THE COST OF HYPERPARAMETER TUNING IN ONLINE REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

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 can misrepresent 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 specific to the online learning setting. 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 suggest the value of a tuning-aware evaluation perspective 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.

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, 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 other Theorems and Corollaries.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

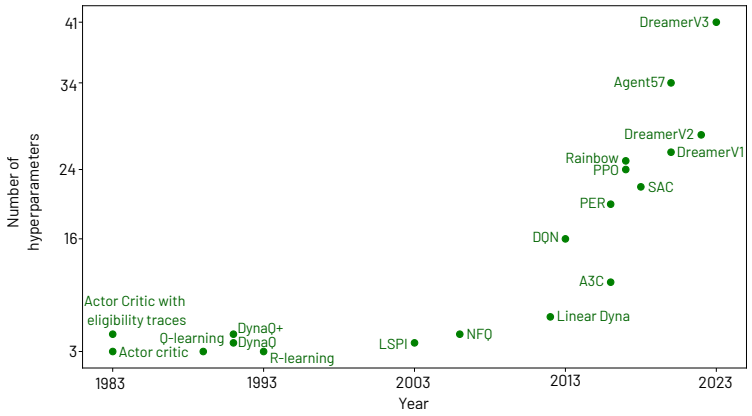


Figure 1: The number of hyperparameters in reinforcement learning algorithms proposed over the last decades (Adkins et al., 2024).

When tuning cost is non-negligible, we propose new evaluation protocols that account for the HPO overhead. These enable fairer comparisons across algorithms and support practical questions such as:

- Is Algorithm A more efficient than Algorithm B while HPO cost is accounted for?
- 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 model selection 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;

Algorithm 1 HPO via Data Splitting in SL

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

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 offline settings like supervised learning. In those settings, data splitting strategies such as cross-validation enables HPO seaminglessly. We begin by formalizing this baseline in SL, to be used to contrast with the RL setting in the following section.

3.1 The HPO Cost in Supervised Learning

For clarity of discussion, we focus on the regression problem. Let $D = \{(x, y)_i\}_{i=1}^N$ be an i.i.d. dataset sampled from a distribution P. Assume a SL 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:

$$\mathbb{E}_{(x,y)\sim P}[(\mathcal{A}(D)(x) - y)^2] - \min_{f\in\mathcal{F}} \mathbb{E}[(f(x) - y)^2] \leq \epsilon(N, \delta). \tag{1}$$

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

Many SL algorithms achieve an optimality gap of the form:

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

where $C_\mathcal{F}$ denotes the complexity of $\mathcal{F}$. An alternative metric to quantify learning efficiency is the **sample complexity**, i.e. the number of samples it requires to reach a certain optimality gap. For instance, equation 2 would translate to a sample complexity of

$$N(\epsilon, \delta) = O\left(\frac{C_\mathcal{F} \log(1/\delta)}{\epsilon^2}\right). \tag{3}$$

In SL, HPO can be implemented using a 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:*

$$\mathbb{E}[(\hat{f}(x) - y)^2] - \min_{f\in\mathcal{F}} \mathbb{E}[(f(x) - y)^2] \leq 2 \min_{\theta\in\Theta} \epsilon_\theta(N/2, \delta/|\Theta|) \approx \tilde{O}\left(\min_{\theta\in\Theta} \sqrt{\frac{\log|\Theta| C_{\mathcal{F}_\theta}}{N}}\right). \tag{4}$$

In other words, Algorithm 1 is equivalent to 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 **algorithm-agnostic 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, as we shall see next, in online RL this cost can become substantially larger.

