# Performance Metrics: Comparing PPO and TRPO Across Multiple Evaluation Criteria in MuJoCo Environments

Phong Ho, Quang Hieu Pham, Yang Guo
*Department of Computing Science*
*University of Alberta*
Edmonton, Canada
{tph, quanghie, yguo19}@ualberta.ca

***Abstract*—Reinforcement learning algorithms are often evaluated primarily on final performance, overlooking important aspects such as sample efficiency, stability across random seeds, and worst-case robustness. This narrow evaluation paradigm can limit insight into algorithm behavior and provide insufficient guidance for real-world deployment. In this work, we present a comprehensive empirical study comparing Proximal Policy Optimization (PPO) and Trust Region Policy Optimization (TRPO) across five key dimensions: final policy performance, sample efficiency, consistency across seeds, worst-case robustness, and hyperparameter sensitivity. We evaluate each algorithm using 50 random seeds per environment on continuous control tasks. Our results reveal that algorithm superiority is highly environment-dependent: PPO excels in the complex Ant-v5 environment across most metrics, achieving better sample efficiency and substantially lower variability, while TRPO dominates in HalfCheetah-v5, which is a more simple environment, with superior final performance and worst-case robustness. Most critically, we demonstrate that algorithm rankings depend on the evaluation metric used, with configurations optimized for final performance often showing poor stability, and vice versa. These findings emphasize the need for multi-metric evaluation when selecting algorithms for specific deployment contexts.**

***Index Terms*—Reinforcement Learning, Policy Optimization, Empirical Evaluation, Performance Metrics, Continuous Control**

## I. Introduction

This section motivates the problem of algorithm evaluation in deep reinforcement learning, states our research questions, and summarizes our contributions.

### A. Motivation

A growing body of empirical research has revealed that the choice of evaluation metrics significantly influences conclusions about reinforcement learning (RL) algorithm superiority [1], [2]. The evaluation and comparison of RL algorithms remain challenging due to the inherent stochasticity in both learning dynamics and policy execution [3]. Algorithms that appear superior in one metric may underperform in another, yet most existing comparisons focus primarily on final performance, leaving critical questions about sample efficiency, robustness, and practical deployment unanswered.

This discrepancy between evaluation procedures and real-world deployment requirements motivates our thorough multi-metric empirical investigation. Conventional algorithm evaluations usually provide learning curves or ultimate performance for a limited number of seeds, which may not be enough to make statistically sound inferences. Furthermore, crucial features of algorithm behavior such as learning speed, stability across initialization, tolerance to worst-case scenarios, and sensitivity to hyperparameter selections are commonly overlooked. Recent work by Agarwal et al. [1] has demonstrated that proper statistical tools including stratified bootstrap confidence intervals, performance profiles, and robust aggregation metrics like the interquartile mean are essential for reliable algorithm comparison. Similarly, Chan et al. [2] showed that worst-case performance metrics such as conditional value at risk reveal critical reliability characteristics invisible in mean performance alone.

Trust Region Policy Optimization (TRPO) [4] and Proximal Policy Optimization (PPO) [5] represent two prominent approaches to policy gradient methods that aim to stabilize learning through constraint mechanisms. Despite their widespread adoption and PPO becoming one of the most widely used RL algorithms in practice, comprehensive comparisons across multiple evaluation metrics remain limited. Most existing comparisons focus on learning curves and final returns with insufficient statistical rigor, making it difficult to answer practical deployment questions.

When practitioners encounter real-world deployment constraints, algorithm selection becomes especially crucial. When robot interaction time is costly, should we prioritize sampling efficiency? When dependability guarantees are essential, should we prioritize worst-case robustness? Or should we focus solely on achieving the best possible performance? It remains challenging to provide systematic answers to these practical questions without thorough multi-metric examination. Our study addresses this gap by conducting a rigorous empirical comparison across five complementary performance dimensions using 50 random seeds per algorithm-environment pair.

### B. Research Questions

A comprehensive algorithm evaluation must answer multiple questions beyond "which algorithm achieves the highest return?" Based on our evaluation framework, we address the following research questions:

**RQ1: Quality of the final policy.** How good is the final policy after training? Which algorithm produces higher quality policies measured by mean, median, and interquartile mean?

**RQ2: Learning speed.** How fast does the agent learn? Which algorithm demonstrates better sample efficiency, achieving higher performance with fewer environment interactions?

**RQ3: Consistency between seeds.** How consistent are results across random initializations? Which algorithm shows lower median absolute deviation between runs, indicating more reliable and predictable behavior?

**RQ4: Worst-case robustness.** How robust is each algorithm to worst-case runs? Which algorithm has higher conditional value at risk, indicating better performance even in unfavorable scenarios?

**RQ5: Hyperparameter sensitivity.** How sensitive is each algorithm to hyperparameter tuning? Which algorithm is more forgiving of suboptimal configurations and requires less careful tuning for deployment?

### C. Contributions

We present a comprehensive empirical comparison of PPO and TRPO that goes beyond traditional single-metric evaluation, answering five fundamental questions about algorithm behavior. Our evaluation framework systematically assesses both algorithms across five complementary performance dimensions that address different deployment concerns. For final policy performance, we measure ultimate policy quality using mean, median, and interquartile mean from offline deterministic rollouts. For sample efficiency, we analyze learning speed through area-under-curve calculations. For stability across seeds, we quantify consistency and predictability using median absolute deviation. For worst-case robustness, we assess reliability in unfavorable scenarios using conditional value at risk. For hyperparameter sensitivity, we examine robustness to suboptimal tuning using median absolute deviation and conditional value at risk across configurations.

Our rigorous statistical methodology conducts experiments with 50 random seeds per algorithm-environment combination and 10 seeds per hyperparameter configuration, totaling 1,920 independent training runs. For each metric, we report mean with standard deviation, median, interquartile mean for robust central tendency, and full distribution visualizations. Finally, we provide evidence-based recommendations for when to prefer PPO versus TRPO based on specific deployment priorities, whether prioritizing sample efficiency, reliability, final performance, or limited tuning resources.

The remainder of this paper is organized as follows. Section II reviews background on the reinforcement learning formalism and policy gradient algorithms, followed by related work on empirical evaluation. Section III formalizes our performance metrics. Section IV details our experimental methodology. Section V presents results. Section VI discusses practical implications. Section VII concludes.

## II. Background

This section provides the technical foundations for our study, covering the reinforcement learning formalism and the algorithms under comparison.

### A. Reinforcement Learning Framework

We consider reinforcement learning in the standard Markov Decision Process (MDP) setting, defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, $P$ is the transition probability function, $R$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor.

A stochastic policy $\pi_\theta$ parameterized by $\theta$ maps states to probability distributions over actions. The goal is to find parameters $\theta^*$ that maximize the expected discounted cumulative reward:

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

where $\tau$ denotes a trajectory sampled by following policy $\pi_\theta$.

