The Statistical Cost of Hyperparameter Tuning in Reinforcement Learning

Ziqi Tang [*1] **Xuezhou Zhang** [*1]

Abstract

The performance of reinforcement learning (RL) algorithms is often benchmarked without accounting for the cost of hyperparameter tuning, despite its significant practical impact. In this paper, we show that such practices distort the perceived efficiency of RL methods and impede meaningful algorithmic progress. We formalize this concern by proving a lower bound showing that tuning m hyperparameters in RL can induce an exponential $\exp(m)$ blow-up in the sample complexity or regret, in stark contrast to the linear $O(m)$ overhead observed in supervised learning. This highlights a fundamental inefficiency unique to RL. In light of this, we propose evaluation protocols that account for the number and cost of tuned hyperparameters, enabling fairer comparisons across algorithms. Surprisingly, we find that once tuning cost is accounted for, elementary algorithms can outperform their successors with more sophisticated design. These findings call for a shift in how RL algorithms are benchmarked and compared, especially in settings where efficiency and scalability are critical.

1. Introduction

While lacking a universally agreed definition, *hyperparameters* are broadly considered parameters that are set prior to running an algorithm and remain fixed throughout its execution. Examples include the step size in optimization, regularization coefficients, neural network architecture choices (e.g., depth, width), and activation functions. Although theoretical guidelines exist for some of these parameters (e.g., $\Theta(1/\sqrt{T})$ step size in stochastic gradient descent), practical deployments typically require manual or automated **hyperparameter optimization (HPO)** to identify the problem-specific optimal values. Due to the non-differentiable and sometimes discrete nature of this search space, HPO is usually done via grid search or derivative-free optimization over a combinatorial search space.

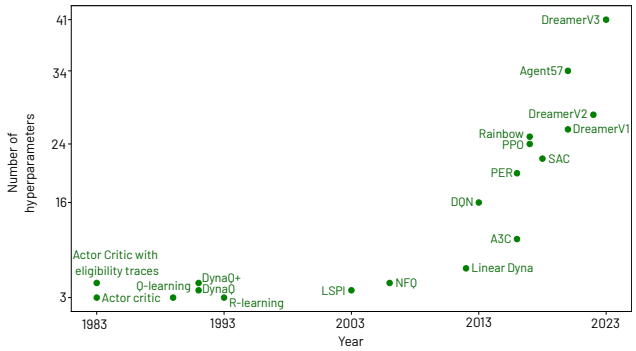


Figure 1. The number of hyperparameters in reinforcement learning algorithms proposed over the last decades (Adkins et al., 2024).

When it comes to reinforcement learning (RL), algorithm performance is notoriously sensitive to hyperparameter choices (Patterson et al., 2023; Eimer et al., 2023; Adkins et al., 2024; Obando-Ceron et al., 2024). Alarmingly, some studies have even reported practices of tuning random seeds as hyperparameters to overfit to public benchmarks(Henderson et al., 2018). Over the past decades, the number of hyperparameters in RL algorithms has steadily increased. For example, the original DQN algorithm (Mnih et al., 2015) required selecting 16 hyperparameters, while Rainbow (Hessel et al., 2018) introduced 25. This rising trend is illustrated in Figure 1. Numerous works have acknowledged this phenomenon and provided practical guidelines for selecting and evaluating hyperparameters in RL (Franke et al., 2021; Eimer et al., 2022; 2023; Patterson et al., 2024).

In this paper, we take on a more quantitative perspective and pose the following central question:

What is the statistical cost of HPO in online RL, and how should it be quantified?

We define this statistical cost as the additional sample complexity or regret incurred during HPO. In Section 3, we show that tuning m hyperparameters results in a sample complexity overhead of $\Theta(\exp(m))$ in RL—a stark contrast to the $O(m)$ overhead typically observed in supervised learning (SL). This exponential cost is especially problematic in settings where real-world interaction is expensive,

[1]Faculty of Computing & Data Sciences, Boston University. Correspondence to: Xuezhou Zhang <xuezhouz@bu.edu>.

Preprint. February 5, 2026.

simulators are unavailable, or large-scale offline datasets are lacking. We refer interested readers to the appendix for a formal proof of the exponential lower bound (Theorem 3.6) and omit proofs of the other Theorems and Corollaries as they are either classic results or are rather direct.

In light of the unneglectable cost of HPO in RL, we propose evaluation protocols that explicitly account for the HPO overhead. These enable fairer comparisons across algorithms and support practical questions such as:

- Is Algorithm A truly more efficient than Algorithm B?
- Which hyperparameters are worth tuning in a given deployment?

We explore these use cases in Section 5.

2. Related work

Early works such as Henderson et al. (2018); Machado et al. (2018) emphasized reproducibility challenges in RL, attributing much of the variance to opaque or inconsistent hyperparameter settings. Subsequent studies examined hyperparameter sensitivity more systematically across benchmark environments, introducing new sensitivity metrics and evaluation methods (Eimer et al., 2022; 2023; Adkins et al., 2024). Others advocated for AutoML solutions to automate HPO in RL (Franke et al., 2021; Eimer et al., 2023), or proposed benchmarks requiring shared hyperparameter configurations across tasks (Patterson et al., 2024). However, these works primarily focus on empirical sensitivity or tuning practices. None explicitly quantify the *statistical cost* of HPO in RL or examine its impact on algorithm comparison. Our paper fills this gap, offering a theoretical framework that complements these empirical efforts.

On the theoretical side, the cost of tuning has been explored under the lens of *online model selection*, where the goal is to choose the best base algorithm from a finite set (Agarwal et al., 2017; Abbasi-Yadkori et al., 2020; Ghosh et al., 2020; Chatterji et al., 2020; Bibaut et al., 2020; Foster et al., 2020; Lee et al., 2020; Wei et al., 2022). Since tuning m hyperparameters often corresponds to evaluating $M = O(\exp(m))$ configurations, HPO effectively reduces to a model selection problem over an exponentially large set. Approaches include FTRL-based methods (e.g., EXP4 (Odalric & Munos, 2011), Corral (Agarwal et al., 2017), Tsallis-INF (Arora et al., 2020)) and regret-balancing schemes (Pacchiano et al., 2020; Cutkosky et al., 2021). A common limitation is the requirement for known regret bounds for each base algorithm—a barrier in real-world RL where such guarantees are unavailable.

A recent exception is Dann et al. (2024), who propose an algorithm that competes against the *realized regret* of the best base model without requiring candidate regret bounds. They achieve regret $\widetilde{O}(d_T^\star M\sqrt{T} + (d_T^\star)^2\sqrt{MT})$, where $d_T^\star\sqrt{T}$ is the presumed regret of the best base model. On the other hand, the best known lower bound takes the form of $\Omega((d_T^\star)^2\sqrt{T})$ (Marinov & Zimmert, 2021), implying that recovering the regret rate of the best base learner is in general not possible. Interestingly, prior work in this community has emphasized dependence on $d^\star$ over M—in contrast to our focus on the exponential dependence in M arising from HPO.