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

$$\max_{\pi} V^{\pi}_{\mathcal{M}} - V^{\hat{\pi}}_{\mathcal{M}} \leq \epsilon_{\mathcal{M}}(T,\delta), \tag{5}$$

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

Naively adapting Algorithm 1 to the online RL setting implies training a policy for each $\theta \in \Theta$ using $T/|\Theta|$ episodes, then evaluating each and selecting the best. This is because data can no longer be shared seamlessly across different RL instances. Analogous to Theorem 3.2, this procedure yields the following PAC bound:

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

$$\max_{\pi} V^{\pi} - V^{\hat{\pi}} \leq \min_{\theta\in\Theta} \epsilon_{\theta}(T/|\Theta|, \delta/|\Theta|) \approx \tilde{O}\left(\min_{\theta\in\Theta} \sqrt{\frac{|\Theta| C_{\mathcal{F}_\theta}}{T}}\right). \tag{6}$$

We can thus observe an exponential gap between Theorem 3.4 and Theorem 3.2, where the multiplicative blow-up induced by HPO is $O(\log|\Theta|)$ in SL but is $O(|\Theta|)$ in RL. In the simple setting of a grid search over m hyperparameters, $|\Theta| \coloneqq M = O(\exp(m))$ implies that tuning m hyperparameters incurs an $\Omega(\exp(m))$ overhead in this algorithm-agnostic HPO procedure. We do note that this result should be interpreted as a characterization of the worst-case HPO cost in online RL rather than a complete model of all practical tuning workflows. Taking one step further, we can show that the above inefficiency is not just a weakness of this naive algorithm, but rather a necessary cost for any algorithm-agnostic HPO procedure, formally defined below.

Definition 3.5 (Algorithm-agnostic HPO in RL)**.** A *algorithm-agnostic* 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. 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. In particular, there is no data sharing across base algorithms.

Most practical HPO procedures satisfy this definition, i.e. no data sharing across runs with different hyperparameters. With that, we have the following result:

Theorem 3.6 (Algorithm-agnostic HPO is inefficient in online RL)**.** *There exists a set of base learners indexed by Θ with $M \coloneqq |\Theta| \geq 2$ and a family of episodic MDPs $\mathcal{M}_\Theta$, such that no algorithm-agnostic meta-algorithm HPO can output a policy $\hat{\pi}$ satisfying*

$$V^{\star}_{\mathcal{M}} - V^{\hat{\pi}}_{\mathcal{M}} \;\leq\; \min_{\theta\in\Theta} \epsilon_{\mathcal{M},\theta}\left(\frac{2T}{M}, \frac{1}{2}\right) \tag{7}$$

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

The formal proof is deferred to the appendix. This lower bound builds on two key observations: First, since no data sharing is allowed between base agents, 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, e.g. the algorithms that implement the explore-then-commit strategy. Therefore, no data-splitting strategy can guarantee to consistently outperform $T/|\Theta|$ 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 algorithm-agnostic HPO algorithm can achieve a regret bound better than*

$$\min_{\theta\in\Theta} \frac{|\Theta|}{2} Reg_\theta(2T/|\Theta|, 1/2) \tag{8}$$

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 may be important to account for when tuning effort is part of the evaluation objective.

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 2a. Such practices can favor 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, as well as answering HPO related questions in a quantitative manner that is previous not possible.

4 HPO-aware Evaluation for Online RL

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 provide a complementary, protocol-dependent lens for comparing algorithms under tuning constraints.

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

$$|\Theta| \times \min_{\theta\in\Theta} T_\theta(\epsilon), \tag{9}$$

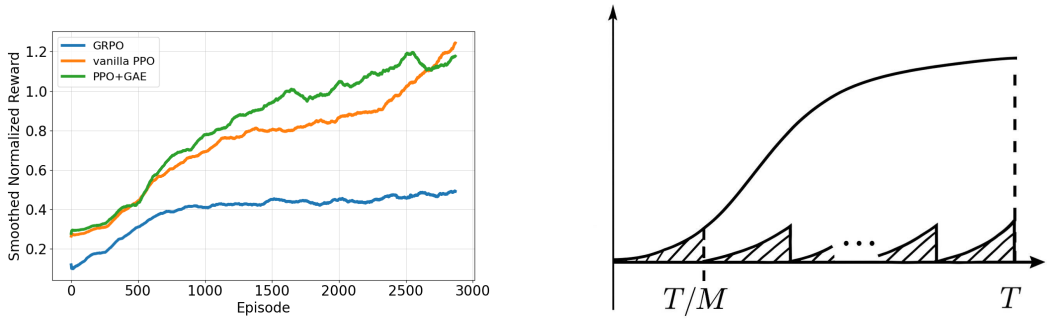
where $T_\theta(\epsilon)$ is the number of episodes needed for $\mathcal{A}_\theta$ to produce a policy $\hat{\pi}$ such that $\max_\pi V^\pi - V^{\hat{\pi}} \leq \epsilon$.

