999]. It only modifies the way the gradient is computed during the optimization step, without affecting the trajectory distribution or introducing systematic drift. This unbiasedness is crucial for ensuring that RPO retains the convergence guarantees of standard policy gradient methods [Sutton et al., 1999, Schulman et al., 2015, Kakade, 2001], while benefiting from the regularization and smoothing effects of the perturbation.

3.4 Induced Regularization Effect

Beyond preserving the unbiasedness of the policy gradient, RPO also induces an implicit regularization effect. By introducing perturbations to the policy parameters during optimization, the loss landscape becomes smoothed, discouraging convergence to narrow optima and improving generalization.

To illustrate this effect, consider the expected log-likelihood under perturbation:

$$\mathbb{E}_z \left[-\log \pi_\theta(a \mid s; \mu + z, \sigma)\right].$$

Due to the convexity of the negative log-likelihood in the perturbed mean and the symmetry of the perturbation distribution, this expectation introduces a regularization term. Specifically, using a second-order Taylor expansion, it can be shown (details in Appendix) that:

$$\mathbb{E}_z\left[-\log \pi_\theta(a \mid s; \mu + z, \sigma)\right] \approx -\log \pi_\theta(a \mid s) + \frac{\alpha^2}{6\sigma^2}.$$

This extra term penalizes low variance in the action distribution by discouraging overly sharp likelihood peaks. As a result, RPO acts similarly to entropy regularization, but without requiring explicit entropy bonuses or tuning of exploration coefficients. Instead, it maintains stochasticity through parameter-space smoothing, providing a self-regularizing mechanism that encourages more robust, generalizable policies. We provide a full derivation of this regularization effect in Appendix.

3.5 Smoothing of the Loss Landscape

A key effect of Robust Policy Optimization (RPO) is that it smooths the loss landscape by averaging the policy gradient objective over a local neighborhood in parameter space. This reduces the optimizer's sensitivity to sharp local variations and encourages convergence to flatter, more stable regions.

Formally, instead of minimizing the standard loss $L(\theta)$, RPO optimizes the smoothed objective:

$$\tilde{L}(\theta) = \mathbb{E}_{z\sim\mathcal{U}(-\alpha,\alpha)}\left[L(\theta + z)\right],$$

which corresponds to convolving the original loss function with a uniform kernel. This operation averages the loss over a region around θ, effectively dampening high-curvature fluctuations in the landscape.

Using a second-order Taylor expansion, we show (details in Appendix) that this smoothing introduces a regularization term proportional to the trace of the Hessian:

$$\tilde{L}(\theta) \approx L(\theta) + \frac{\alpha^2}{6}\mathrm{Tr}(\nabla^2 L(\theta)).$$

This additional term penalizes sharp curvature in the loss function, encouraging updates that lead to flatter minima. As flatter solutions are known to generalize better in deep learning and reinforcement learning, this implicit smoothing mechanism contributes to RPO's improved robustness and stability without requiring any explicit regularization term or change to the reward structure.

4 Experiments

4.1 Setup

Environments We conducted experiments on continuous control task from four reinforcement learning benchmarks: DeepMind Control [Tunyasuvunakool et al., 2020], OpenAI Gym Mujoco control tasks [Brockman et al., 2016], PyBullet [Coumans and Bai, 2016–2021], and Nvidia IsaacGym [Makoviychuk et al., 2021]. These benchmarks contain diverse environments with many different tasks, from low to high-dimensional environments (observations and actions space). Thus our evaluation contains a diverse set of tasks with various difficulties.

Baselines We compare our method RPO with the standard **PPO** [Schulman et al., 2017] algorithm. Here our method RPO uses the perturbed Gaussian distribution to represent the action output from the policy network. In contrast, the PPO uses standard Gaussian distribution to represent its action output. Unless otherwise mentioned, we tested on Gaussian distribution. However, we provide separate experiments for different parameterized distributions, including Laplace and Gumbel.

Another common approach to increase entropy is to use the **Entropy Regularization** [Mnih et al., 2016] in the RL objective. A coefficient determines how much weight the policy would give to the entropy. We observe that various weighting might result in different levels of entropy increment. We use the entropy coefficient 0.0, 0.01, 0.05, 0.5, 1.0, and 10.0 and compare their performance in entropy and, in return, with our RPO algorithm. Note that our method does not use the entropy coefficient hyper-parameters.

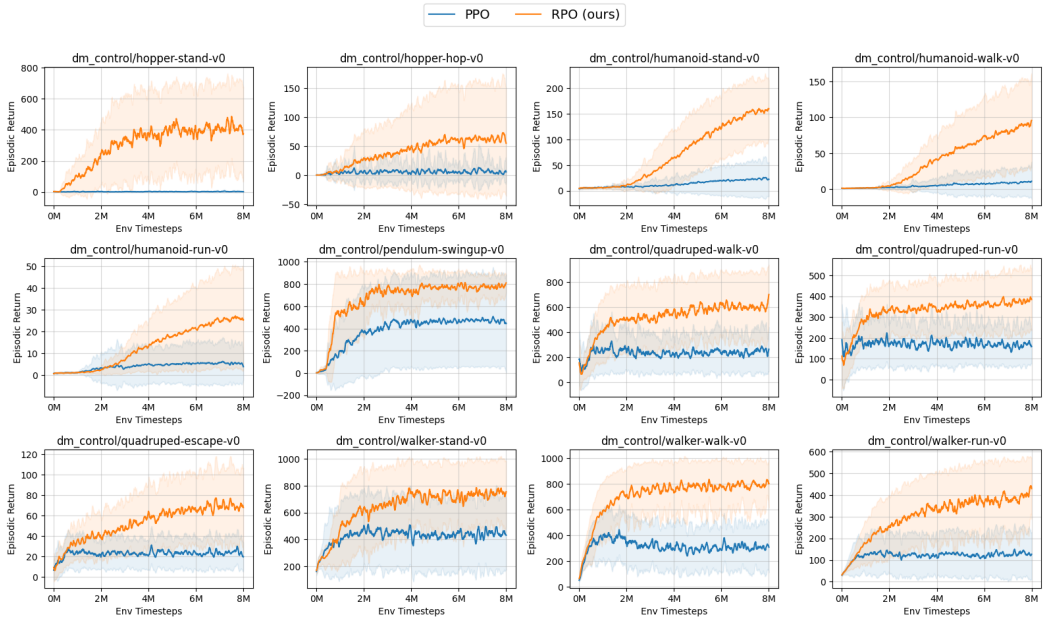

Figure 1: Results on DeepMind Control Environments. PPO agent fails to learn any useful behavior and thus results in low episodic return in some environments (humanoid (stand, walk, run) and hopper (stand hop)). Overall, our method RPO performs better in episodic return. In many settings, PPO agents stop improving their performance after around 2M timestep while RPO consistently improves over the entire training time.

To account for the stochasticity in the environment and policy, we run each experiment several times and report the mean and standard deviation. Unless otherwise specified, we run each experiment with **10 random seeds**. Further implementation details are in the Appendix.

4.2 Comparison with PPO

We notice a consistent improvement in the performance of our method in many continuous control environments compared to standard PPO. This evaluation is the direct form of comparison where no method uses any aid (such as data augmentation or entropy regularization) in the entropy value. Figure 1 shows results comparison on *DeepMind Control Environments*. Results on more environments and entropy comparisons are in the Appendix.

In most scenarios, our method RPO shows consistent performance improvement in these environments compared to the PPO. In some environments, such as humanoid stand, humanoid run, and hopper hop, the PPO agent fails to learn any useful behavior and thus results in low episodic return. In contrast, our method RPO shows better performance and achieves a much higher episodic return. The RPO also shows a better mean return than the PPO in other environments.

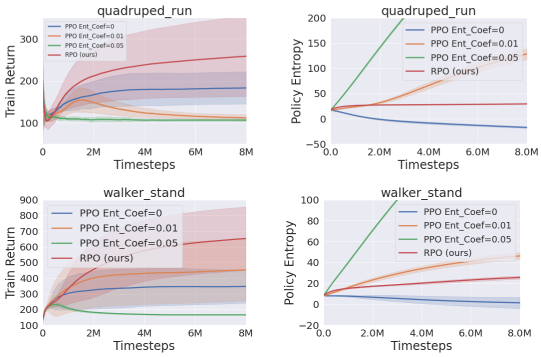

Figure 2: Comparison with Entropy Regularization. While entropy regularization can help with careful tuning (e.g., walker stand), poor coefficient choices often degrade performance. In contrast, RPO consistently outperforms PPO and all entropy-tuned baselines.

In many settings, such as in quadruped (walk, run, and escape), walker (stand, walk, run), fish swim, acrobot swingup, the PPO agents stop im-

proving their performance after around 2M timesteps. In contrast, our agent RPO shows consistent improvement over the course of training. This performance gain might be due to the proper management of policy entropy, as in our setup, the agent is encouraged to keep exploring as the training progresses. On the other hand, the PPO agent might settle in sub-optimal performance as the policy entropy, in this case, decreases as the agent trains for more timesteps. These results show the effectiveness of our method in diverse control tasks with varying complexity. *Experimental results for other continuous benchmarks, such as those in IsaacGym, Gym(nasium) Mujoco control, and PyBullet environments, are provided in the Appendix.*

4.3 Comparison with Entropy Regularization

A way to control entropy is to use an entropy regularizer [Mnih et al., 2016, Ahmed et al., 2019], which often shows beneficial effects. However, it has been observed that increasing entropy in such a way has little effect in specific environments [Andrychowicz et al., 2020]. These results show difficulty in setting proper entropy throughout the agent's training. Due to the variety in the environments, the entropy requirement might be different. Thus, improper setting of the entropy coefficient might result in bad training performance. In the tested environments, we observe that sometimes the performance improves with a proper setup of tuned coefficient value and often worsens the performance (in Figure 2). However, our method RPO consistently performs better than PPO and the different entropy coefficient baselines. Importantly, our method does not use these coefficient hyper-parameters and still consistently performs better in various environments, which shows our method's robustness in various environment variability.