Designing **parameter-free** algorithms directly is an alternative. This goal has been extensively pursued in optimization (Defazio & Mishchenko, 2023; Carmon & Hinder, 2022; Ivgi et al., 2023; Cutkosky et al., 2024; Khaled & Jin, 2024) and online learning (Orabona & Pál, 2018; Cutkosky & Orabona, 2018; van der Hoeven et al., 2020). In RL, however, work on parameter-free algorithms is scarce. Recent theoretical contributions include algorithms that adapt to unknown reward scales or state-space sizes (Chen & Zhang, 2023; 2024; Chen et al., 2024). Empirical studies like Yu et al. (2021) also explore adaptive hyperparameter schemes, though without the explicit goal of parameter-free design.

3. The Statistical Cost of Hyperparameter Optimization (HPO)

Historically, HPO has received minimal attention in both algorithm design and theoretical analysis because its cost is modest in classic settings like SL. In those domains, data splitting strategies such as cross-validation yield efficient HPO procedures. We begin by formalizing this baseline in SL and contrast it with the RL setting.

3.1. The HPO Cost in Supervised Learning

Let $D = \{(x, y)_i\}_{i=1}^N$ be an i.i.d. dataset sampled from a distribution P. Assume a supervised learning algorithm $\mathcal{A}_\theta$ is parameterized by hyperparameter θ and returns a predictor in a hypothesis class $\mathcal{F}$.

Definition 3.1 (PAC-learner in SL)**.** A learner $\mathcal{A}$ is a PAC-learner if, for all $\delta \in (0, 1)$, with probability at least $1 - \delta$ over draw of dataset D, we have:

$$\mathbb{E}_{(x,y)\sim P}\left[(\mathcal{A}(D)(x) - y)^2\right] - \min_{f\in\mathcal{F}} \mathbb{E}\left[(f(x) - y)^2\right] \leq \epsilon(N, \delta).$$

where $\epsilon(N, \delta)$ denotes the optimality gap.

Many SL algorithms achieve an optimality gap of the form:

$$\epsilon(N, \delta) = O\left(\sqrt{\frac{C_\mathcal{F} \log(1/\delta)}{N}}\right), \tag{1}$$

where $C_\mathcal{F}$ denotes the complexity of $\mathcal{F}$. An alternative metric to quantify learning efficiency is the **sample complexity**,

Algorithm 1 HPO via Data Splitting in SL

Input: Dataset D of size N, hyperparameter set Θ, learning algorithm $\mathcal{A}_\theta$
Split D into D_{train} and D_{val} of size $N/2$
for $\theta \in \Theta$ **do**
  Train $f_\theta = \mathcal{A}_\theta(D_{\text{train}})$
  Evaluate $\epsilon_\theta = \frac{2}{N}\sum_{(x,y)\in D_{\text{val}}}(f_\theta(x)-y)^2$
end for
Return $\hat{\theta} = \arg\min_{\theta\in\Theta}\ \epsilon_\theta$, model $f_{\hat{\theta}}$

i.e. the number of samples it requires to reach a certain optimality gap. For instance, (1) would translate to a sample complexity of

$$N(\epsilon,\delta) = O\left(\frac{C_{\mathcal{F}}\log(1/\delta)}{\epsilon^2}\right). \tag{2}$$

In supervised learning, HPO can be implemented using a simple data-splitting approach (Algorithm 1), where models are trained on the training data and hyperparameters are selected based on losses on the validation data:

Theorem 3.2 (PAC Guarantee of SL HPO)**.** *With probability $1-\delta$, Algorithm 1 returns $\hat{f}$ satisfying:*

$$\begin{aligned}\mathbb{E}\left[(\hat{f}(x)-y)^2\right] - \min_{f\in\mathcal{F}}\mathbb{E}\left[(f(x)-y)^2\right] &\le 2\min_{\theta\in\Theta}\epsilon_\theta(N/2,\delta/|\Theta|)\\ &\approx \tilde{O}\left(\min_{\theta\in\Theta}\sqrt{\frac{\log|\Theta|\,C_{\mathcal{F}_\theta}}{N}}\right).\end{aligned}$$

In other words, Algorithm 1 is itself a PAC-learner with no hyperparameter and a sample complexity $\log|\Theta|$ times that of the best base learner. This $\log|\Theta|$ comes from a union bound over Θ and is negligible in the big data regime ($N \gg \log|\Theta|$). This benign scaling is possible mainly due to data sharing across different hyperparameters, i.e. $\mathcal{A}_{\theta_1}$ and $\mathcal{A}_{\theta_2}$ can use the same training and validation data. As a result, Algorithm 1 is *modular* — it only requires **black-box access** to the learner, making it applicable to any learner $\mathcal{A}$. As a result of how effortless HPO is in SL, modern deep learning has evolved under nearly no selection pressure towards having fewer hyperparameters. However, it turns out to be catastrophic for HPO in reinforcement learning.

3.2. The HPO Cost in Online Reinforcement Learning

In RL, we no longer have fixed datasets nor trivial validation procedures. Consider an online PAC-RL setting, where the agent interacts with an unknown environment with the goal of finding a near-optimal policy in as few episodes as possible.

Definition 3.3 (PAC-agent in Online RL)**.** Given an MDP $\mathcal{M}$, a PAC-agent is defined as a learning agent that, with probability at least $1-\delta$, after interacting with $\mathcal{M}$ for T episodes, returns a policy $\hat{\pi}$ that satisfies

$$\max_\pi V^\pi_{\mathcal{M}} - V^{\hat{\pi}}_{\mathcal{M}} \le \epsilon(T,\delta),$$

for some function $\epsilon : \mathbb{N}\times(0,1)\to\mathbb{R}^+$.

Naively adapting Algorithm 1 to RL involves training a policy for each $\theta\in\Theta$ using $T/|\Theta|$ episodes (since data can no longer be shared seamlessly across different agents and will instead be split evenly), then evaluating each and selecting the best. This yields the following result:

Theorem 3.4 (PAC Guarantee of *naive* RL HPO)**.** *With probability $1-\delta$, this procedure returns $\hat{\pi}$ satisfying:*

$$\begin{aligned}\max_\pi V^\pi - V^{\hat{\pi}} &\le \min_{\theta\in\Theta}\epsilon_\theta(N/|\Theta|,\delta/|\Theta|)\\ &\approx \tilde{O}\left(\min_{\theta\in\Theta}\sqrt{\frac{|\Theta|\,C_{\mathcal{F}_\theta}}{T}}\right).\end{aligned}$$

The exponential size of $|\Theta| := M = O(\exp(m))$ implies that tuning m hyperparameters incurs an $\Omega(\exp(m))$ overhead. This is in fact how hyperparameters are tuned in most RL research. Taking a step further, we can in fact show that the above inefficiency is not just a weakness of this naive algorithm but rather a necessary cost for any black-box HPO algorithm, formally defined below.