### B. Policy Gradient Algorithms: TRPO and PPO

We focus on two widely used policy gradient algorithms that constrain policy updates to ensure stable learning.

**Trust Region Policy Optimization (TRPO)** [4] addresses the challenge of step sizing in policy gradient methods by enforcing a hard constraint on the KL divergence between successive policies. The optimization problem is:

$$\max_{\theta} \quad \mathbb{E}_t \left[ \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t \right] \tag{2}$$

$$\text{s.t.} \quad \mathbb{E}_t \left[ D_{\text{KL}}(\pi_{\theta_{\text{old}}}(\cdot|s_t) \| \pi_\theta(\cdot|s_t)) \right] \leq \delta \tag{3}$$

where $\hat{A}_t$ is an estimate of the advantage function, and $\delta$ is the trust region size. TRPO solves this constrained problem using the conjugate gradient algorithm followed by a line search, providing theoretical guarantees of monotonic improvement but at significant computational cost.

**Proximal Policy Optimization (PPO)** [5] simplifies TRPO's approach by using a clipped surrogate objective:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[ \min \left( r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{4}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio, and $\epsilon$ is the clipping parameter (typically 0.2). The clipping mechanism removes the incentive for large policy changes without requiring second-order optimization, making PPO simpler to implement and computationally efficient.

For both algorithms, hyperparameters such as trust region size (TRPO), clipping parameter (PPO), optimization epochs, and value function updates can significantly affect performance, motivating systematic evaluation across configurations.

## III. Related Work

This section reviews prior work on empirical evaluation in reinforcement learning and positions our contributions.

### A. Evaluation Practices in Reinforcement Learning

Evaluation in deep reinforcement learning is challenging due to stochasticity, sensitivity to initialization, and hyperparameter dependence [3]. Common practices that report only final performance or single-seed results can yield misleading conclusions. Henderson et al. [3] documented widespread issues with reproducibility and evaluation practices, including insufficient seeds, improper confidence intervals, and selective reporting. Engstrom et al. [6] demonstrated that implementation details often have larger effects on performance than algorithmic differences.

Recent work has proposed more rigorous statistical tools to address these challenges. Agarwal et al. [1] recommended stratified bootstrap confidence intervals, performance profiles using CDFs, and the interquartile mean (IQM) for robust aggregation. The IQM computes the mean of the middle 50% of scores, providing robustness to outliers while maintaining higher statistical efficiency than the median. Statistical efficiency refers to the property that IQM makes better use of available data than the median, producing more precise estimates with the same sample size by incorporating information from half the distribution rather than just the center point. Chan et al. [2] proposed using conditional value at risk (CVaR) to measure worst-case performance and reliability in RL algorithms. CVaR captures the expected performance in the tail of the distribution, which is critical for safety-sensitive applications. For measuring consistency and variability, the median absolute deviation (MAD) [7] provides a robust alternative to standard deviation that is less sensitive to outliers.

### B. Comparisons of PPO and TRPO

Despite these methodological advances, most empirical comparisons of PPO and TRPO remain focused on learning curves and final returns with limited statistical rigor. The original PPO paper [5] demonstrated competitive or superior performance compared to TRPO on several continuous control tasks, but primarily reported mean performance across a small number of seeds. Subsequent applications have widely adopted PPO due to its simplicity and empirical success, yet comprehensive multi-metric comparisons using rigorous statistical methodology remain limited in the literature.

Our work addresses this gap by performing a multi-metric evaluation across 50 seeds per algorithm-environment pair, reporting mean, median, IQM, MAD, and CVaR to provide a comprehensive view of algorithm behavior. We demonstrate that algorithm rankings depend critically on the evaluation metric used, with implications for algorithm selection in different deployment contexts.

## IV. Performance Metrics and Evaluation Framework

We evaluate each algorithm over $N$ independent training runs (seeds) in a fixed environment. Let $R_{j,t}$ denote the episodic return for seed $j$ at evaluation index $t$, and $G_j$ denote a scalar summary of that run. Aggregated metrics are computed across the set $\{G_1, \ldots, G_N\}$ to capture both central tendencies and variability in performance.

### A. Final Policy Performance

Final policy performance captures the overall quality of the learned policy after training. For each seed, we compute the average return over the last $K$ evaluation steps:

$$G_j^{\text{final}} = \frac{1}{K} \sum_{t=T-K+1}^{T} R_{j,t}. \tag{5}$$

We summarize these results across seeds using three complementary statistics:

**Mean.** The arithmetic average of final returns across seeds provides a simple central tendency but can be sensitive to extreme outliers.

**Median.** The 50th percentile of the final returns is robust to outliers and represents a typical run more reliably than the mean.

**Interquartile Mean (IQM).** Following Agarwal et al. [1], we compute the mean of the middle 50% of final returns. IQM balances robustness and informativeness, mitigating the influence of very poor or very good runs while maintaining higher statistical efficiency than the median.

Together, these statistics provide a comprehensive view of policy quality. While the mean highlights overall performance, the median and IQM reveal how representative that performance is across runs.

### B. Sample Efficiency

Sample efficiency measures how quickly an algorithm learns to achieve high performance. At each evaluation index $t$, we compute the IQM across seeds, producing a robust learning curve. To summarize learning speed as a single number, we compute the area under the IQM curve (AUC):

$$\text{AUC} = \sum_{t=1}^{T} \text{IQM}(R_{\cdot,t}) \cdot \Delta t. \tag{6}$$

Higher AUC indicates that an algorithm reaches strong performance with fewer environment interactions, which is especially important when real-world samples are costly.

### C. Stability Across Seeds

To quantify consistency across different random initializations, we use the median absolute deviation (MAD) [7]:

$$\text{MAD} = \text{median}\big(|G_j^{\text{final}} - \text{median}(G_j^{\text{final}})|\big). \tag{7}$$

Lower MAD indicates that results are reproducible and less dependent on random factors such as initialization or stochastic environment dynamics. High stability is critical for reliable deployment. MAD is preferred over standard deviation because it is robust to outliers, providing a more reliable measure of typical variability.

### D. Worst-Case Robustness

Robustness captures how well an algorithm avoids catastrophic failures. Following Chan et al. [2], we use conditional value at risk (CVaR) at quantile $\alpha$, defined as the expected performance of the worst $\alpha$ fraction of runs:

$$\text{CVaR}_\alpha = \mathbb{E}\big[G_j^{\text{final}} \mid G_j^{\text{final}} \leq \text{VaR}_\alpha\big], \tag{8}$$

where $\text{VaR}_\alpha$ is the $\alpha$-quantile of final returns. We use $\alpha = 0.1$ (bottom 10% of runs) in our experiments.