We observe that in some environments, the coefficient 0.01 improves the performance of standard PPO with coefficient 0.0 while increasing the entropy. However, an increase in entropy to 0.05 and above results in an unbounded entropy increase. Thus, the performance worsens in most scenarios where the agent fails to achieve a reasonable return. Results with all the coefficient variants in the Appendix. Overall, our method RPO achieves better performance in the evaluated environments in our setup. Moreover, our method does not use the entropy coefficient hyperparameter and controls the entropy level automatically in each environment.

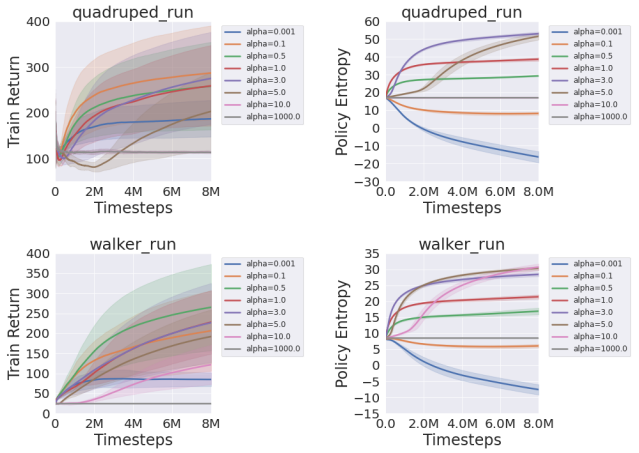

Figure 3: Ablation on $alpha$ values of the uniform distribution for RPO. An α value between 0.1 to 3 often results in better performance, while a large value often results in worse performance. The suggested default value is $\alpha = 0.5$.

5 Ablation Study

5.1 Effect of α on RPO

We conducted experiments on the α value ranges in the Uniform distribution. Figure 3 shows the return and policy comparison. We observe that the value of α affects the policy entropy and, thus, return performance. A smaller value of α (e.g., 0.001) seems to behave similarly to PPO, where policy entropy decreases over time, thus hampered performance. Higher entropy values, such as 1000.0, make the policy somewhat random as the uniform distribution dominates over the Gaussian distribution. This scenario keeps the entropy somewhat at a constant level; thus, the performance is hampered. Overall, a value between 0.1 to 3 often results in better performance. Due to overall performance advantage, in this paper, we report results with $\alpha = 0.5$ for all environments.

5.2 Results on RPO-α Annealing

We implement a version of our RPO algorithm that varies the α parameter, which controls the amount of uniform perturbation applied to the policy. Specifically, we anneal α from its default value (0.5 in our main experiments) down to zero over the course of training. The value decreases uniformly at each policy update, ensuring that the injected randomness is gradually reduced and ultimately removed by the end of training.

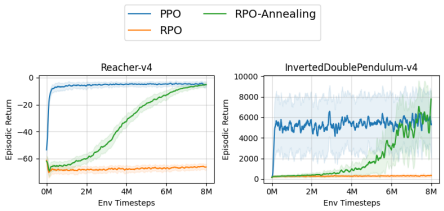

Figure 4: RPO-Annealing on Mujoco-v4. In two challenging cases where the default RPO ($\alpha = 0.5$) fails, the annealed version recovers and eventually matches PPO's performance.

We evaluate RPO-Annealing on challenging Mujoco-v4 environments, where the default RPO setting ($\alpha = 0.5$) fails to learn. In these cases, the annealed version recovers and eventually matches the performance of the base PPO. Notably, RPO can also succeed in these environments by fine-tuning the α value, as shown in Figure 4. Overall, annealing α is a promising strategy that can improve performance and enhance stability. Moreover, proper tuning of α can lead to even better outcomes compared to the default, as evidenced in Figure 4. Further results are in the Appendix.

5.3 Effect of Action Distribution

Figure 5 shows the performance of RPO across different action distributions (Gaussian, Laplace, and Gumbel) on Mujoco and Bullet environments. Empirically, RPO improves performance over PPO even when PPO uses alternative action distributions, demonstrating RPO's robustness. While RPO consistently enhances learning across all tested distributions, we observe that the Gaussian distribution tends to perform best for continuous control tasks. Additional results are provided in the Appendix.

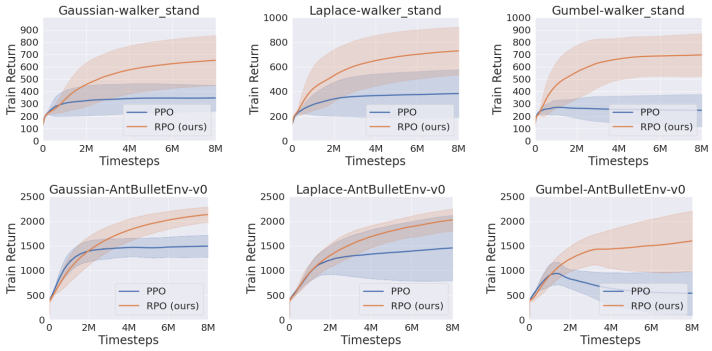

Figure 5: Results on different action distributions. Our method RPO shows improvement compared to base distributions.

6 Conclusion

We introduced Robust Policy Optimization (RPO), an effective enhancement to standard policy gradient methods that improves robustness by applying parameter perturbations during optimization. Unlike existing approaches that rely on entropy regularization or action noise, RPO preserves policy behavior during environment interaction and removes the need for tuning entropy coefficients. Our theoretical analysis shows that RPO implicitly regularizes the loss surface by penalizing high curvature, guiding optimization toward flatter and more stable solutions. Extensive experiments across diverse continuous control benchmarks demonstrate that RPO consistently outperforms PPO and entropy-regularized variants, achieving faster convergence, higher returns, and improved robustness under varying action distributions. Overall, RPO provides a practical and theoretically grounded framework for more consistent and adaptable on-policy learning in reinforcement learning.

Acknowledgments and Disclosure of Funding

This work was supported by NSF under Award No. 521982.

References

Zafarali Ahmed, Nicolas Le Roux, Mohammad Norouzi, and Dale Schuurmans. Understanding the impact of entropy on policy optimization. In *International conference on machine learning*, pages 151–160. PMLR, 2019.

Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Leonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, et al. What matters for on-policy deep actor-critic methods? a large-scale study. In *International conference on learning representations*, 2020.

Aditya Bhatt, Daniel Palenicek, Boris Belousov, Max Argus, Artemij Amiranashvili, Thomas Brox, and Jan Peters. Crossq: Batch normalization in deep reinforcement learning for greater sample efficiency and simplicity. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=PczQtTsTIX`.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning, 2016–2021.

Gintare Karolina Dziugaite and Daniel Roy. Entropy-sgd optimizes the prior of a pac-bayes bound: Generalization properties of entropy-sgd and data-dependent priors. In *International Conference on Machine Learning*, pages 1377–1386. PMLR, 2018.

Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=6Tm1mposlrM`.

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.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

Shengyi Huang, Rousslan Fernand Julien Dossa, Antonin Raffin, Anssi Kanervisto, and Weixun Wang. The 37 implementation details of proximal policy optimization. In *ICLR Blog Track*, 2022a. URL `https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/`. https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022b. URL `http://jmlr.org/papers/v23/21-1342.html`.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

Sham M Kakade. A natural policy gradient. *Advances in neural information processing systems*, 14, 2001.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations*, 2017.

Misha Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. *Advances in Neural Information Processing Systems*, 33, 2020.

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu-based physics simulation for robot learning, 2021.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In *International Conference on Machine Learning*, pages 16828–16847. PMLR, 2022.

Roberta Raileanu, Max Goldstein, Denis Yarats, Ilya Kostrikov, and Rob Fergus. Automatic data augmentation for generalization in deep reinforcement learning. *arXiv preprint arXiv:2006.12862*, 2020.

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

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

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

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *nature*, 529:484–489, 2016.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362:1140–1144, 2018. doi: 10.1126/science.aar6404. URL `https://www.science.org/doi/abs/10.1126/science.aar6404`.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

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`.

A Derivation of Gradient Unbiasedness under Parameter Perturbation

We now provide a more detailed justification for the identity used in the unbiasedness proof:

$$\mathbb{E}_z \left[\nabla_\theta \log \pi_\theta(a \mid s; \mu + z, \sigma)\right] = \nabla_\theta \log \pi_\theta(a \mid s).$$

Let $\pi_\theta(a \mid s) = \mathcal{N}(\mu_\theta(s), \sigma_\theta(s))$ be a Gaussian policy. During RPO optimization, the mean is perturbed as $\mu' = \mu + z$, where $z \sim \mathcal{U}(-\alpha, \alpha)$ is independent of the policy parameters θ.