Definition 3.5 (black-box HPO)**.** A *black-box* hyperparameter optimization (HPO) meta-algorithm HPO interacts with an unknown episodic MDP for a total of T episodes, and may adaptively allocate episodes among the base agents $\{\mathcal{A}_\theta|\theta\in\Theta\}$ while observing the realized trajectories and rewards, but without
(i) inspecting internal states of $\mathcal{A}_\theta$ beyond their externally visible actions or
(ii) sharing experience or internal state across different θ.
At the end of interaction, HPO must output a policy $\hat{\pi}$ that coincides with the output of one of the instantiated base agents after it has been trained on the allocated episodes.

Theorem 3.6 (Black-box HPO is inefficient in online RL)**.** *Let Θ be a finite hyperparameter set with $M := |\Theta| \ge 2$, and let $\{\mathcal{A}_\theta\}_{\theta\in\Theta}$ be a family of reinforcement-learning agents indexed by θ. Consider any* black-box *hyperparameter optimization (HPO) meta-algorithm* HPO.

For each $\theta\in\Theta$, let $\epsilon_\theta(n,\delta)$ denote a valid PAC optimality-gap function for $\mathcal{A}_\theta$ on a given MDP $\mathcal{M}$, meaning that when $\mathcal{A}_\theta$ is run for n episodes on $\mathcal{M}$, it outputs a policy π satisfying

$$\Pr(V^\star_{\mathcal{M}} - V^\pi_{\mathcal{M}} \le \epsilon_\theta(n,\delta)) \ \ge\ 1-\delta,$$

where $V^\star_{\mathcal{M}} := \max_\pi V^\pi_{\mathcal{M}}$.

Then there exists a family of episodic MDP $\mathcal{M}_\Theta$ such that no black-box meta-algorithm HPO *can output a policy $\hat{\pi}$ satisfying*

$$V^\star_{\mathcal{M}} - V^{\hat{\pi}}_{\mathcal{M}} \;\le\; \min_{\theta\in\Theta} \epsilon_\theta\left(\frac{2T}{M}, \frac{1}{2}\right) \tag{3}$$

with probability strictly greater than $1/2$ on all of $\mathcal{M}_\Theta$.

The formal proof is deferred to the appendix. This lower bound builds on two key observations: First, no data sharing between base agents is possible in online RL without algorithm specific structures, which is unobtainable in the black-box setting. Thus, the total budget of T episodes must be split between $|\Theta|$ different base agents. Second, no base agents can be dropped prematurely without additional assumption on $\epsilon_\theta(T, \delta)$ beyond monotonicity, e.g., a base agent that starts off performing poorly could potentially catch up and become the best after some time. Therefore, there is in fact no strategy that guarantees to be better than spending $T/|\Theta|$ episodes on each base agent in the worst case.

Subsequently, the regret to sample complexity reduction implies a similar lower bound on the regret:

Corollary 3.7 (Regret Lower Bound)**.** *Given any base agent $\mathcal{A}$ with a hyperparameter set Θ and denote $Reg_\theta(T, \delta)$ the regret corresponding to hyperparameter θ, where regret is defined as the cumulative optimality gap during the execution of the algorithm, $\sum_{t=1}^{T}(\max_\pi V^\pi - V^{\hat{\pi}_t})$. Then, no black-box HPO algorithm can achieve a regret bound better than*

$$\min_{\theta\in\Theta} \frac{|\Theta|}{2} Reg_\theta(2T/|\Theta|, 1/2)$$

with probability more than $1/2$.

Notice that the lower bound in Theorem 3.6 matches with the upper bound in Theorem 3.4, implying that the naive strategy of splitting data equally across all hyperparameters is in fact optimal for the pure exploration problem. When it comes to regret, Corollary 3.7 complements the existing lower bound of Marinov & Zimmert (2021). Yet, there is still a gap between the best known upper bound of Dann et al. (2024) and both lower bounds, which is left for future research to resolve. Nevertheless, our lower bounds are sufficient to show that, unlike SL, the cost of hyperparameter tuning in RL is *multiplicative* rather than logarithmic, and therefore must be accounted for during algorithm evaluation.

However, most existing RL research ignores this cost in reporting algorithm performance, often presenting results for the best-tuned hyperparameter configuration while omitting the number of trials or total samples used. An example of such practices is given in Figure 2. Such practices give an unfair advantage to complex algorithms with more tunable hyperparameters. A model with twice the learning speed but ten times the tuning burden may still be presented as superior. This dynamic skews empirical comparisons and hampers progress toward scalable, real-world deployable RL algorithms. To address this lack of rigor, we next propose evaluation metrics that properly accounts for the cost of HPO, and demonstrate how the new metrics can reshape algorithm rankings, and how elementary algorithms can outperform sophisticated baselines when tuning is properly accounted for.

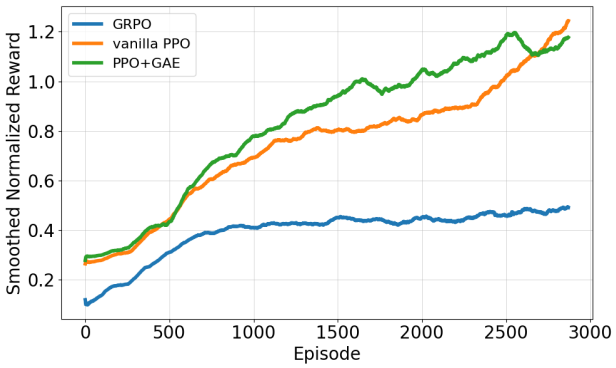


Figure 2. A typical comparison between different RL algorithms based on the performance with the best hyperparameter configuration for each algorithm. A plausible conclusion one may draw from a plot like this is `PPO+GAE` > `vanilla PPO` > `GRPO`.

4. HPO-aware Evaluation for Online RL

Despite the lack of an optimal hyperparameter optimization (HPO) algorithm for RL (in terms of regret), it remains essential to measure and compare the learning efficiency of RL algorithms in a way that fairly incorporates the cost of tuning. In this section, we adopt an *optimistic* yet principled approach: we use the theoretical lower bounds established in Section 3 as a guideline to construct practical evaluation metrics. These metrics serve to assess the performance of RL algorithms while explicitly penalizing for the number of hyperparameters being tuned.

We introduce two core metrics: *Effective Sample Complexity* and *Effective Area Under the Curve*. These metrics aim to mirror real-world deployment scenarios where tuning is costly, and help distinguish algorithms that are truly efficient from those that merely perform well under exhaustive hyperparameter search.