Higher CVaR means the algorithm performs better even in unfavorable scenarios. This metric is critical in safety-critical applications where occasional failures can be costly.

### E. Hyperparameter Sensitivity

Hyperparameter sensitivity measures how performance varies across different algorithm configurations. Let $H_c$ denote a central performance statistic (IQM across seeds) for configuration $c$. We compute:

$$\text{MAD}_{\text{HP}} = \text{median}\big(|H_c - \text{median}(H_c)|\big), \tag{9}$$

$$\text{CVaR}_{\alpha,\text{HP}} = \mathbb{E}\big[H_c \mid H_c \leq \text{VaR}_\alpha(\{H_c\})\big]. \tag{10}$$

Smaller $\text{MAD}_{\text{HP}}$ indicates stable performance across hyperparameter settings, reducing the need for careful tuning. Higher $\text{CVaR}_{\alpha,\text{HP}}$ suggests that even worst-performing configurations remain acceptable. These metrics inform how forgiving an algorithm is to suboptimal hyperparameter choices.

## V. Experimental Setup

This section describes our experimental methodology, including environments, implementations, hyperparameters, and evaluation protocols.

### A. Environments

We benchmark PPO and TRPO on two MuJoCo environments: HalfCheetah-v5 and Ant-v5 [8]. These environments were selected to represent distinct levels of complexity in continuous control. HalfCheetah-v5 simulates a planar 2D robot, whereas Ant involves a more complex 3D quadrupedal structure. Consequently, the Ant-v5 environment entails a significantly larger problem space, with a higher-dimensional observation space ($|\mathcal{S}| = 105$ versus $|\mathcal{S}| = 17$) and action space ($|\mathcal{A}| = 8$ versus $|\mathcal{A}| = 6$) compared to HalfCheetah.

To maintain stability during training, we normalize both input features and feedback signals. For the observation space, the system maintains a running estimate of the mean and standard deviation based on the history of all visited states. Each raw observation is then standardized by subtracting this running mean and dividing by the standard deviation before being passed to the policy. Simultaneously, we normalize the reward signal to ensure consistent scale across different environments. Unlike observations, rewards are only divided by a running estimate of their standard deviation without mean subtraction. This omission is intentional to preserve the underlying shift of the reward distribution, which is essential for the accurate calculation of cumulative returns.

TABLE I: Hyperparameter configurations for PPO and TRPO. The sweep covers 16 combinations for each algorithm.

| Algorithm | Parameter | Symbol | Value(s) |
|---|---|---|---|
| **PPO** | *Fixed Parameters* | | |
| | Sample Size | $T$ | 2048 |
| | Mini batch size | $b$ | 64 |
| | Policy Learning Rate | $\alpha$ | $3 \times 10^{-4}$ |
| | Value Function Learning Rate | $\alpha_{value}$ | $1 \times 10^{-3}$ |
| | *Grid Search (Sweep)* | | |
| | Clipping Parameter | $\epsilon$ | $\{0.2, 0.3\}$ |
| | Target KL Divergence | $d_{\text{targ}}$ | $\{0.01, 0.02\}$ |
| | Policy Update Iterations | $K_\pi$ | $\{10, 20\}$ |
| | Value Function Iterations | $K_v$ | $\{10, 20\}$ |
| **TRPO** | *Fixed Parameters* | | |
| | Sample Size | $T$ | 2048 |
| | Mini batch size (for value function) | $b$ | 64 |
| | Value Function Learning Rate | $\alpha_{value}$ | $1 \times 10^{-3}$ |
| | *Grid Search (Sweep)* | | |
| | Trust Region Size | $\delta$ | $\{0.01, 0.02\}$ |
| | Backtracking Coefficient | $\beta$ | $\{1.0, 0.8\}$ |
| | Value Function Iterations | $K_v$ | $\{10, 20, 80, 120\}$ |

### B. Implementation Details

We implement PPO and TRPO using 2-layer neural network architectures with 64 hidden dimensions and tanh activation for both actors and critics (value functions). The policy is parameterized as a diagonal Gaussian distribution $\mathcal{N}(\mu(s_t), \sigma)$, where the mean $\mu(s_t)$ is predicted by the neural network and the standard deviation $\sigma$ is maintained as a separate, state-independent learnable parameter vector. The log standard deviation parameter is initialized to $-0.5$ times the action limit to ensure appropriate initial exploration variance. Sampled actions are clamped within the $[-1, 1]$ range to adhere to environment constraints.

### C. Hyperparameter Configurations

To ensure a fair and comprehensive comparison, we conduct a hyperparameter sweep over 16 distinct configurations for each algorithm. Table I summarizes our hyperparameter values for grid search.

For PPO, we specifically tune the clipping parameter ($\epsilon$), which constrains the policy update, and the target KL divergence threshold ($d_{targ}$) used for early stopping. Additionally, we vary the number of training iterations for both the policy ($K_\pi$) and value function ($K_v$) networks.

For TRPO, the grid search focuses on the trust region size ($\delta$), the backtracking coefficient (governing line-search acceptance), and the number of value function update iterations ($K_v$). Since TRPO relies on large batch updates for the policy ($T = 2048$) to accurately estimate the Fisher Information Matrix via Conjugate Gradient, we hypothesize that the accuracy of the value function is critical for update stability. Therefore, we sweep over substantially larger values for $K_v$ (up to 120), to observe if training with more steps reduces variance in advantage estimation, thereby improving the efficacy and stability of the Conjugate Gradient step.