Define the perturbed log-likelihood:

$$\ell_z(\theta) := \log \pi_\theta(a \mid s; \mu + z, \sigma).$$

We want to show:

$$\mathbb{E}_z[\nabla_\theta \ell_z(\theta)] = \nabla_\theta \log \pi_\theta(a \mid s).$$

This is justified using the following steps:

Step 1: Expand the log-likelihood of the Gaussian. Let us write:

$$\pi_\theta(a \mid s; \mu + z, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(a - \mu - z)^2}{2\sigma^2}\right).$$

Taking the log:

$$\log \pi_\theta(a \mid s; \mu + z, \sigma) = -\frac{(a - \mu - z)^2}{2\sigma^2} - \log(\sqrt{2\pi\sigma^2}).$$

Step 2: Compute the gradient w.r.t. θ. We compute $\nabla_\theta \log \pi_\theta(a \mid s; \mu + z, \sigma)$. Since z is independent of θ, and $\mu = \mu_\theta(s)$, the derivative only applies to μ (and possibly σ):

$$\nabla_\theta \log \pi_\theta(a \mid s; \mu + z, \sigma) = \frac{(a - \mu - z)}{\sigma^2} \nabla_\theta \mu_\theta(s).$$

Step 3: Take expectation over z. Now take expectation:

$$\mathbb{E}_z \left[\nabla_\theta \log \pi_\theta(a \mid s; \mu + z, \sigma)\right] = \nabla_\theta \mu_\theta(s) \cdot \frac{1}{\sigma^2} \mathbb{E}_z \left[(a - \mu - z)\right].$$

Because z is zero-mean and independent of a and μ:

$$\mathbb{E}_z \left[a - \mu - z\right] = a - \mu.$$

So:

$$\mathbb{E}_z \left[\nabla_\theta \log \pi_\theta(a \mid s; \mu + z, \sigma)\right] = \frac{(a - \mu)}{\sigma^2} \nabla_\theta \mu_\theta(s) = \nabla_\theta \log \pi_\theta(a \mid s).$$

Thus, the expectation of the perturbed gradient matches the original unperturbed policy gradient:

$$\mathbb{E}_z \left[\nabla_\theta \log \pi_\theta(a \mid s; \mu + z, \sigma)\right] = \nabla_\theta \log \pi_\theta(a \mid s),$$

which proves that the RPO gradient estimator is unbiased.

B Derivation of the Regularization Effect

We derive how Robust Policy Optimization (RPO) induces an implicit regularization effect by applying noise to the policy mean during the optimization phase. Specifically, we show that the expected log-likelihood under perturbation introduces a curvature-dependent penalty term, which prevents variance collapse and promotes higher-entropy, more robust policies.

B.1 Setup

Let the policy be Gaussian:

$$\pi_\theta(a \mid s) = \mathcal{N}(\mu_\theta(s), \sigma_\theta(s)),$$

with fixed standard deviation $\sigma = \sigma_\theta(s)$, and $\mu = \mu_\theta(s)$ denoting the mean.

In RPO, we perturb the mean by adding a noise vector $z \sim \mathcal{U}(-\alpha, \alpha)$, leading to:

$$\mu' = \mu + z.$$

We analyze the effect of computing the expected negative log-likelihood over this perturbation:

$$\mathbb{E}_z \left[-\log \pi_\theta(a \mid s; \mu + z, \sigma) \right].$$

B.2 Negative Log-Likelihood

Recall the negative log-likelihood for a Gaussian:

$$-\log \pi_\theta(a \mid s; \mu + z, \sigma) = \frac{(a - \mu - z)^2}{2\sigma^2} + \frac{1}{2} \log(2\pi\sigma^2).$$

We expand the squared term:

$$(a - \mu - z)^2 = (a - \mu)^2 - 2z(a - \mu) + z^2.$$

Taking expectation over the uniform distribution $z \sim \mathcal{U}(-\alpha, \alpha)$: - $\mathbb{E}[z] = 0$, - $\mathbb{E}[z^2] = \frac{\alpha^2}{3}$.

Thus:

$$\mathbb{E}_z \left[(a - \mu - z)^2 \right] = (a - \mu)^2 + \frac{\alpha^2}{3}.$$

Substituting into the expectation of the log-likelihood:

$$\mathbb{E}_z \left[-\log \pi_\theta(a \mid s; \mu + z, \sigma) \right] = \frac{(a - \mu)^2}{2\sigma^2} + \frac{\alpha^2}{6\sigma^2} + \frac{1}{2} \log(2\pi\sigma^2).$$

The first and third terms correspond to the unperturbed negative log-likelihood. The second term is the added regularization:

$$\frac{\alpha^2}{6\sigma^2}.$$

B.3 Interpretation

We conclude that:

$$\mathbb{E}_z \left[-\log \pi_\theta(a \mid s; \mu + z, \sigma) \right] \approx -\log \pi_\theta(a \mid s) + \frac{\alpha^2}{6\sigma^2}.$$

This extra term penalizes small standard deviation σ, thereby preventing entropy collapse and implicitly encouraging broader, smoother action distributions. Importantly, this effect arises without requiring an explicit entropy bonus in the objective function.

Hence, RPO behaves like an entropy-regularized policy gradient method, but achieves this through optimization-time parameter perturbation, without modifying the reward or introducing additional hyperparameters.

C Loss Landscape Smoothing via Taylor Expansion

We show how Robust Policy Optimization (RPO) smooths the policy gradient loss landscape by averaging over perturbed parameters. This leads to a regularization effect that penalizes sharp curvature and encourages more stable, generalizable solutions.

C.1 Interpretation as a Convolution with a Uniform Kernel

The RPO objective can be interpreted as a convolution between the loss function and a uniform smoothing kernel. Specifically, the smoothed loss is defined as:

$$\tilde{L}(\theta) = \mathbb{E}_{z\sim\mathcal{U}(-\alpha,\alpha)}\left[L(\theta+z)\right] = \int_{\mathbb{R}^d} L(\theta+z)p(z)\,dz.$$

Here, $p(z)$ is the probability density function of a uniform distribution over a hypercube $[-\alpha,\alpha]^d$. It is defined as:

$$p(z) = \frac{1}{(2\alpha)^d}\cdot\mathbb{I}[z\in[-\alpha,\alpha]^d],$$

where $\mathbb{I}[z\in[-\alpha,\alpha]^d]$ is the indicator function, equal to 1 when all components of z lie within the interval $[-\alpha,\alpha]$, and 0 otherwise. This ensures that the density is constant over the support and integrates to 1:

$$\int_{\mathbb{R}^d} p(z)\,dz = 1.$$

This convolutional smoothing can be understood as a low-pass filter on the loss surface: it averages the value of the loss over a local neighborhood, reducing sensitivity to high-frequency variations. As we will show in the next subsection, this results in an explicit curvature-penalizing regularization term.

C.2 Second-Order Expansion

To analyze the regularization effect, we apply a second-order Taylor expansion of $L(\theta+z)$ around θ:

$$L(\theta+z) \approx L(\theta) + \nabla L(\theta)^\top z + \frac{1}{2}z^\top HL(\theta)z,$$

where $HL(\theta) = \nabla^2 L(\theta)$ is the Hessian of the loss.

Taking the expectation over $z\sim\mathcal{U}(-\alpha,\alpha)^d$, we use the facts: - $\mathbb{E}[z_i]=0$, - $\mathbb{E}[z_iz_j]=0$ for $i\neq j$, - $\mathbb{E}[z_i^2]=\frac{\alpha^2}{3}$.

Then:

$$\mathbb{E}_z\left[z^\top HL(\theta)z\right] = \sum_{i=1}^d H_{ii}\mathbb{E}[z_i^2] = \frac{\alpha^2}{3}\text{Tr}(HL(\theta)).$$

So the smoothed loss becomes:

$$\tilde{L}(\theta) = L(\theta) + \frac{\alpha^2}{6}\text{Tr}(HL(\theta)).$$

C.3 Interpretation

This shows that the perturbation introduces a curvature-dependent regularization term:

$$\tilde{L}(\theta) = L(\theta) + \frac{\alpha^2}{6}\sum_{i=1}^d \lambda_i,$$

where λ_i are the eigenvalues of the Hessian. The trace term penalizes high curvature, effectively encouraging convergence to flatter regions of the loss landscape.

This regularization: (i) Damps sensitivity to small parameter shifts, (ii) Reduces susceptibility to sharp minima, (iii) Promotes better generalization.

Thus, RPO performs a *convolutional smoothing* of the policy gradient loss, without modifying the reward or requiring additional loss terms.

D Choice of Noise Distribution in RPO

The choice of perturbation distribution plays a critical role in the effectiveness and stability of Robust Policy Optimization (RPO). In this section, we examine the theoretical properties and practical implications of using different noise distributions for parameter perturbation. Our focus is on three representative choices: uniform, Laplace, and Gumbel noise (Table 1).

Table 1: Properties of noise distributions considered for RPO perturbations

Distribution	$\mathbb{E}[z]$	$\mathrm{Var}(z)$	**Effect in RPO**
Uniform $\mathcal{U}(-\alpha, \alpha)$	0	$\frac{\alpha^2}{3}$	Bounded, unbiased, analytically stable
Laplace $\mathrm{Lap}(0, b)$	0	$2b^2$	Heavy-tailed, promotes exploration but less stable
Gumbel $\mathrm{Gumbel}(\mu, \beta)$	$\mu + \beta\gamma$	$\frac{\pi^2\beta^2}{6}$	Asymmetric, introduces gradient bias

Laplace noise has a sharp peak and heavier tails than Gaussian, which can encourage exploration through larger and more frequent jumps in parameter space. However, this increased variance often leads to less stable learning and can hinder convergence in high-variance environments typical of reinforcement learning.

Gumbel noise, which is commonly used in sampling-based techniques, is asymmetric and has a nonzero mean $\mu + \beta\gamma$, where $\gamma \approx 0.577$ is the Euler–Mascheroni constant. This asymmetry introduces a directional bias in gradient updates, violating the zero-mean requirement needed for unbiased learning in RPO.

In contrast, uniform noise is symmetric, zero-mean, and bounded. These properties make it analytically tractable and ensure that perturbations remain within a well-controlled region of the parameter space. It enables closed-form derivations of the smoothing effect and regularization strength, while maintaining unbiasedness in gradient estimation.

We also provide an empirical comparison of these noise types in our ablation studies. While Laplace and Gumbel noise may encourage short-term exploration, they often result in higher performance variance and instability during training. In contrast, uniform noise consistently produces smoother learning curves and better generalization, confirming its suitability as the default perturbation model for RPO.