Definition 4.1 (Effective Sample Complexity (SC))**.** Let $\mathcal{A}$ be a reinforcement learning algorithm with a hyperparameter set Θ. The **Effective Sample Complexity** required to reach an optimality threshold ϵ is defined as:

$$|\Theta| \times \min_{\theta\in\Theta} T_\theta(\epsilon), \tag{4}$$

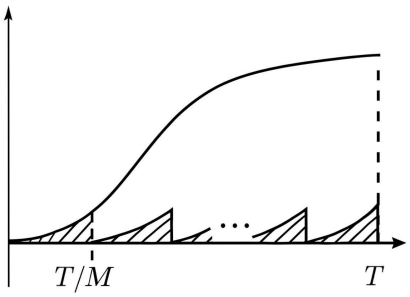


Figure 3. An illustration of the Effective AUC, represented by the shaded area under the curve.

where $T_\theta(\epsilon)$ is the number of episodes needed for $\mathcal{A}_\theta$ to produce a policy $\hat{\pi}$ such that $\max_\pi V^\pi - V^{\hat{\pi}} \leq \epsilon$.

This metric captures the cost of HPO in the number of environment interactions to find a near-optimal policy. Notice that (4) resembles the matching upper and lower bounds of Theorem 3.6 and Theorem 3.4, and the multiplicative $|\Theta|$ factor reflects the fundamental inefficiency identified in our theoretical analysis. Importantly, we assume the best configuration is selected in hindsight, making this a lower bound on true tuning cost. In our experiments, the threshold ϵ is chosen as the 90th percentile of episodic returns aggregated over all configurations and algorithms.

While sample complexity measures the data needed to achieve competent performance, many real-world applications also care about cumulative reward during the learning process. Thus, we introduce a regret-based metric:

Definition 4.2 (Effective Area Under the Curve (AUC))**.** Let $\mathcal{A}$ be an RL algorithm with hyperparameter set Θ. The **Effective AUC** over T episodes is defined as:

$$\min_{\theta\in\Theta} |\Theta| \times \sum_{t=1}^{T/|\Theta|} V^{\pi_{\theta,t}}, \tag{5}$$

where $V^{\pi_{\theta,t}}$ is the expected reward of the policy at episode t when using configuration θ.

Intuitively, (5) measures the cumulative rewards achieved by each configuration over a period of $T/|\Theta|$ episodes, then multiplying it by $|\Theta|$, as illustrated in Figure 3. This is derived directly from our lower bound in Corollary 3.7, and should be viewed as *optimistic*, in the sense that this is the least amount of regret one would suffer by calling an online model selection algorithm for hyperparameter tuning. Notice that the naive data splitting framework in Algorithm 1 would incur a $O(T)$ regret in the worst case, because there is no guarantee on how much more regret a suboptimal hyperparameter would incur comparing to the optimal one.

The Effective AUC is designed to measure learning performance under a fixed total training budget T. When the hyperparameter space $|\Theta|$ is larger, each configuration is allocated fewer episodes ($T/|\Theta|$), making early learning behavior critical. The multiplicative factor $|\Theta|$ ensures comparability across different hyperparameter set sizes, while the reduction in per-configuration training horizon encodes the cost of tuning larger hyperparameter spaces. This metric captures how efficiently an algorithm performs at the beginning of training, providing a more practical assessment of learning effectiveness in resource-constrained settings.

Taken together, these two metrics allow us to rethink what it means for an RL algorithm to be efficient. Rather than asking "how well does this algorithm perform after tuning?", we ask "how much reward and performance is achievable if we must account for the tuning effort?"

5. Experiment

We now demonstrate the proposed metrics on a suite of continuous control tasks from the MuJoCo benchmark, including Hopper, Ant, Swimmer, HalfCheetah, and Walker2d. Our goal is twofold: (1) guide practitioners in making tuning decisions under practical constraints, and (2) assess how tuning overhead alters algorithm comparisons.

Algorithms We focus on four policy gradient algorithms: Group Relative Policy Optimization (`GRPO`), vanilla Proximal Policy Optimization (`PPO`), PPO+Generalised Advantage Estimator (`PPO+GAE`), and a variant of PPO that performs Advantage per-minibatch zero-mean normalization (`PPO+ADVN`).

All algorithms optimize the same clipped surrogate PPO objective:

$$L(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right], \tag{6}$$

Each algorithm differs in how $\hat{A}_t$ is computed.

- `GRPO`: $\hat{A}_t^{GRPO} = \frac{R_t - \text{mean}(R_t)}{\text{std}(R_t)}$
- `Vanilla PPO`: $\hat{A}_t^{PPO} = R_t - V(s_t)$
- `PPO+GAE`: $\hat{A}_t^{GAE} = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1}$, where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$.
- `PPO+ADVN`: $\hat{A}_t^{ADVN} = \frac{\hat{A}_t^{GAE} - \text{mean}(\hat{A}_t^{GAE})}{\text{std}(\hat{A}_t^{GAE})}$

In particular, `GRPO` does not use a value network and therefore only has two hyperparameters: actor learning rate (LR) and entropy regularizer coefficient (τ). Vanilla `PPO` uses a value network to help with the estimation of the advantage function and thus have the critic learning rate (LR) as an additional hyperparameter. Both `PPO+GAE` and `PPO+ADVN` additionally have the GAE hyperparameter (λ). The corresponding hyperparameters and their values are listed in

Algorithm	Actor LR	Critic LR	Entropy Coef (τ)	GAE λ	M
GRPO	{1e-5, 3e-5, 1e-4}	NA	{1e-3, 1e-2, 1e-1}	NA	3^2
Vanilla PPO	{1e-5, 3e-5, 1e-4}	{1e-4, 3e-4, 1e-3}	{1e-3, 1e-2, 1e-1}	NA	3^3
PPO+GAE	{1e-5, 3e-5, 1e-4}	{1e-4, 3e-4, 1e-3}	{1e-3, 1e-2, 1e-1}	{0.5, 0.7, 0.9}	3^4
PPO+ADVN	{1e-5, 3e-5, 1e-4}	{1e-5, 1e-4, 1e-3}	{1e-3, 3e-3, 1e-2}	{0.3, 0.5, 0.7}	3^4

Table 1. Hyperparameter set for each algorithm.

Number of HPs tuned	4	3	2	1
PPO+GAE (AUC)	actorlr, criticlr, τ, λ	criticlr, τ, λ	τ, λ	λ
PPO+ADVN (AUC)	actorlr, criticlr, τ, λ	criticlr, τ, λ	criticlr, λ	λ
PPO+GAE (SC)	actorlr, criticlr, τ, λ	actorlr, τ, λ	τ, λ	λ
PPO+ADVN (SC)	actorlr, criticlr, τ, λ	actorlr, τ, λ	actorlr, λ	λ

Table 2. Hyperparameters tuned for PPO+GAE and PPO+ADVN.

Table 1. The hyperparameter ranges are picked around the reported best performing hyperparameter configuration for each algorithm and thus could be different across algorithms. For each hyperparameter configuration, we run 10 independent trials per environment, with each trial consisting of 3,000,000 maximum timesteps.

Preprocessing For each environment, we first truncate all reward sequences to a common length, determined by the minimum sequence length among all trials for all configurations. This common length defines the effective budget T used in Definition 4.2. The truncation is needed because each environment may have different episode length. After truncation, we average the reward-vs-T curve across 10 trials for each configuration to get a single curve per (environment, algorithm, hyperparameter) tuple. Based on these averaged reward trajectories, we collect all reward values across full-grid configurations within each environment, and compute the 5% and 95% quantiles of this empirical reward distribution, denoted as $p_5(e)$ and $p_{95}(e)$, where e represents the environment.