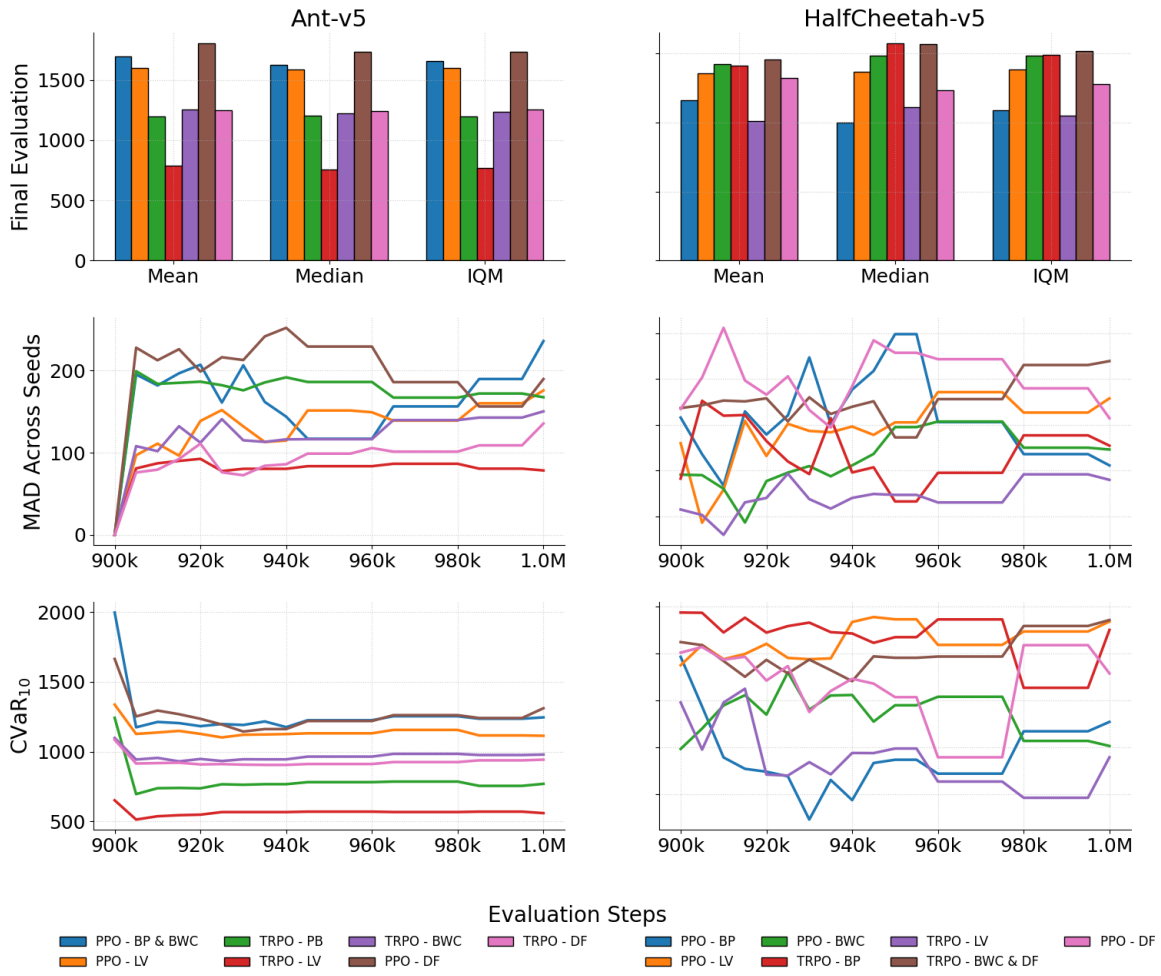


Fig. 1: Comparison of PPO and TRPO's performance metrics on two different MuJoCo environments: Ant-v5 (Left) and HalfCheetah-v5 (Right). Each row represents different types of metrics: final evaluation, MAD across seeds, and CVaR respectively. Each environment had different parameter settings for each algorithm. BP = Best Performance, LV = Least Variability, BWC = Best Worst Case, DF = Default.

### *D. Evaluation Protocol*

Our experimental framework is designed to ensure statistical significance and reproducibility. Each algorithm is trained for a total of 1 million agent steps. The evaluation process is divided into two phases:

1) **Tuning Phase:** Each of the 16 hyperparameter configurations is evaluated using 10 random seeds to identify the optimal settings.
2) **Comparison Phase:** The best-performing configurations are subsequently re-evaluated using 50 distinct random seeds to estimate the expected performance and variance.

In total, our study includes 1,920 independent training runs. To mitigate the effects of training instability, besides logging online training return, we log the average return calculated from offline rollouts over 100 episodes. These rollouts are collected every 2,000 steps during the final 100,000 steps of training, providing a robust measure of the converged policy's quality.

## VI. Results

This section presents our experimental results organized by environment, followed by cross-environment analysis.

### *A. Hyperparameter Selection*

After performing a full hyperparameter sweep for each environment and each algorithm, we identify the configuration that achieves the best score under three complementary evaluation criteria. **BP (Best Performance)** denotes the configuration that maximizes the average final-return metric, representing the strongest asymptotic policy performance. **LV (Least Variability)** is the configuration with the smallest MAD across seeds, capturing the most stable and consistent learning dynamics. **BWC (Best Worst Case)** corresponds to the configuration that achieves the highest CVaR with $\alpha = 0.3$, emphasizing robustness by improving worst-case performance in the return distribution.

TABLE II: Summary of Best Hyperparameter Configurations

| Env. | Alg. | Criteria | Configuration |
|---|---|---|---|
| Ant-v5 | PPO | BP | $\epsilon = 0.2, d_{\text{targ}} = 0.02, K_\pi = 20, K_v = 10$ |
| | | LV | $\epsilon = 0.3, d_{\text{targ}} = 0.01, K_\pi = 10, K_v = 20$ |
| | | BWC | $\epsilon = 0.2, d_{\text{targ}} = 0.02, K_\pi = 20, K_v = 10$ |
| | TRPO | BP | $\delta = 0.02, \beta = 1.0, K_v = 10$ |
| | | LV | $\delta = 0.01, \beta = 1.0, K_v = 80$ |
| | | BWC | $\delta = 0.02, \beta = 0.8, K_v = 10$ |
| HalfCheetah -v5 | PPO | BP | $\epsilon = 0.3, d_{\text{targ}} = 0.02, K_\pi = 20, K_v = 20$ |
| | | LV | $\epsilon = 0.3, d_{\text{targ}} = 0.01, K_\pi = 10, K_v = 10$ |
| | | BWC | $\epsilon = 0.3, d_{\text{targ}} = 0.01, K_\pi = 20, K_v = 10$ |
| | TRPO | BP | $\delta = 0.02, \beta = 1.0, K_v = 10$ |
| | | LV | $\delta = 0.02, \beta = 0.8, K_v = 120$ |
| | | BWC | $\delta = 0.01, \beta = 1.0, K_v = 10$ |

TABLE III: AUC of Online IQM Learning Curves (Fig. 4) for PPO and TRPO Across Both Environments

| Method | Criteria | Ant-v5 (AUC) | HalfCheetah-v5 (AUC) |
|---|---|---|---|
| PPO | BP | $1.62 \times 10^8$ | $9.11 \times 10^8$ |
| | BWC | $1.62 \times 10^8$ | $7.78 \times 10^8$ |
| | DF | $1.16 \times 10^8$ | $9.41 \times 10^8$ |
| | LV | $4.28 \times 10^7$ | $7.36 \times 10^8$ |
| TRPO | BP | $3.97 \times 10^7$ | $1.00 \times 10^9$ |
| | BWC | $3.48 \times 10^7$ | $8.02 \times 10^8$ |
| | DF | $3.11 \times 10^7$ | $8.02 \times 10^8$ |
| | LV | $-1.25 \times 10^7$ | $2.90 \times 10^8$ |

We include **DF (Default)** configurations taken directly from the Deep RL GitHub Repository [1], which corresponds to the standard parameter settings of PPO and TRPO. These defaults serve as a practical reference point for evaluating how much performance, stability, and robustness can be improved through targeted hyperparameter tuning.