E Additional Experiments Details

Implementation Details: Our algorithm and the baselines are based on the PPO [Schulman et al., 2017] implementation available in [Huang et al., 2022a,b]. This implementation incorporated many important advancements from existing literature in recent years on policy gradient (e.g., Orthogonal Initialization, GAE, Entropy Regularization). We refer reader to [Huang et al., 2022a] for further references. The experiments were conducted on a CPU-enabled machine, where each run of the algorithms generally took between 4 and 10 hours.

The pure data augmentation baseline RAD [Laskin et al., 2020] uses data processing before passing it to the agent, and the DRAC [Raileanu et al., 2020] uses data augmentation to regularize the loss of value and policy network. We experimented with vector-based states and used *random amplitude scaling* proposed in RAD [Laskin et al., 2020] as a data augmentation method for RAD and DRAC. In the random amplitude scaling, the state values are multiplied with random values generated uniformly between a range α to β. We used the suggested [Laskin et al., 2020] and better performing range $\alpha = 0.6$ to $\beta = 1.2$ for all the experiments. Moreover, both RAD and DRAC use PPO as their base algorithm. However, our RPO method does not use any form of data augmentation.

We use the hyperparameters reported in the PPO implementation of continuous action spaces [Huang et al., 2022a,b], which incorporate best practices in the continuous control task. Furthermore, to mitigate the effect of hyperparameters choice, we keep them the same for all the environments. Further, we keep the same hyperparameters for all agents for a fair comparison. The common hyperparameters can be found in Table 2.

Table 2: Hyperparameters for the experiments.

Description	Value
Number of rollout steps	2048
Learning rate	$3e-4$
Discount factor gamma	0.99
Lambda for the GAE	0.95
Number of mini-batches	32
Epochs to update the policy	10
Advantages normalization	True
Surrogate clipping coefficient	0.2
Clip value loss	Yes
Value loss coefficient	0.5

F Additional Results Comparison with PPO

Results on All 48 tasks DeepMind Control is in Figure 6. We observe that the performance improved in many of those environments and remain similar in others.

Results of *OpenAI Gym* environments: Pendulum and BipedalWalker are in Figure 7. Overall, our method, RPO, performs better compared to the PPO. In Pendulum environments, the PPO agent fails to learn any useful behavior in this setup. In contrast, RPO consistently learns with the increase in timestep and eventually learns the task. We see the policy entropy of RPO increases initially and eventually remains at a certain threshold, which might help the policy to stay exploratory and collect more data. In contrast, the PPO policy entropy decreases over time, and thus eventually, the performance remains the same, and the policy stops learning. This scenario might contribute to the bad performance of the PPO.

In the BipedalWalker environment, we see that both PPO and RPO learn up to a certain reward quickly. However, as we keep training both policies, we observe that after a certain period, the PPO's performance drops and even starts to become worse. In contrast, the RPO stays robust as we train for more and eventually keep improving the performance. These results show the robustness of our method when ample train time is available. The entropy plot shows a similar pattern, as PPO decreases entropy over time, and RPO keeps the entropy at a certain threshold.

Figure 8 shows results comparison on **IsaacGym** environments: Cartpole, and BallBalance. In this setup, we run the simulation up to 100M timesteps which take around 30 minutes for each run in each environment in a Quadro RTX 4000 GPU. We see that for Cartpole, both PPO and RPO learn the reward quickly, around 450. However, as we kept training for a long, the performance of PPO started to degrade over time, and the policy entropy kept decreasing. On the other hand, our RPO agent keeps improving the performance; notably, the performance never degrades over time. The policy entropy shows that the entropy remains at a threshold. This exploratory nature of RPO's policy might help keep learning and get better rewards. These results show the robustness of our method RPO over PPO even when an abundance of simulation is available. Interestingly, more simulation data might not always be good for RL agents. In our setup, the PPO even suffers from further training in the Cartpole environment. In the BallBalance environment (results are averaged over 3 random seed runs), our method RPO achieves a slight performance improvement over PPO. Overall, our method RPO performs better than the PPO in the two IsaacGym environments.

Figure 9 shows results comparison on **PyBullet** environments: Ant, and Minituar. We observe that our method RPO performs better than the PPO in the Ant environment. In the Minitaur environment, PPO quickly (at around 2M) learns up to a certain reward and remains on the same performance as time progresses. In contrast, RPO starts from a lower performance, eventually surpassing the PPO's performance as time progresses. These results show the robustness of consistently improving the policy of the RPO method. The entropy pattern remains the same in both cases; PPO reduces entropy while RPO keeps the entropy at a certain threshold which it learns automatically in an environment.

Figure 10 shows the results comparison of RPO in **OpenAI Mujoco-v4** environments (with defaults $\alpha = 0.5$) values.

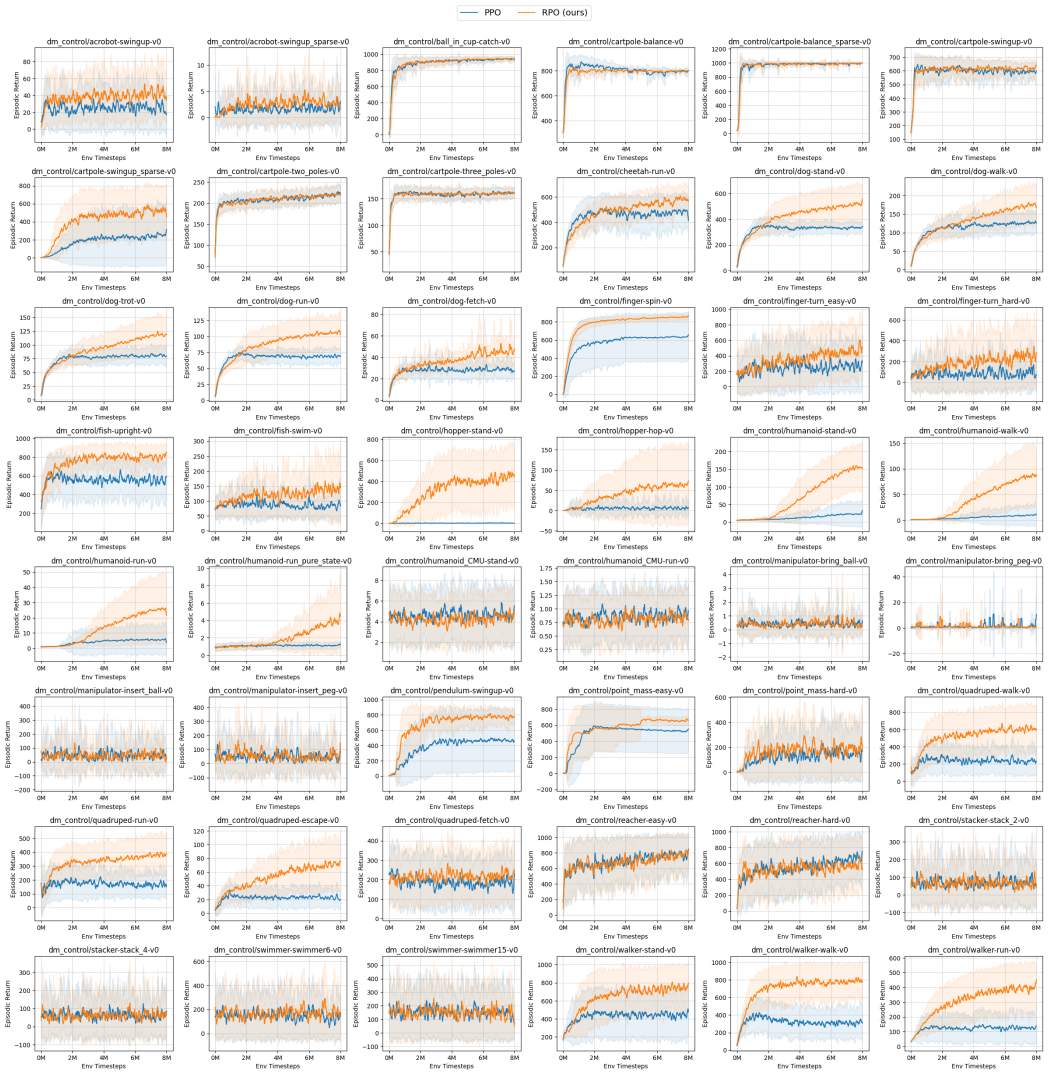

Figure 6: Our method RPO performs better or similarly in **48 DeepMind control environments**.

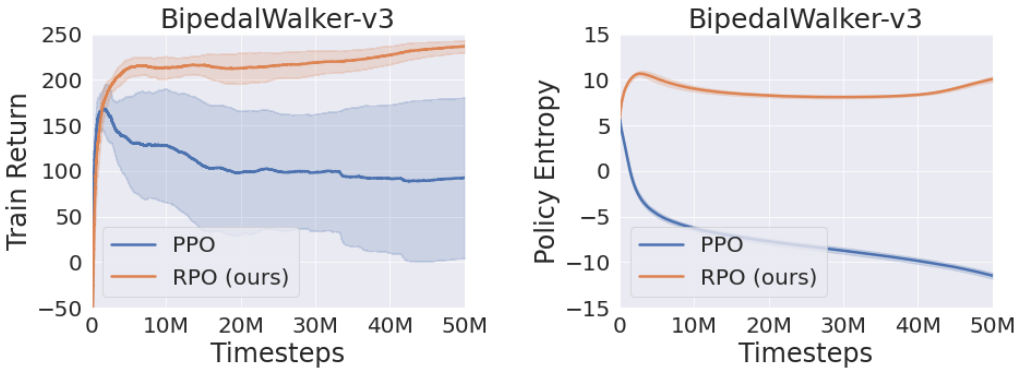

Figure 7: Results on OpenAI **Gym Environments**, BipedalWalker environment. Our RPO agent consistently improves its performance throughout training.