Since raw reward scales differ across environments, we normalize the return $R(\tau)$ of each averaged trajectory to the range $[0, 1]$ using per-environment quantile normalization prior to computing the Effective SC and Effective AUC:

$$\bar{R}(\tau) = \frac{R(\tau) - p_5(e)}{p_{95}(e) - p_5(e)}, \tag{7}$$

where $p_5(e)$ and $p_{95}(e)$ are the 5% and 95% quantiles of returns in environment e. This ensures fair metric comparison across tasks. We then calculate the effective SC and effective AUC using these normalized averaged curves.

In this fixed-step setting, we record the resulting total number of trajectories in each individual run. Let m denote the number of tuned hyperparameters, and $M = 3^m$ represents the total number of hyperparameter configurations.

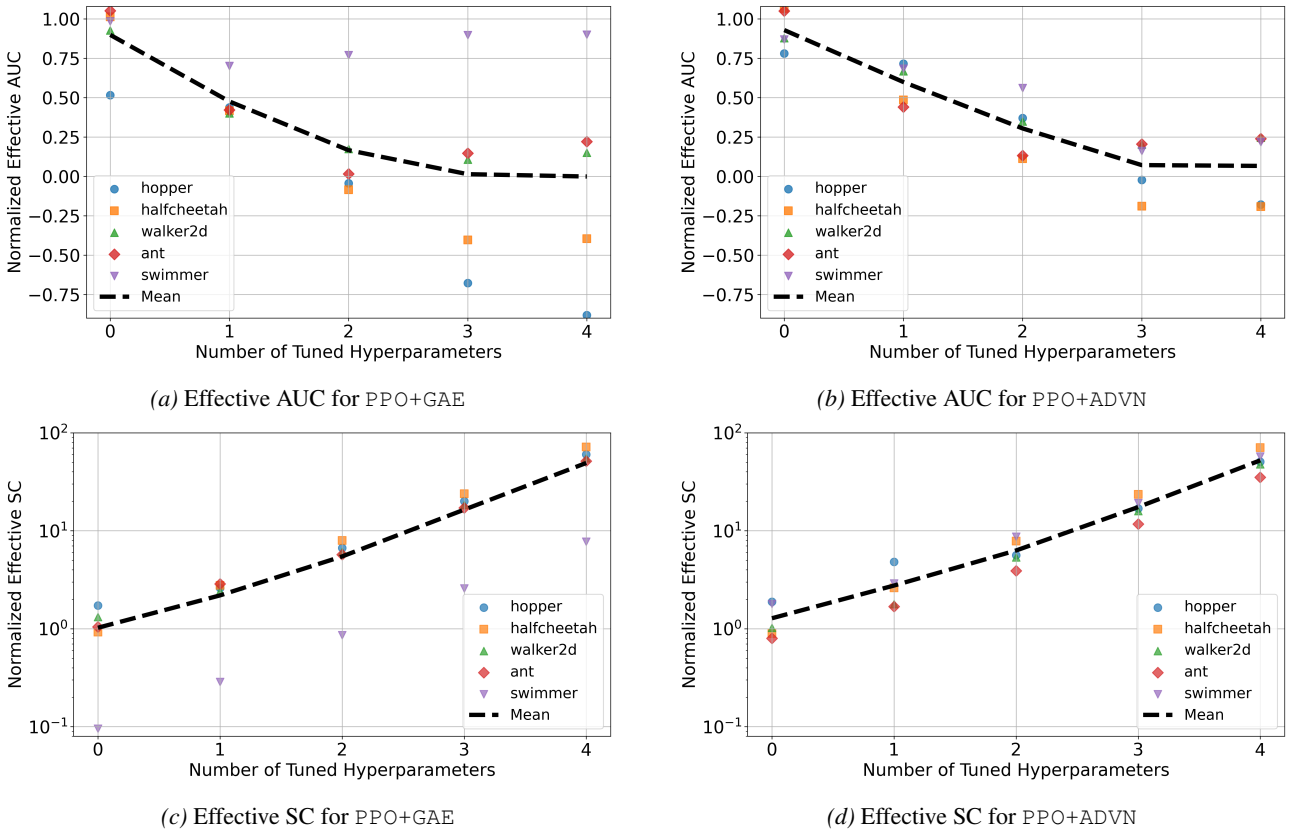
5.1. Use Case I: Choosing Hyperparameters to Tune

Consider a practitioner deploying RL in a new task similar to MuJoCo. They face a practical question: which hyperparameters should be tuned online, and which can be fixed based on prior knowledge or auxiliary environments? Tuning all hyperparameters may yield the best result in hindsight, but it also incurs exponential cost. We simulate this setting using PPO+GAE and PPO+ADVN, both with 4 hyperparameters.

For each value $m = 0, 1, 2, 3, 4$, we search over all combinations of hyperparameters where m of them can vary across environments while the other $(4 - m)$ are fixed across all environments. This procedure results in a sequence of hyperparameter spaces of increasing size: $M = 1$, 3^1, 3^2, 3^3 and 3^4, corresponding to zero, one, two, three, and four tunable hyperparameters.

We start with the full grid of $M = 3^4$ configurations for both algorithms, computing the effective AUC (respectively SC) of each configuration. We first identify the configuration that achieves the highest average effective AUC (respectively lowest SC) across all environments in each algorithm, which serves as the fixed reference point for all subsequent search. Starting from this configuration ($M = 1$), we progressively increase the number of tunable hyperparameters by releasing them one by one, following a dynamic order determined by the influence of each hyperparameter on the learning performance. Then, we tune it in an increased configuration space for both algorithms. This process continues until all hyperparameters have been released (i.e., all 4 hyperparameters are tunable). Any fixed hyperparameter always retains its value from the global best configuration to ensure consistent baseline comparisons.

For each environment, we measure the effective AUC (respectively SC) of all $M = 3^4$ original configurations using the full trajectory budget ($T/1$), and extract the 5th and



(a) Effective AUC for PPO+GAE

(b) Effective AUC for PPO+ADVN

(c) Effective SC for PPO+GAE

(d) Effective SC for PPO+ADVN

Figure 4. Normalized effective AUC&SC vs. number of tuned hyperparameters.

95th quantile values. The quantiles of effective SC values are selected before scaling by M. These quantiles serve as normalization anchors.

To compare across different M values and environments, we normalize all effective AUC (respectively SC) values using the corresponding environment-specific quantiles, similar to how rewards are normalized, ensuring that easy environment and hard environment are weighted equally.

$$\text{Normalized effective AUC} = \frac{AUC - p_{5,\text{AUC}}(e)}{p_{95,\text{AUC}}(e) - p_{5,\text{AUC}}(e)}, \tag{8}$$

where $p_{5,\text{AUC}}(e)$ and $p_{95,\text{AUC}}(e)$ represent the 5% and 95% quantiles of the effective AUC in environment e.