The complete hyperparameter sweep results and performance curves are provided in the Appendix, while Table II summarizes the selected configurations. These configurations are then compared to each other to characterize differences in asymptotic performance, variability across seeds, and worst-case robustness.

### *B. Ant Environment*

The Ant-v5 environment reveals clear distinctions between PPO and TRPO across the performance, efficiency, and robustness metrics described in Figure 1 and Table III. Across all three final evaluation statistics, PPO configurations consistently achieve higher returns than their TRPO counterparts. Among these, the PPO-DF attains the highest overall performance, aligning with the findings reported in the original PPO paper. However, although PPO–DF produces the strongest final policies, its variability across seeds is notably high, indicating that the configuration is not the most stable choice. Instead, PPO–LV exhibits substantially lower dispersion in final returns, reflecting more reproducible behavior and confirming its role as the least-variable PPO configuration.

[1]Deep RL GitHub Repository

The sample-efficiency results corroborate this trend. PPO configurations generally achieve higher AUC values than TRPO, with PPO–BP & BWC obtaining the largest AUC in Ant-v5 and PPO–DF also performing strongly. In contrast, TRPO configurations exhibit lower overall AUC, and in some cases negative AUC values indicate substantial degradation in early or mid-training performance. These observations suggest that PPO not only reaches high performance but does so reliably with fewer environment interactions.

Stability and robustness metrics provide further insight into the behavior of each configuration. MAD curves show that PPO–LV and PPO–DF maintain the lowest variability across seeds, whereas TRPO–BWC and TRPO–DF exhibit substantially higher dispersion, often exceeding MAD values of 200 during the final stages of training. Worst-case robustness is assessed through $\text{CVaR}_{0.1}$ and highlights strengths of the seeds. PPO–DF remains competitive even in the lowest-performing 10% of runs, and PPO–BP & BWC performs nearly as well, indicating resilience to high-risk failures. Within TRPO, the TRPO–BWC configuration obtains the strongest CVaR values, outperforming the other TRPO variants despite not achieving the highest central-performance metrics. This suggests that TRPO–BWC provides the most reliable behavior under adverse conditions, even though TRPO–DF is the best-performing configuration on average.

### *C. HalfCheetah Environment*

In contrast to Ant-v5, the HalfCheetah-v5 environment shows a reversal in relative performance between the two algorithms. As illustrated in Figure 1, TRPO achieves superior final policy performance across all three summary metrics. Both TRPO-DF and TRPO-BP obtain higher mean, median, and IQM returns than any of the PPO configurations, indicating that TRPO is able to exploit the smoother, more predictable dynamics of HalfCheetah to learn better policies.

The sample-efficiency comparison is broadly consistent with these observations. Table III shows that TRPO configurations achieve some of the largest AUC values in HalfCheetah-v5, with TRPO-BP and TRPO-DF performing particularly well. PPO-DF and PPO-BP also attain high AUC, but do not clearly dominate TRPO as in Ant-v5. These findings suggest that in this environment TRPO is able to combine strong asymptotic performance with competitive sample efficiency.

Stability and robustness metrics provide additional nuance. The MAD curves reveal that TRPO-LV achieves the lowest variability across seeds and is the most stable configuration overall, whereas other TRPO settings such as TRPO-BP and TRPO-BWC exhibit much larger fluctuations throughout training. PPO-LV and PPO-DF maintain moderate dispersion but do not reach the same level of stability as TRPO-LV. Thus, while TRPO attains the highest peak performance, this reliability is strongly configuration-dependent: only carefully chosen settings such as TRPO-LV combine strong returns with low across-seed variability.

Worst-case robustness, assessed through $\text{CVaR}_{0.1}$, largely mirrors the ranking in final performance. TRPO configura-

tions, particularly TRPO-BP, TRPO-BWC, and TRPO-DF, attain higher CVaR values than all PPO variants, indicating that even their worst-performing runs tend to outperform the tails of PPO. Consequently, in HalfCheetah-v5, TRPO dominates not only in average performance but also in worst-case returns, whereas PPO configurations are comparatively more prone to low-return failures.

Overall, HalfCheetah-v5 demonstrates a different regime from Ant-v5: TRPO offers the highest final returns and strongest tail performance, and with the TRPO-LV configuration it can also achieve very low across-seed variability. PPO remains competitive but does not match the best TRPO configurations in this environment.

## VII. Discussion

This section synthesizes the empirical findings across both environments and addresses the research questions posed in the introduction. Together, these results highlight not only the differences between PPO and TRPO, but also the importance of evaluating reinforcement learning algorithms using a diverse set of metrics.

### A. Algorithm Comparison Summary

**RQ1 (Final Policy Quality).** Final evaluation metrics show that the relative performance of PPO and TRPO depends strongly on the environment. In Ant-v5, PPO-DF achieves the highest final returns, with all PPO configurations outperforming their TRPO counterparts. In contrast, HalfCheetah-v5 exhibits the opposite pattern: TRPO-DF and TRPO-BWC obtain the highest mean, median, and IQM scores, outperforming all PPO configurations. These results demonstrate that no single algorithm consistently dominates in raw performance across environments.

**RQ2 (Learning Speed).** Sample efficiency, measured through AUC, largely mirrors these trends. PPO achieves higher AUC values in Ant-v5, reflecting faster and more reliable learning. In HalfCheetah-v5, TRPO produces competitive and in some cases superior AUC values, particularly in the TRPO-BP and TRPO-DF configurations. Thus, both algorithms can exhibit strong sample efficiency, but their relative advantage is environment-dependent.

**RQ3 (Consistency Across Seeds).** Stability results, as quantified by MAD, show clearer differences between the algorithms. In Ant-v5, PPO-LV and PPO-DF exhibit substantially lower variability than any TRPO configuration. In HalfCheetah-v5, TRPO-LV achieves the lowest MAD and is the most stable configuration overall, although other TRPO settings show high dispersion. These findings indicate that stability is highly configuration-dependent and cannot be inferred directly from final returns.

**RQ4 (Worst-Case Robustness).** Worst-case performance, measured by $\mathrm{CVaR}_{0.1}$, reveals additional trade-offs. In Ant-v5, PPO-DF and PPO-BP & BWC produce the strongest tail-performance, outperforming all TRPO configurations. In HalfCheetah-v5, however, TRPO-BP, TRPO-BWC, and TRPO-DF achieve higher CVaR scores than PPO, indicating that TRPO can produce more reliable worst-case returns in this environment. These results underscore that robustness is not solely a consequence of mean performance.