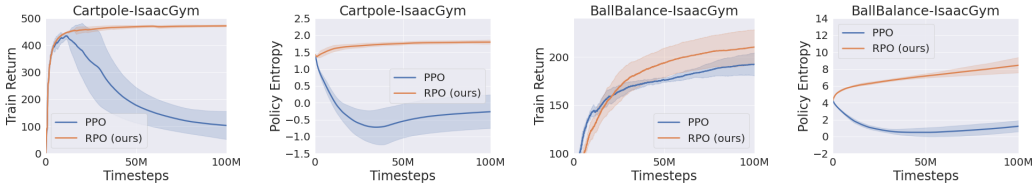

Figure 8: Results on **IsaacGym Environments**. In the Cartpole environment, the performance of PPO started to degrade over time, and the policy entropy kept decreasing. In contrast, our RPO agent keeps improving the performance over the entire training time. These results show the robustness of our method RPO over PPO even when an abundance of simulation is available. In the BallBalance environment, our method RPO achieves a slight performance improvement compared to PPO.

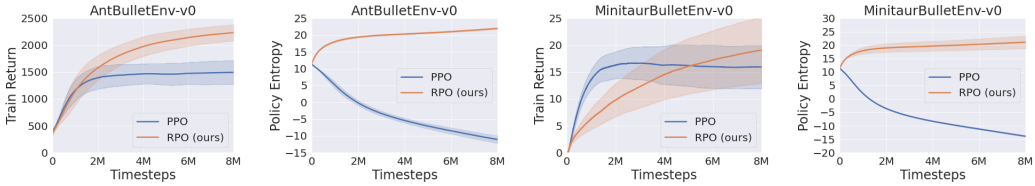

Figure 9: Results on **PyBullet Environments**. Our method RPO consistently performs better than the PPO in the Ant environment. On the other hand, in the Minitaur environment, PPO quickly learns up to a particular reward and remains on the same performance as time progresses while RPO surpasses the PPO's performance.

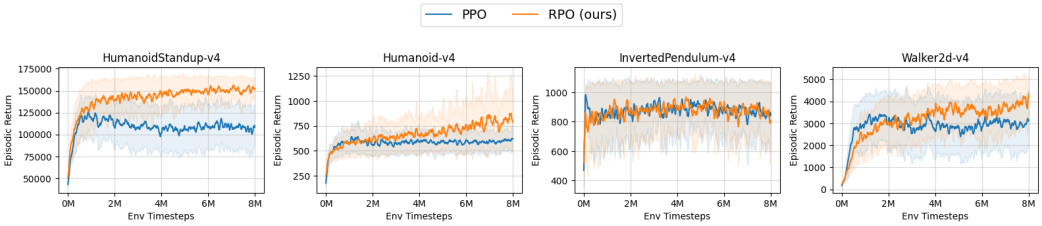

Figure 10: Results comparison of RPO with default $\alpha = 0.5$ on **Gym Mujoco-v4 environments**.

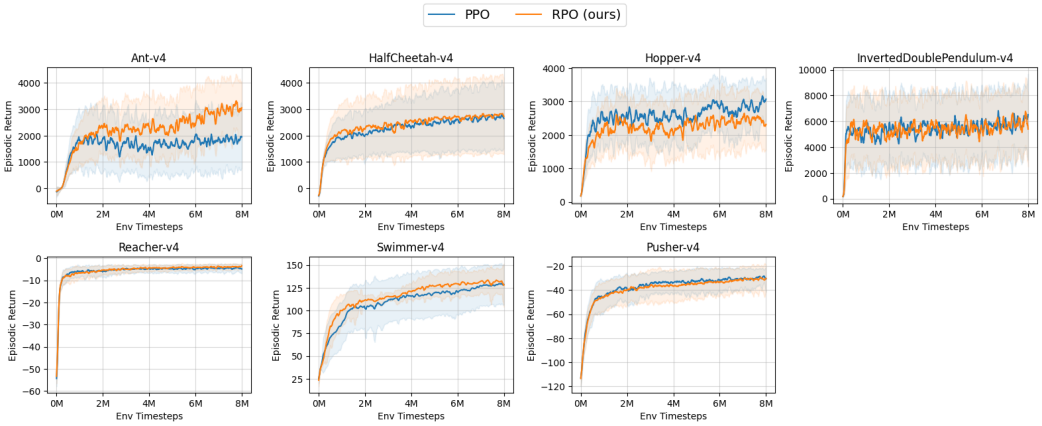

Figure 11: Results comparison of RPO with tuned $\alpha = 0.01$ on **Gym Mujoco-v4 environments**.

Additionally Figure 11 shows the result comparison with a tuned $\alpha = 0.01$ value. Overall, our method RPO improves of mathces the performance of PPO in these control tasks.

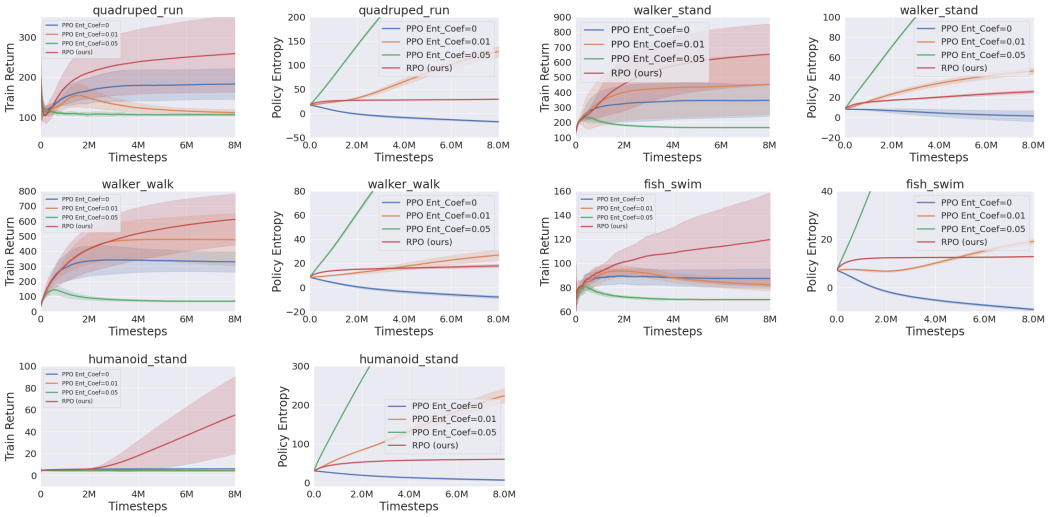

Figure 12: Results Comparison with **Entropy Regularization** on DeepMind Control.

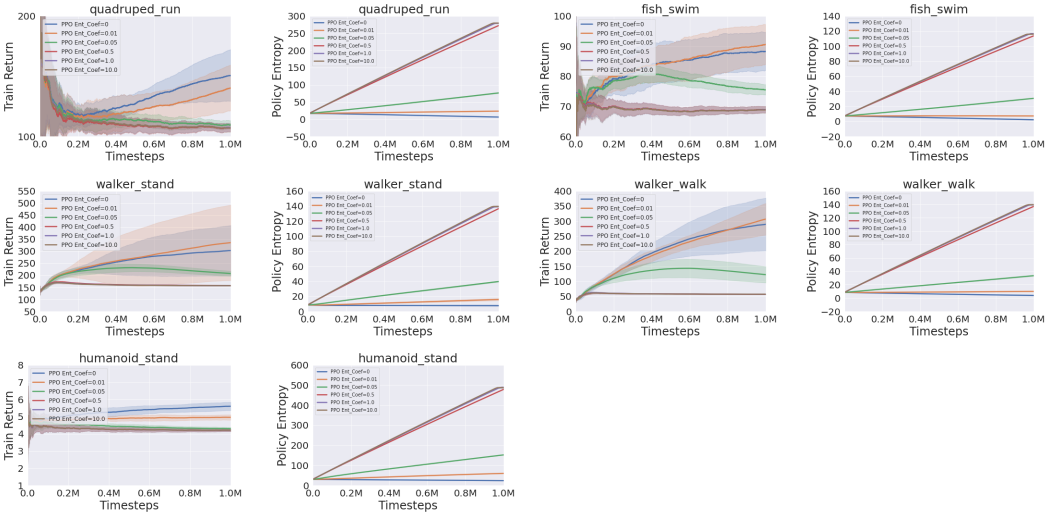

Figure 13: Comparison of return and policy entropy with **Entropy Regularization** on different (**more**) coefficient values in DeepMind Control Environments.

G Additional Results Comparison with Entropy Regularization

The results comparison of RPO with entropy coefficient are in Figure 12. More entropy coefficient results are in Figure 13.

Figure 14 shows data augmentation return curve and entropy comparison with RPO.

H Ablation Study - Effect of α

We conducted experiments on the α value ranges in the Uniform distribution. Figure 15 shows the return and policy comparison. We observe that the value of α affects the policy entropy and, thus, return performance. A smaller value of α (e.g., 0.001) seems to behave similarly to PPO, where policy entropy decreases over time, thus hampered performance. Higher entropy values, such as 1000.0, make the policy somewhat random as the uniform distribution dominates over the Gaussian distribution. This scenario keeps the entropy somewhat at a constant level; thus, the performance

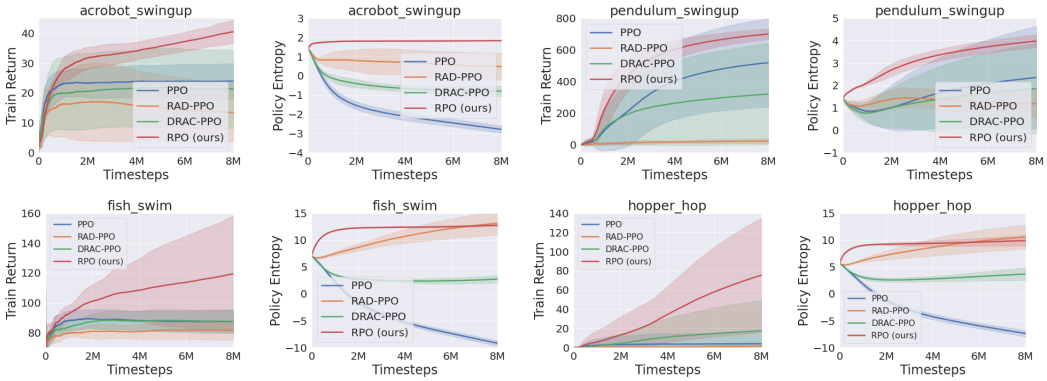

Figure 14: Comparison with PPO and **data augmentation** RAD, and DRAC on DeepMind Control.