Considering that the effective SC can vary significantly across configurations within the same environment, we apply a different normalization scheme for it :

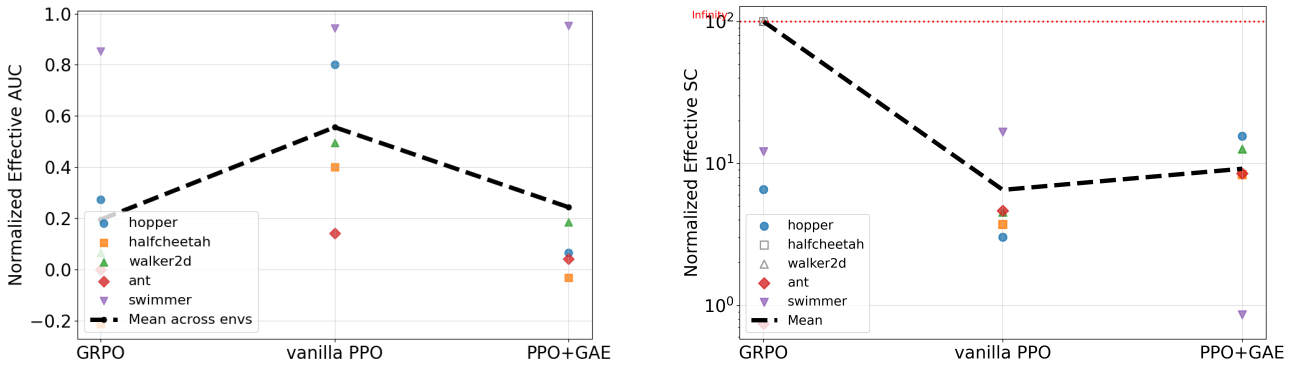
$$\text{Normalized effective SC} = \frac{SC}{p_{5,\text{SC}}(e)}, \tag{9}$$

where $p_{5,\text{SC}}(e)$ represents the 5% quantile of the effective SC among the full-grid configurations in environment e, before scaling by M.

Since m varies at each stage, the scaling factor 3^m changes accordingly. For each configuration, the number of steps required to reach the performance threshold remains constant across different m values, therefore, multiplying this constant by the varying scaling factor naturally leads to a positive correlation between effective SC and m.

We then calculate the average normalized effective AUC (respectively SC) across all five environments. The hyperparameter configuration that achieves the highest average normalized effective AUC (respectively lowest SC) is chosen as the optimal setup for its specific m. In other words, the optimal configuration for each m tells us which hyperparameter should be fixed across environments and at what value, while the other hyperparameters should be tuned per environment. The hyperparameters that are tuned for each m is shown in Table 2.

The performances of the best configuration for each m is shown in Figure 4, across both algorithms and metrics. The dashed line represents the mean normalized effective AUC (respectively SC) across environments at each m, highlighting overall trends.



(a) Effective AUC

(b) Effective SC

Figure 5. Normalized Effective AUC/SC vs different algorithms. Infinite sample complexity values are represented by (10^2) for visualization purposes.

Figure 4 shows a consistent trend: allowing more tunable hyperparameters does not always improve performance. In fact, both effective AUC and SC typically degrade with additional tuning flexibility. This suggests that in practice, a judicious selection of one or two hyperparameters to tune can outperform more complex tuning setups, particularly when the tuning budget is constrained.

5.2. Use Case II: Fair Comparison between RL Algorithms

Next, we use our metrics to reevaluate algorithm comparisons. We focus on three algorithms of increasing complexity: `GRPO` (2 hyperparameters), vanilla `PPO` (3 hyperparameters), and `PPO+GAE` (4 hyperparameters).

In standard RL benchmarking, algorithms are often compared based on the performance of their best hyperparameter configuration. As shown in Figure 2, this naive evaluation favors `PPO+GAE` in both AUC and sample complexity. However, this ranking does not reflect the underlying cost of searching through different hyperparameter configurations, nor the number of steps required to reach high performance.

To address this, we compare the algorithms using our proposed metrics. We first construct the full configuration grids for each algorithm: GRPO ($M = 3^2$), vanilla PPO ($M = 3^3$), and PPO+GAE ($M = 3^4$). For effective AUC calculation, instead of using the standard T/M truncation, we sum rewards over the first $T/(M/3^2)$ episodes for each configuration. This is equivalent to training each algorithm for 9 times longer and then evaluate. This is because we found empirically that $T/(3^4)$ is too short for PPO+GAE to find a reasonably good policy.

Once we adjust for tuning cost via normalized effective AUC and SC (Figure 5), the ranking changes dramatically. Vanilla PPO consistently outperforms both `GRPO` and `PPO+GAE`, achieving a better balance between performance and tuning overhead. Interestingly, `GRPO`, despite having the fewest hyperparameters, performs the worst under both metrics and even fails to meet the 90% performance threshold in two environments, leading to infinite sample complexity, which is represented by 10^2 for visualization purposes.

6. Conclusion and Discussion

This work draws attention to a pervasive blind spot in reinforcement learning research: the statistical and computational cost of hyperparameter optimization. While supervised learning tolerates a modest $\log(|\Theta|)$ tuning overhead, RL suffers a $|\Theta|$ penalty, both in theory and in practice. Our results show that ignoring this cost leads to misleading conclusions in algorithm comparisons and suboptimal design decisions.

To address this, we introduce two metrics—*Effective Sample Complexity* and *Effective AUC*—that quantify an algorithm's learning efficiency while accounting for the cost of hyperparameter tuning. These metrics, grounded in theory, enables algorithm comparisons that properly accounts for the HPO overhead.

A case study on the MuJoCo tasks demonstrates that:

- Many hyperparameters are not worth tuning; their cost outweighs their benefit.
- Algorithms with more complex design do not always outperform simpler baselines when fair comparisons are made.

We advocate for a shift in evaluation practices. RL research should routinely include HPO-aware metrics and prioritize the development of **hyperparameter-free algorithms** that minimize or eliminate tuning altogether. Only then can we build RL systems that are scalable, robust, and ready for real-world deployment.

Impact Statement

This work argues that a large fraction of reported progress in reinforcement learning can be an artifact of unreported hyperparameter search rather than genuine algorithmic improvement. By formalizing the statistical cost of hyperparameter optimization in online RL and proposing HPO-aware evaluation metrics, our primary intended impact is methodological: to improve the validity, comparability, and reproducibility of RL benchmarks, and to incentivize research directions that reduce reliance on extensive tuning (e.g., robust or hyperparameter-free algorithms).

The most direct positive societal consequence is indirect: more reliable evaluation can accelerate the translation of RL into domains where interaction data are costly or safety-critical, by discouraging brittle systems whose performance depends on expensive, problem-specific tuning. The proposed metrics also support more transparent reporting of experimentation budgets, which can reduce wasteful compute use and improve scientific efficiency.