**RQ5 (Hyperparameter Sensitivity).** The hyperparameter sweeps reveal notable differences in sensitivity. PPO demonstrates broader regions of stable performance in Ant-v5, whereas TRPO requires more careful tuning to avoid large variability or instability. In HalfCheetah-v5, however, TRPO-LV emerges as both stable and high-performing, showing that TRPO can achieve strong reliability when hyperparameters are chosen appropriately. Overall, neither algorithm is universally more forgiving; robustness to suboptimal tuning is both environment- and configuration-specific.

### B. Broader Implications for RL Evaluation

A key takeaway from our study is that evaluating algorithms solely on final performance can lead to misleading conclusions. Many configurations that achieve the highest returns perform poorly in terms of stability or worst-case robustness, while configurations with strong reliability may not achieve the highest asymptotic performance. This multi-metric perspective highlights the inherent trade-offs in reinforcement learning: an algorithm may excel in one metric while failing dramatically in another.

Our findings therefore emphasize the importance of using a comprehensive evaluation framework by incorporating central performance, learning speed, across-seed variability, and worst-case robustness when comparing RL algorithms. Such multi-dimensional analysis not only provides a more complete understanding of algorithmic behavior but also guides practitioners toward configurations that better match their operational requirements, whether prioritizing raw performance, consistency, or safety.

### C. Limitations

Our study has the following limitations:

**Limited environments.** We evaluate on only two MuJoCo environments. Results may not generalize to other domains such as Atari, robotic manipulation, or real-world systems.

**Hyperparameter configurations.** While we conduct a comprehensive 16-configuration sweep per algorithm, the main 50-seed comparison focuses on best-performing configurations. Extensive per-environment tuning across different hyperparameter ranges might reveal additional insights.

**Single implementation.** We use a single popular open-source implementation[2]. Implementation details can significantly affect performance [6], and results might differ with other implementations.

**Continuous control only.** Both environments involve continuous action spaces. Conclusions may differ for discrete action spaces.

**Fixed training budget.** Training for 1 million steps may be insufficient for some configurations to reach asymptotic performance, particularly for TRPO configurations with high value function iteration counts.

[2]Deep RL GitHub Repository

## VIII. Conclusion

We presented a comprehensive empirical comparison of PPO and TRPO across five evaluation dimensions: final policy performance, sample efficiency, stability across seeds, worst-case robustness, and hyperparameter sensitivity. Using 50 random seeds per algorithm-environment pair and rigorous statistical methodology following Agarwal et al. [1], we provided a more complete picture of algorithm behavior than traditional single-metric evaluations.

### A. Summary of Findings

Our study reveals that algorithm superiority is highly environment-dependent and metric-specific. In Ant-v5, PPO consistently outperforms TRPO across most metrics: PPO-DF achieves higher final performance, PPO demonstrates better sample efficiency, and PPO configurations exhibit substantially lower variability across seeds. In contrast, HalfCheetah-v5 shows the opposite pattern: TRPO-DF and TRPO-BP achieve higher final returns, TRPO-BP attains superior sample efficiency, and TRPO-LV provides the lowest variability overall.

Most importantly, our results demonstrate that algorithm rankings depend critically on the evaluation metric used. Configurations optimized for final performance (BP) often show high variability across seeds, while configurations with the lowest variability (LV) may not achieve the highest returns. Similarly, worst-case robustness (BWC) does not always correlate with mean performance. This multi-dimensional perspective reveals inherent trade-offs in reinforcement learning that are invisible when evaluating algorithms on a single metric.

The hyperparameter sensitivity analysis shows that PPO is generally more forgiving to suboptimal tuning in complex environments like Ant-v5, while TRPO requires more careful configuration but can achieve excellent stability when properly tuned (as demonstrated by TRPO-LV in HalfCheetah-v5). Neither algorithm uniformly dominates across all deployment contexts, emphasizing the need for practitioners to select algorithms based on their specific priorities and environmental characteristics.

While our study provides insights into algorithm behavior in HalfCheetah and Ant environments, systematically evaluating whether these conclusions generalize across different locomotion tasks remains important future work. High-dimensional tasks such as Humanoid, sparse-reward settings, and domains outside MuJoCo such as robotic manipulation present different challenges that may reveal whether our findings about sample efficiency, robustness, and hyperparameter sensitivity hold more broadly. Investigating environment-specific factors that influence relative algorithm performance would provide valuable guidance for algorithm selection.

Second, investigating sensitivity to additional hyperparameters, particularly network architecture and GAE parameters, would provide a more complete picture of each algorithm's robustness to configuration choices.

Third, applying the same multi-metric evaluation methodology to other algorithm comparisons would demonstrate the broader utility of our framework. Comparisons between on-policy and off-policy methods or between model-free and model-based approaches would benefit from similar rigorous analysis.

Finally, examining computational cost (wall-clock time) alongside sample efficiency would provide practical guidance for deployment, as TRPO's second-order optimization incurs additional overhead that may matter in time-constrained settings.


## Acknowledgments

This work was conducted as part of CMPUT 655 at the University of Alberta. We thank the course instructors for their guidance on experimental methodology and scientific writing.


## References


[1] R. Agarwal, M. Schwarzer, P. S. Castro, A. C. Courville, and M. Bellemare, "Deep reinforcement learning at the edge of the statistical precipice," in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 29 304–29 320.

[2] S. C. Y. Chan, S. Fishman, J. Canny, A. Korattikara, and S. Guadarrama, "Measuring the reliability of reinforcement learning algorithms," *arXiv preprint arXiv:1912.05663*, 2020.

[3] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *AAAI Conference on Artificial Intelligence*, 2018.

[4] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*. PMLR, 2015, pp. 1889–1897.

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," in *arXiv preprint arXiv:1707.06347*, 2017.

[6] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janber, and A. Madry, "Implementation matters in deep policy gradients: A case study on ppo and trpo," in *International Conference on Learning Representations*, 2020.

[7] C. Leys, C. Ley, O. Klein, P. Bernard, and L. Licata, "Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median," *Journal of Experimental Social Psychology*, vol. 49, no. 4, pp. 764–766, 2013.

[8] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.