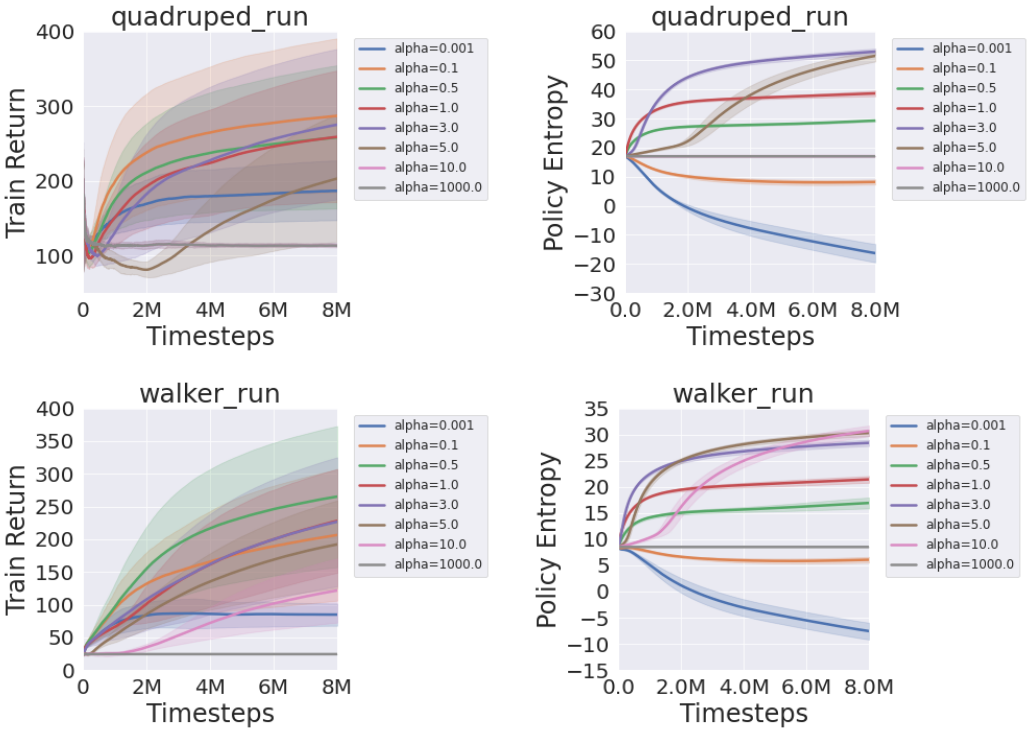

Figure 15: Ablation on $alpha$ values of the uniform distribution for RPO. An α value between 0.1 to 3 often results in better performance, while a large value often results in worse performance. The suggested default value is $\alpha = 0.5$.

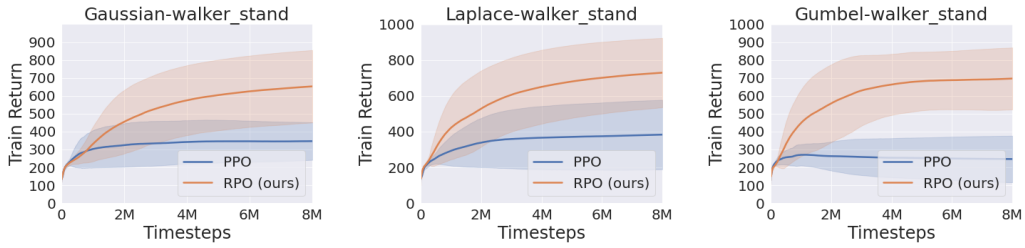

Figure 16: Results on different **action distributions**. Our method RPO shows improvement compared to base distributions. In these cases, the perturbed version of the distribution methods performs the best.

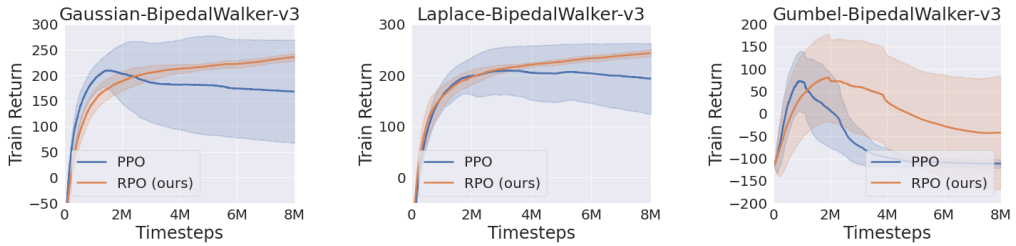

Figure 17: **Gym**: Results on different **action distributions**. Our method RPO shows improvement compared to base distributions.

is hampered. Overall, a value between 0.1 to 3 often results in better performance. Due to overall performance advantage, in this paper, we report results with $\alpha = 0.5$ for all environments.

I Comparison with Other (Laplace and Gumbel) Distributions

In this section, we compare the results of the other two distributions: Laplace and Gumbel. We show empirical evidence that the choice of action distribution is important in solving a reinforcement learning task.

Furthermore, we combine our method of adding uniform distribution with the distributions (Gaussian, Laplace, and Gumbel). The results in Figure 16 show that our method overall improved the performance compared to the base distributions. These show the implication of our method in varieties of distributions. Our method also results in higher policy entropy throughout training, potentially improving exploration (see Figure 21, 22).

All Gym Environments results on different action distributions are in Figure 17.

Results on different action distribution on PyBullet environments are in Figure 18.

Results on different action distribution on DeepMind Control environments are in Figure 19.

J Policy Entropy Comparison

Entropy Plot for DeepMind Control is in Figure 20.

Entropy Plot of Gym and Pybullet Environments for different action distributions is in Figure 21.

Entropy Plot of DeepMind Control for different action distributions is in Figure 22.

K Additional Results on RPO-alpha Annealing

Return: Figure 23 shows a comparison of more results in other environments.

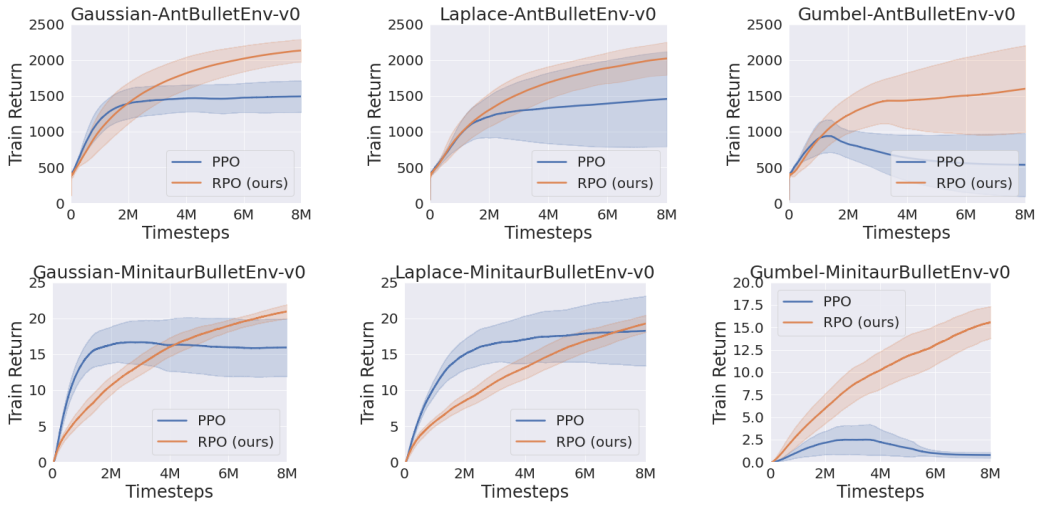

Figure 18: **PyBullet**: Results on different **action distributions**. Our method RPO shows improvement compared to base distributions.

Policy Entropy: The policy entropy is impacted by the annealing of the α value, as depicted in Figure 24. The entropy initially increases and then begins to decline again as the training progresses towards its end. This indicates that the annealing approach provides more control over how the policy entropy changes during training.

L Comparison with Data Augmentation

We observe that the **data augmentation** method can help increase the policy's entropy by often randomly perturbing observations. This process might improve the performance where higher entropy is preferred. Thus, we compare our method with two data augmentation-based methods: RAD [Laskin et al., 2020], and DRAC [Raileanu et al., 2020]. Here, The pure data augmentation baseline RAD uses data processing before passing it to the agent, and the DRAC uses data augmentation to regularize the value and policy network. Both of these data augmentation methods use PPO as their base RL policy.

Seeing the data augmentation through the lens of entropy, we observe that empirically, it can help the policy achieve a higher entropy than without data augmentation. However, this process often requires prior knowledge about the environments and a preprocessing step of the agent experience. Moreover, such methods might result in an uncontrolled increase in action entropy, eventually hampering the return performance [Raileanu et al., 2020]. The results on DeepMind Control environments are shown in Table 3. Our method performs better in mean episodic return in most environments than PPO and other data augmentation baselines RAD and DRAC.