Potential negative impacts stem from misuse or misinterpretation. First, penalizing tuning may lead some practitioners to under-tune systems deployed in high-stakes settings, reducing performance or safety margins; our recommendation is not to avoid tuning, but to report and account for its cost when making claims of efficiency. Second, standardizing tuning-aware evaluation could concentrate effort on small hyperparameter grids or benchmarks that favor certain design choices; we therefore view our protocols as complements to, not replacements for, task-specific validation and domain constraints. Overall, we expect the net effect to be improved rigor in RL evaluation and more honest accounting of real-world sample and compute requirements.

References

Abbasi-Yadkori, Y., Pacchiano, A., and Phan, M. Regret balancing for bandit and rl model selection. *arXiv preprint arXiv:2006.05491*, 2020.

Adkins, J., Bowling, M., and White, A. A method for evaluating hyperparameter sensitivity in reinforcement learning. *Advances in Neural Information Processing Systems*, 37:124820–124842, 2024.

Agarwal, A., Luo, H., Neyshabur, B., and Schapire, R. E. Corralling a band of bandit algorithms. In *Conference on Learning Theory*, pp. 12–38. PMLR, 2017.

Arora, R., Marinov, T. V., and Mohri, M. Corralling stochastic bandit algorithms. *arXiv preprint arXiv:2006.09255*, 2020.

Bibaut, A. F., Chambaz, A., and van der Laan, M. J. Rate-adaptive model selection over a collection of black-box contextual bandit algorithms. *arXiv preprint arXiv:2006.03632*, 2020.

Carmon, Y. and Hinder, O. Making sgd parameter-free. In *Conference on Learning Theory*, pp. 2360–2389. PMLR, 2022.

Chatterji, N., Muthukumar, V., and Bartlett, P. Osom: A simultaneously optimal algorithm for multi-armed and linear contextual bandits. In *International Conference on Artificial Intelligence and Statistics*, pp. 1844–1854, 2020.

Chen, M. and Zhang, X. Improved algorithms for adversarial bandits with unbounded losses. *arXiv preprint arXiv:2310.01756*, 2023.

Chen, M. and Zhang, X. Scale-free adversarial reinforcement learning. *arXiv preprint arXiv:2403.00930*, 2024.

Chen, M., Pacchiano, A., and Zhang, X. State-free reinforcement learning. *Advances in Neural Information Processing Systems*, 37:117708–117736, 2024.

Cutkosky, A. and Orabona, F. Black-box reductions for parameter-free online learning in banach spaces. In *Conference On Learning Theory*, pp. 1493–1529. PMLR, 2018.

Cutkosky, A., Dann, C., Das, A., Gentile, C., Pacchiano, A., and Purohit, M. Dynamic balancing for model selection in bandits and rl. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 2276–2285. PMLR, 2021.

Cutkosky, A., Defazio, A., and Mehta, H. Mechanic: A learning rate tuner. *Advances in Neural Information Processing Systems*, 36, 2024.

Dann, C., Gentile, C., and Pacchiano, A. Data-driven online model selection with regret guarantees. In *International Conference on Artificial Intelligence and Statistics*, pp. 1531–1539. PMLR, 2024.

Defazio, A. and Mishchenko, K. Learning-rate-free learning by d-adaptation. In *International Conference on Machine Learning*, pp. 7449–7479. PMLR, 2023.

Eimer, T., Benjamins, C., and Lindauer, M. Hyperparameters in Contextual RL are Highly Situational. *CoRR*, abs/2212.10876, 2022.

Eimer, T., Lindauer, M., and Raileanu, R. Hyperparameters in reinforcement learning and how to tune them. In *International conference on machine learning*, pp. 9104–9149. PMLR, 2023.

Foster, D., Gentile, C., Mohri, M., and Zimmert, J. Adapting to misspecification in contextual bandits. In *Advances in Neural Information Processing Systems*, 2020.

Franke, J. K., Köhler, G., Biedenkapp, A., and Hutter, F. Sample-Efficient Automated Deep Reinforcement Learning. In *International Conference on Learning Representations (ICLR)*, 2021.

Ghosh, A., Sankararaman, A., and Ramchandran, K. Problem-complexity adaptive model selection for stochastic linear bandits. *arXiv preprint arXiv:2006.02612*, 2020.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Hessel, M., Modayil, J., Hasselt, H. v., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. G., and Silver, D. Rainbow: Combining Improvements in Deep Reinforcement Learning. In *Conference on Artificial Intelligence (AAAI)*, 2018.

Ivgi, M., Hinder, O., and Carmon, Y. Dog is sgd's best friend: A parameter-free dynamic step size schedule. In *International Conference on Machine Learning*, pp. 14465–14499. PMLR, 2023.

Khaled, A. and Jin, C. Tuning-free stochastic optimization. *arXiv preprint arXiv:2402.07793*, 2024.

Lee, J. N., Pacchiano, A., Muthukumar, V., Kong, W., and Brunskill, E. Online model selection for reinforcement learning with function approximation. *arXiv preprint arXiv:2011.09750*, 2020.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

Marinov, T. V. and Zimmert, J. The pareto frontier of model selection for general contextual bandits. *Advances in Neural Information Processing Systems*, 34:17956–17967, 2021.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-Level Control through Deep Reinforcement Learning. *Nature*, 518:529–533, 2015.

Obando-Ceron, J., Araújo, J. G., Courville, A., and Castro, P. S. On the consistency of hyper-parameter selection in value-based deep reinforcement learning. *arXiv preprint arXiv:2406.17523*, 2024.

Odalric, M. and Munos, R. Adaptive bandits: Towards the best history-dependent strategy. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 570–578. JMLR Workshop and Conference Proceedings, 2011.

Orabona, F. and Pál, D. Scale-free online learning. *Theoretical Computer Science*, 716:50–69, 2018.

Pacchiano, A., Dann, C., Gentile, C., and Bartlett, P. Regret bound balancing and elimination for model selection in bandits and rl. *arXiv preprint arXiv:2012.13045*, 2020.

Patterson, A., Neumann, S., White, M., and White, A. Empirical Design in Reinforcement Learning. *CoRR*, abs/2304.01315, 2023.

Patterson, A., Neumann, S., Kumaraswamy, R., White, M., and White, A. M. The Cross-Environment Hyperparameter Setting Benchmark for Reinforcement Learning. In *Reinforcement Learning Conference (RLC)*, 2024.

van der Hoeven, D., Cutkosky, A., and Luo, H. Comparator-adaptive convex bandits. *Advances in Neural Information Processing Systems*, 33:19795–19804, 2020.

Wei, C.-Y., Dann, C., and Zimmert, J. A model selection approach for corruption robust reinforcement learning. In *International Conference on Algorithmic Learning Theory*, pp. 1043–1096. PMLR, 2022.

Yu, T., Kumar, A., Rafailov, R., Rajeswaran, A., Levine, S., and Finn, C. Combo: Conservative offline model-based policy optimization. *Advances in neural information processing systems*, 34:28954–28967, 2021.