## Appendix

This appendix provides supplementary tables and figures supporting the main analysis. Table IV provides the complete mapping of configuration IDs to their hyperparameter settings for reference. Figures 2 and 3 show the hyperparameter sensitivity analysis for PPO and TRPO respectively. Figure 4 displays IQM learning curves across all training steps. **Please see following pages.**

TABLE IV: PPO and TRPO Hyperparameter Configuration ID Mapping

| PPO ID | PPO Configuration | TRPO ID | TRPO Configuration |
|---|---|---|---|
| P1 | $\epsilon = 0.3, d_{\text{targ}} = 0.01, K_\pi = 20, K_v = 20$ | T1 | $\delta = 0.01, \beta = 0.8, K_v = 120$ |
| P2 | $\epsilon = 0.3, d_{\text{targ}} = 0.01, K_\pi = 10, K_v = 20$ | T2 | $\delta = 0.02, \beta = 0.8, K_v = 120$ |
| P3 | $\epsilon = 0.2, d_{\text{targ}} = 0.01, K_\pi = 20, K_v = 20$ | T3 | $\delta = 0.01, \beta = 1.0, K_v = 120$ |
| P4 | $\epsilon = 0.2, d_{\text{targ}} = 0.01, K_\pi = 10, K_v = 20$ | T4 | $\delta = 0.02, \beta = 1.0, K_v = 120$ |
| P5 | $\epsilon = 0.3, d_{\text{targ}} = 0.01, K_\pi = 10, K_v = 10$ | T5 | $\delta = 0.01, \beta = 0.8, K_v = 80$ |
| P6 | $\epsilon = 0.3, d_{\text{targ}} = 0.02, K_\pi = 20, K_v = 20$ | T6 | $\delta = 0.02, \beta = 1.0, K_v = 80$ |
| P7 | $\epsilon = 0.3, d_{\text{targ}} = 0.01, K_\pi = 20, K_v = 10$ | T7 | $\delta = 0.01, \beta = 1.0, K_v = 80$ |
| P8 | $\epsilon = 0.2, d_{\text{targ}} = 0.01, K_\pi = 20, K_v = 10$ | T8 | $\delta = 0.02, \beta = 0.8, K_v = 80$ |
| P9 | $\epsilon = 0.3, d_{\text{targ}} = 0.02, K_\pi = 10, K_v = 20$ | T9 | $\delta = 0.01, \beta = 0.8, K_v = 20$ |
| P10 | $\epsilon = 0.2, d_{\text{targ}} = 0.01, K_\pi = 10, K_v = 10$ | T10 | $\delta = 0.02, \beta = 0.8, K_v = 20$ |
| P11 | $\epsilon = 0.2, d_{\text{targ}} = 0.02, K_\pi = 10, K_v = 20$ | T11 | $\delta = 0.01, \beta = 1.0, K_v = 20$ |
| P12 | $\epsilon = 0.2, d_{\text{targ}} = 0.02, K_\pi = 20, K_v = 20$ | T12 | $\delta = 0.02, \beta = 1.0, K_v = 20$ |
| P13 | $\epsilon = 0.3, d_{\text{targ}} = 0.02, K_\pi = 20, K_v = 10$ | T13 | $\delta = 0.01, \beta = 0.8, K_v = 10$ |
| P14 | $\epsilon = 0.3, d_{\text{targ}} = 0.02, K_\pi = 10, K_v = 10$ | T14 | $\delta = 0.01, \beta = 1.0, K_v = 10$ |
| P15 | $\epsilon = 0.2, d_{\text{targ}} = 0.02, K_\pi = 10, K_v = 10$ | T15 | $\delta = 0.02, \beta = 0.8, K_v = 10$ |
| P16 | $\epsilon = 0.2, d_{\text{targ}} = 0.02, K_\pi = 20, K_v = 10$ | T16 | $\delta = 0.02, \beta = 1.0, K_v = 10$ |

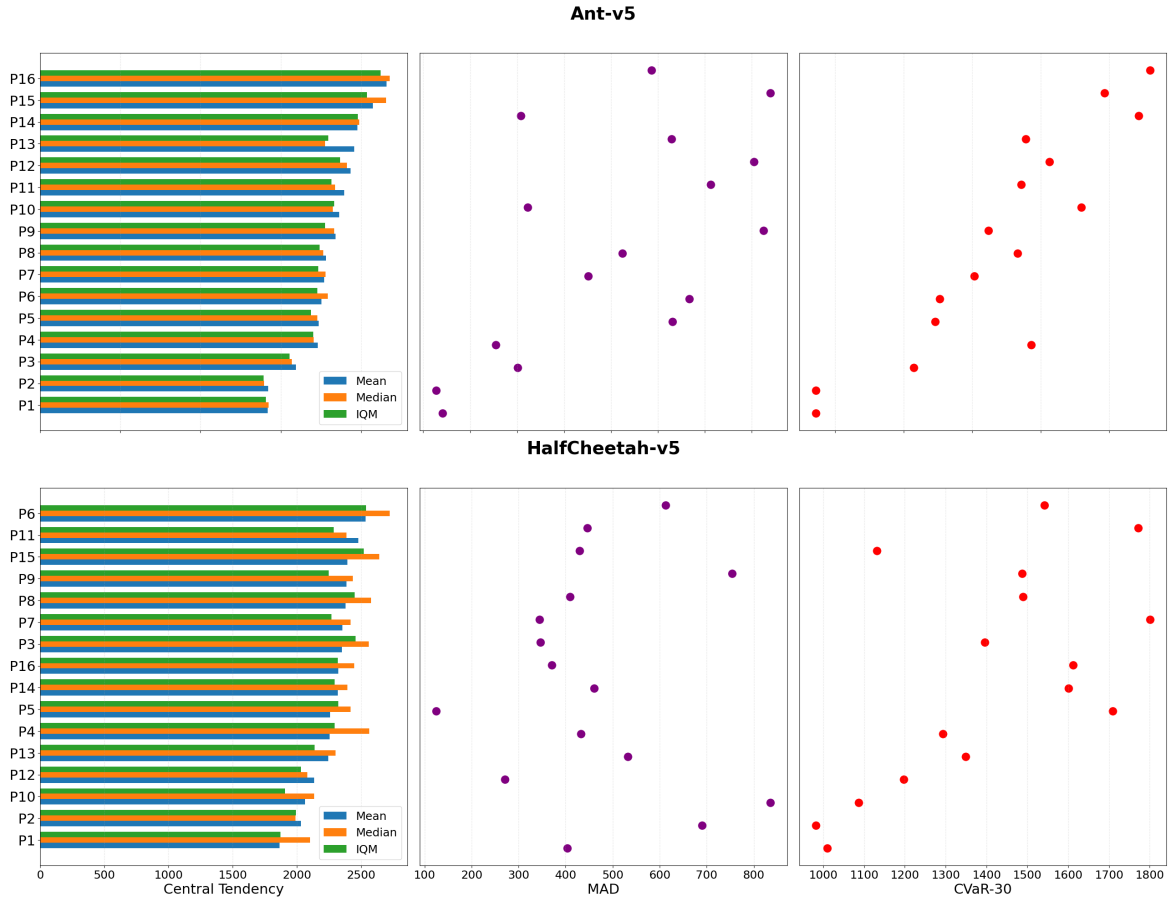


Fig. 2: PPO hyperparameter analysis across performance, variability, and worst-case metrics ($\alpha = 0.3$), computed from the final 100k training steps (i.e., steps $\geq$ 900k). A higher value defines BP, lower MAD defines LV, and higher CVaR defines BWC. (See Table IV for IDs.)