We observe that the data augmentation slightly improved the base PPO algorithms, and the policy entropy shows higher than the base PPO. However, our method RPO still maintains a better mean return than all the baselines. The entropy of our method shows an increase at the initial timestep of the training. However, it eventually becomes stable at a particular value. The data augmentation method, especially RAD, shows an increase in entropy throughout the training process. However, this increase does not translate to the return performance. Moreover, improper handling of the data augmentation may result in worsen performance.

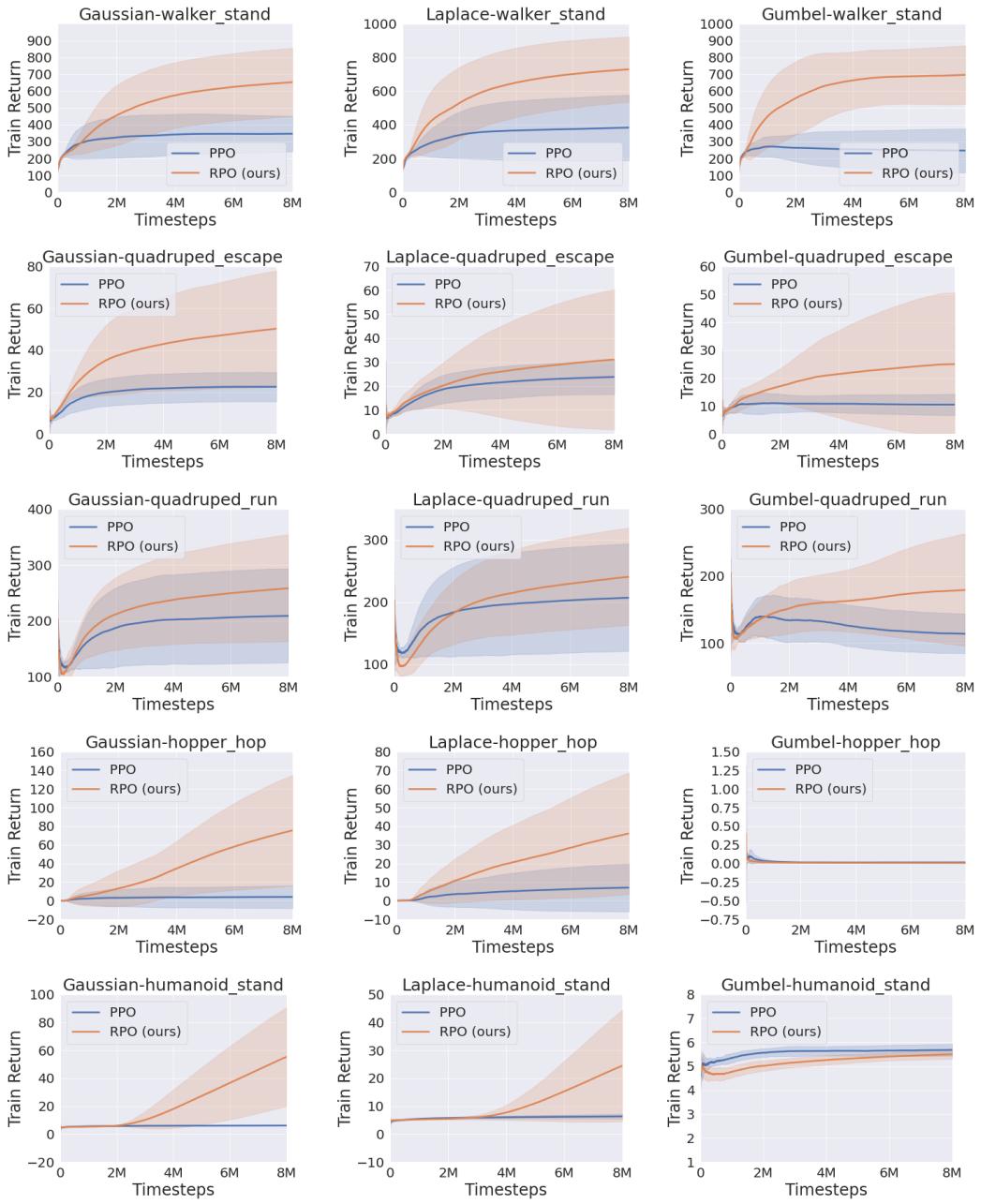

Figure 19: DeepMind Control: Results on different **action distributions**. Our method RPO shows improvement compared to base distributions in many environments.

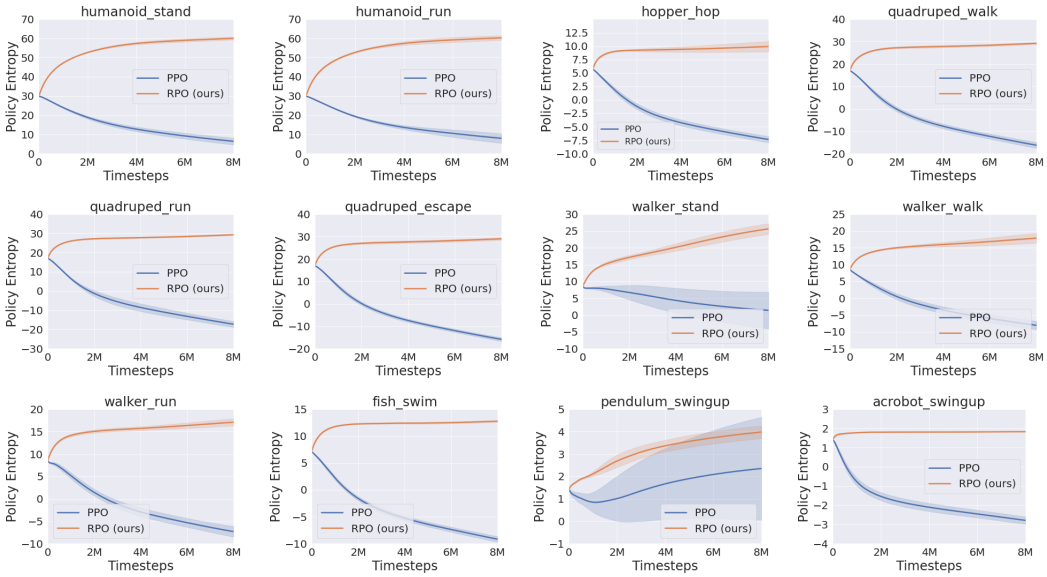

Figure 20: **Entropy** Results on **DeepMind Control Environments**.

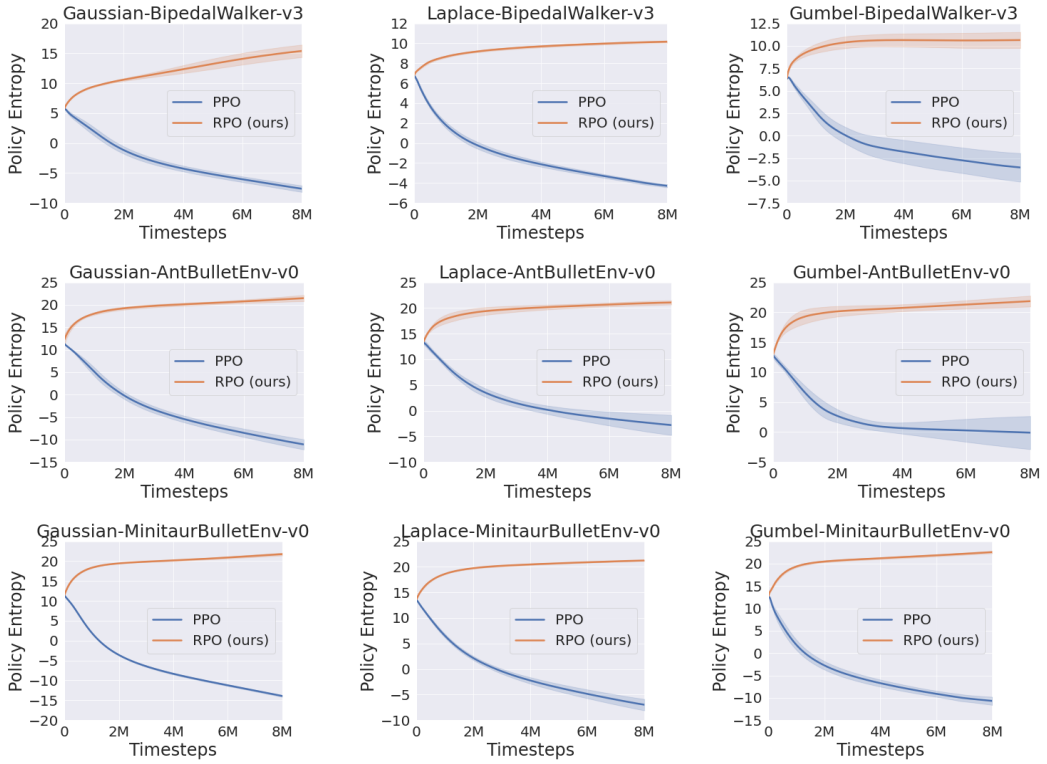

Figure 21: **Gym and Pybullet: Entropy** comparison on different action distribution. In all cases, our method of perturbing distribution with Uniform distribution (RPO) results in higher policy entropy and potentially improved exploration.