A. Proof of Theorem 3.6

Proof. Fix $M \coloneqq |\Theta| \geq 2$ and an episode budget $T \in \mathbb{N}$. Let

$$L \;\coloneqq\; \left\lceil \frac{2T}{M} \right\rceil .$$

We construct (i) a family of episodic MDPs $\{\mathcal{M}_i\}_{i=1}^{M}$ and (ii) a family of base agents $\{\mathcal{A}_\theta\}_{\theta=1}^{M}$ such that any black-box HPO fails on at least one $\mathcal{M}_i$ with probability at least $1/2$.

Step 1: A hard family of MDPs. For each $i \in \{1, \ldots, M\}$, define an episodic MDP $\mathcal{M}_i$ with a single state s and horizon $H = 1$. The action set is

$$\mathcal{A} \coloneqq \{0, 1, 2, \ldots, M\},$$

and the reward is deterministic:

$$r_i(a) \coloneqq \mathbf{1}\{a = i\}.$$

Thus the optimal value is $V^\star_{\mathcal{M}_i} = 1$, achieved by the deterministic policy π_i that plays action i with probability 1. Any policy that never plays i attains value 0 and hence has optimality gap 1.

Step 2: Base agents with delayed identification. We define the base agent $\mathcal{A}_\theta$ for each $\theta \in \{1, \ldots, M\}$ as follows. When run on any $\mathcal{M}_i$, the agent maintains a private counter n_θ equal to the number of episodes it has been executed so far. For the first L executions ($n_\theta < L$), the agent plays action 0 deterministically. Once $n_\theta \geq L$, the agent switches permanently to playing action θ deterministically, and outputs the stationary policy "always play θ".

Note that, for each θ, the first L episodes of interaction are identical across all environments and yield reward 0 in every episode, since action 0 never yields reward.

Step 3: The oracle benchmark is zero. Fix any $i \in \{1, \ldots, M\}$. If $\mathcal{A}_i$ is run alone on $\mathcal{M}_i$ for L episodes, then at the end it outputs the policy "always play i", which is optimal on $\mathcal{M}_i$. Hence, for $\delta = 1/2$, we may take

$$\epsilon_i(L, 1/2) = 0 \qquad \text{on } \mathcal{M}_i.$$

For any $\theta \neq i$, running $\mathcal{A}_\theta$ for L episodes yields the policy "always play θ", which attains value 0 on $\mathcal{M}_i$ and thus has optimality gap 1, so $\epsilon_\theta(L, 1/2) \geq 1$ on $\mathcal{M}_i$. Therefore,

$$\min_{\theta \in \Theta} \epsilon_\theta(L, 1/2) = 0 \qquad \text{on } \mathcal{M}_i. \tag{10}$$

Since $L = \lceil 2T/M \rceil \geq 2T/M$, monotonicity in the first argument (which holds for any sensible PAC bound) implies

$$\min_{\theta \in \Theta} \epsilon_\theta\left(\frac{2T}{M}, \frac{1}{2}\right) \;\leq\; \min_{\theta \in \Theta} \epsilon_\theta(L, 1/2) \;=\; 0,$$

and thus the right-hand side of (3) equals 0 on $\mathcal{M}_i$. Consequently, achieving (3) with probability $> 1/2$ would require outputting an *optimal* policy with probability $> 1/2$ on $\mathcal{M}_i$.

Step 4: A budget constraint on "fully trained" configurations. Consider any black-box meta-algorithm HPO with total budget T. Let N_θ denote the (random) number of episodes that HPO allocates to configuration θ. Then $\sum_{\theta=1}^{M} N_\theta = T$. Define the set of configurations that receive at least L episodes:

$$S \coloneqq \{\theta \in \Theta : \; N_\theta \geq L\}.$$

Since each $\theta \in S$ consumes at least L episodes,

$$|S|\, L \;\leq\; \sum_{\theta \in S} N_\theta \;\leq\; \sum_{\theta=1}^{M} N_\theta \;=\; T,$$

hence

$$|S| \;\le\; \left\lfloor \frac{T}{L} \right\rfloor.$$

By the choice $L = \lceil 2T/M \rceil$, we have $L \ge 2T/M$ and therefore

$$\frac{T}{L} \;\le\; \frac{T}{2T/M} \;=\; \frac{M}{2}, \qquad \text{so} \qquad |S| \;\le\; \left\lfloor \frac{M}{2} \right\rfloor.$$

Step 5: Success on $\mathcal{M}_i$ requires $i \in S$. Fix $i \in \{1, \dots, M\}$. By construction of the base agents, if $N_\theta < L$ then $\mathcal{A}_\theta$ never switches away from action 0 and thus cannot output the optimal policy "always play i" on $\mathcal{M}_i$. Moreover, if $N_\theta \ge L$ then $\mathcal{A}_\theta$ outputs "always play θ", which is optimal on $\mathcal{M}_i$ if and only if $\theta = i$. Since HPO must output a policy produced by one of the base agents, it can output the optimal policy on $\mathcal{M}_i$ only if $\mathcal{A}_i$ has been run for at least L episodes, i.e., only if $i \in S$. Hence,

$$\Pr(\hat{\pi} \text{ is optimal on } \mathcal{M}_i) \;\le\; \Pr(i \in S). \tag{11}$$

Step 6: Averaging over i yields a worst-case instance. Now sample the environment index I uniformly from $\{1, \dots, M\}$, independent of HPO's internal randomness. Conditioned on the realized set S, we have

$$\Pr(I \in S \mid S) \;=\; \frac{|S|}{M} \;\le\; \frac{1}{2},$$

since $|S| \le \lfloor M/2 \rfloor$. Taking expectation over S,

$$\Pr(I \in S) \;=\; \mathbb{E}[\Pr(I \in S \mid S)] \;\le\; \frac{1}{2}.$$

Combining with (11) and the law of total expectation,

$$\Pr(\hat{\pi} \text{ is optimal on } \mathcal{M}_I) \;\le\; \frac{1}{2}.$$

Therefore, there exists some $i^\star \in \{1, \dots, M\}$ such that

$$\Pr(\hat{\pi} \text{ is optimal on } \mathcal{M}_{i^\star}) \;\le\; \frac{1}{2}.$$

On $\mathcal{M}_{i^\star}$, the benchmark in (3) equals 0 by (10), so satisfying (3) is equivalent to outputting an optimal policy. Hence,

$$\Pr\left(V^\star_{\mathcal{M}_{i^\star}} - V^{\hat{\pi}}_{\mathcal{M}_{i^\star}} \;\le\; \min_{\theta \in \Theta} \epsilon_\theta\left(\frac{2T}{M}, \frac{1}{2}\right) \right) \;\le\; \frac{1}{2},$$

which shows that no black-box HPO algorithm can achieve (3) with probability strictly greater than $1/2$ on all MDPs. This completes the proof. □

Software and Data

Code used in this work can be found at https://github.com/Rotkaeppchen33/The-Statistical-Cost-of-Hyperparameter-Tuning-in-Reinforcement-Learning.git