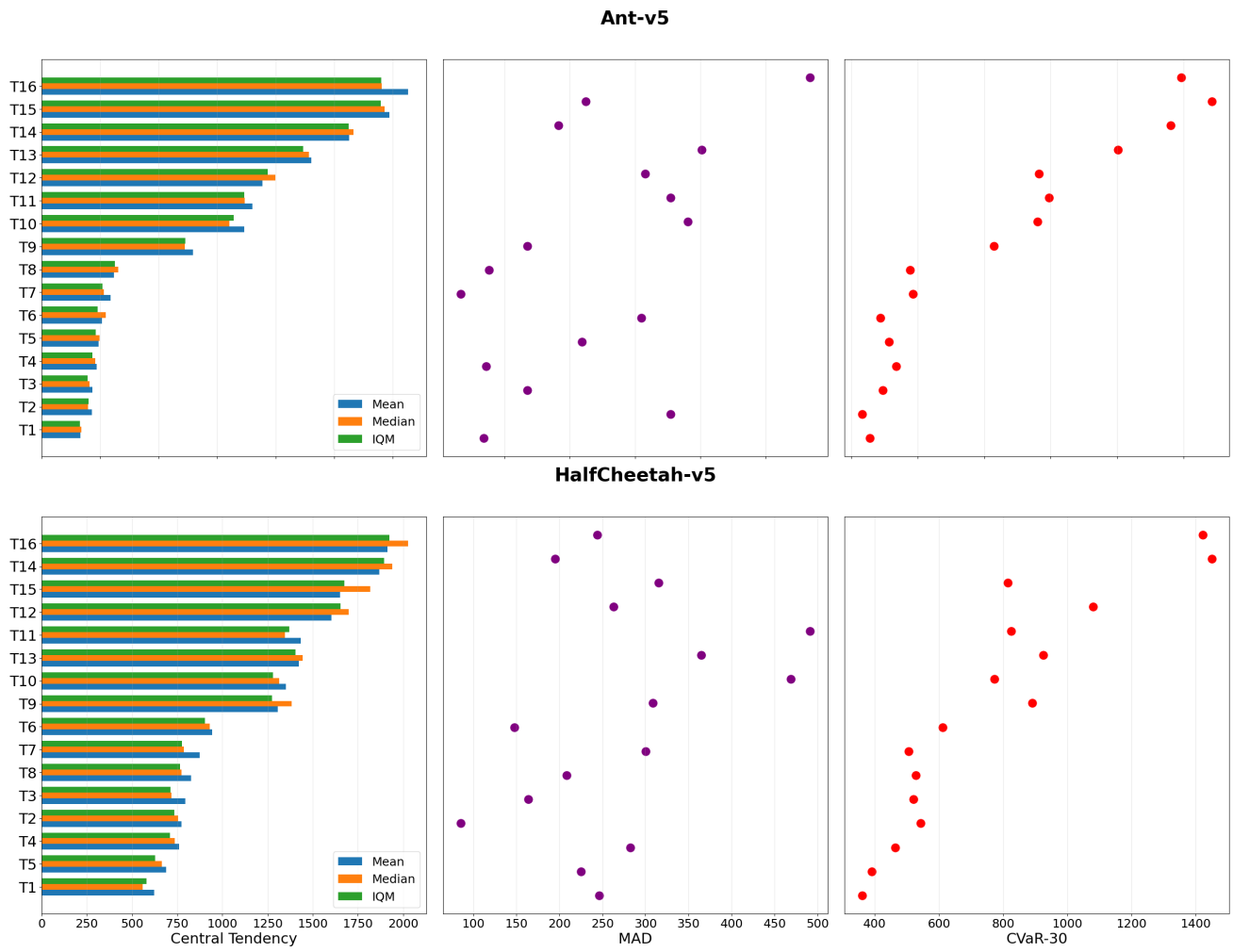


Fig. 3: TRPO hyperparameter analysis across performance, variability, and worst-case metrics ($\alpha = 0.3$), computed from the final 100k training steps (i.e., steps $\geq$ 900k). A higher value defines BP, lower MAD defines LV, and higher CVaR defines BWC. (See Table IV for IDs.)

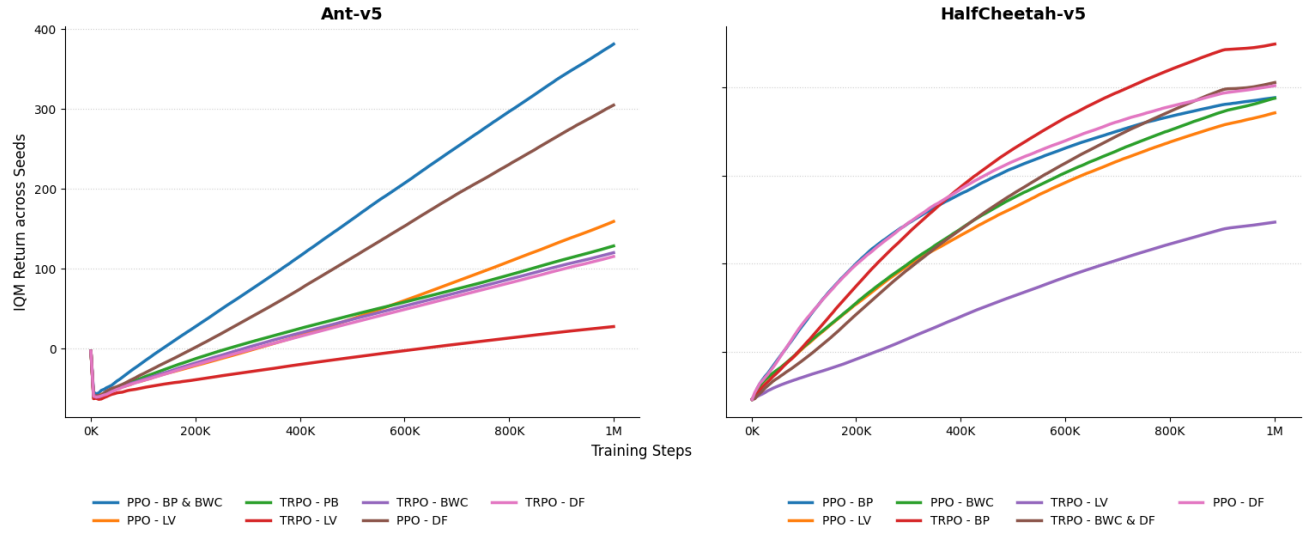


Fig. 4: IQM learning curves for both environments of Ant-v5 and HalfCheetah-v5 across all training steps