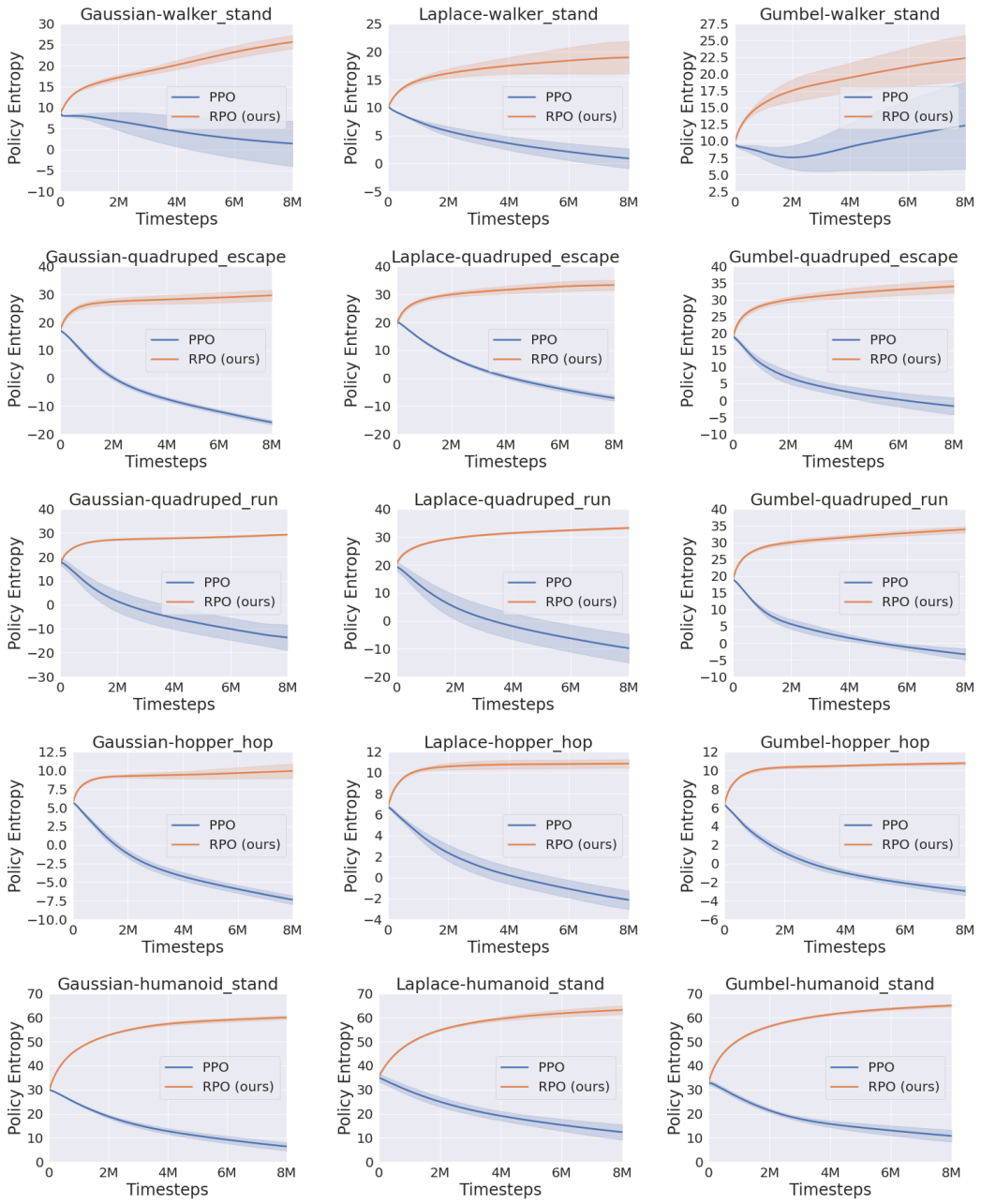

Figure 22: **DeepMind Control**: Policy **Entropy** Results on different **action distribution**.

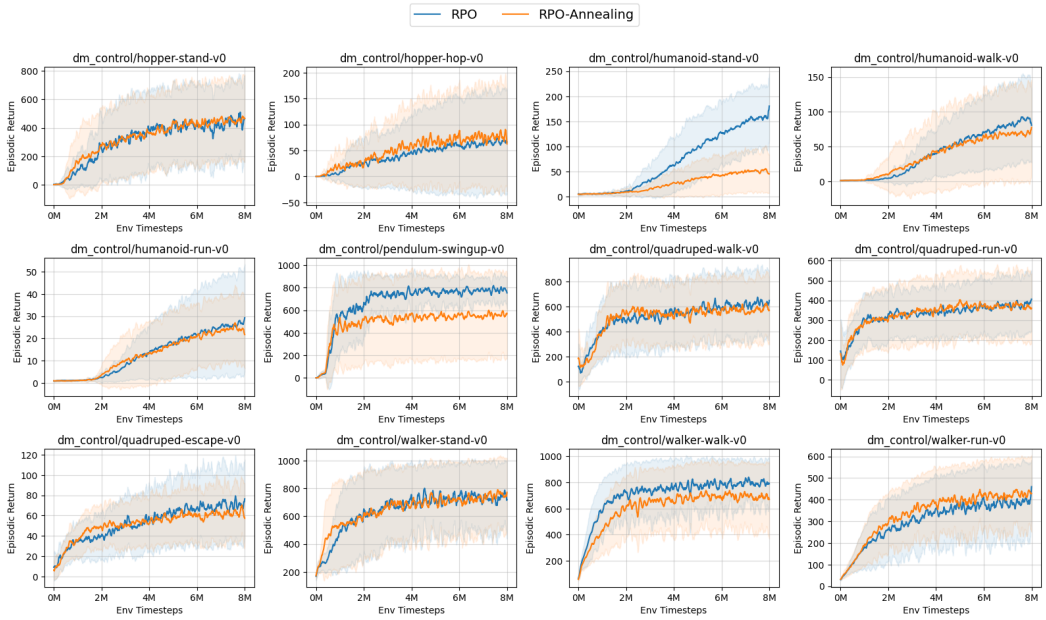

Figure 23: RPO vs **RPO-Annealing Return** Comparison.

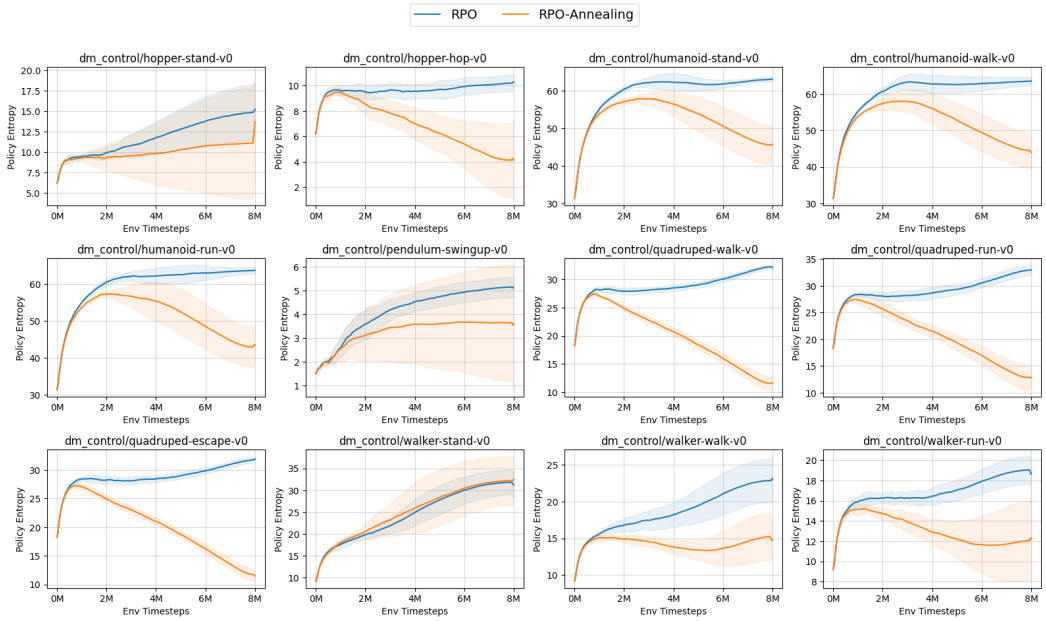

Figure 24: RPO vs **RPO-Annealing Entropy** Comparison. The entropy initially rises and then falls as training progresses, suggesting that the annealing approach offers greater control over policy entropy changes during training.

Table 3: Result comparison with **data augmentation on DeepMind control environments**. Our method RPO performs better in mean episodic return in most environments than PPO and other data augmentation baselines RAD and DRAC. Moreover, in some environments, the data augmentation baselines worsen the performance compared to the base PPO. The results are after training the agent for 8M timesteps. The mean and standard deviations are over 10 seed runs.

Env	PPO	RAD-PPO	DRAC-PPO	RPO (ours)
acrobot swingup	23.93 $\pm$6.0	13.41 $\pm$9.84	21.32 $\pm$13.17	**40.46** $\pm$4.01
fish swim	87.39 $\pm$8.0	81.53 $\pm$6.53	87.5 $\pm$7.74	**119.42** $\pm$38.46
humanoid stand	6.06 $\pm$0.17	6.14 $\pm$0.37	37.58 $\pm$66.09	**55.22** $\pm$35.34
humanoid run	1.55 $\pm$1.59	1.1 $\pm$0.04	13.13 $\pm$16.6	**13.26** $\pm$9.45
pendulum swingup	518.65 $\pm$279.73	23.04 $\pm$17.08	320.51 $\pm$319.76	**699.79** $\pm$32.97
quadruped walk	216.67 $\pm$66.56	395.07 $\pm$124.91	374.78 $\pm$197.98	**437.66** $\pm$191.93
quadruped run	183.14 $\pm$38.48	158.01 $\pm$26.84	247.02 $\pm$83.23	**258.75** $\pm$96.18
quadruped escape	22.51 $\pm$7.0	**54.0** $\pm$22.13	52.62 $\pm$21.89	53.37 $\pm$25.73
walker stand	346.55 $\pm$104.87	558.92 $\pm$131.09	361.99 $\pm$93.44	**652.7** $\pm$202.47
walker walk	329.61 $\pm$67.92	333.22 $\pm$84.38	425.51 $\pm$150.52	**611.88** $\pm$170.46
walker run	123.63 $\pm$53.5	141.43 $\pm$34.77	132.28 $\pm$37.18	**291.83** $\pm$108.8