This metric captures the HPO cost in environment interactions needed to find a near-optimal policy. Notice that equation 9 resembles the 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. In our experiments, the threshold ϵ is chosen as the 90th percentile of episodic returns aggregated over all configurations and algorithms.

While SC measures the data needed to achieve competent performance, many real-world applications also care about cumulative reward during learning. 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:

$$\max_{\theta\in\Theta} |\Theta| \times \sum_{t=1}^{T/|\Theta|} V^{\pi_{\theta,t}}, \tag{10}$$



(a) A typical comparison across RL algorithms.

(b) Illustration of Effective AUC.

Figure 2: (a) 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. (b) An illustration of the Effective AUC, represented by the shaded area under the curve.

Table 1: Hyperparameter set for each algorithm.

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

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

The Effective AUC is designed to measure learning performance under a fixed total training budget T. Intuitively, equation 10 calculates the cumulative rewards achieved by the best configuration over $T/|\Theta|$ episodes, then multiplying it by $|\Theta|$, as illustrated in Figure 2b. When the hyperparameter space $|\Theta|$ is larger, each configuration is allocated fewer episodes, making early learning behavior critical, while $|\Theta|$ ensures comparability across different hyperparameter set sizes by encoding the cost of tuning larger hyperparameter spaces. 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.

Taken together, these two metrics offer alternative ways to evaluate what it means for an RL algorithm to be efficient. Rather than asking "how well does this algorithm perform after tuning?", we ask "how well does the algorithm perform if tuning must be done online?"

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).

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

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

All algorithms optimize the same clipped surrogate PPO objective:

$$L(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right], \tag{11}$$

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 Table 1. The hyperparameter ranges are picked around the reported best performing hyperparameter configuration for each algorithm and thus can 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 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 in Definition 4.2. We then average the reward-vs-T curve across 10 trials for each configuration, collect all reward values from the averaged trajectories over the full grid, and compute the 5% and 95% quantiles within each environment, denoted by $p_5(e)$ and $p_{95}(e)$.

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:

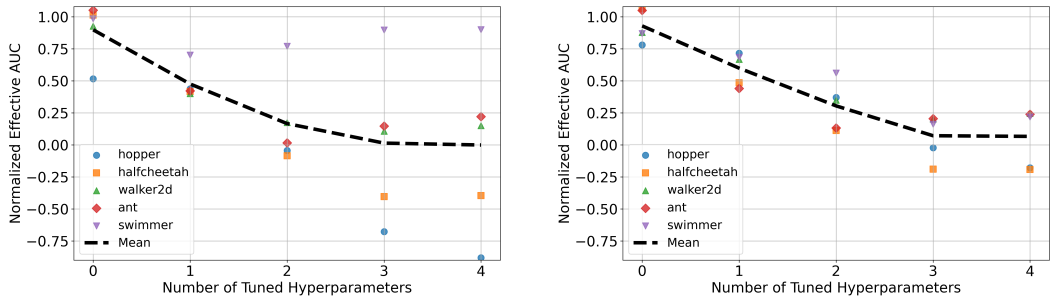
$$\bar{R}(\tau) = \frac{R(\tau) - p_5(e)}{p_{95}(e) - p_5(e)}, \tag{12}$$

This normalization ensures fair metric comparison across tasks. We then calculate the effective SC and effective AUC from the normalized averaged curves. In this fixed-step setting, we record the resulting total number of trajectories in each 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

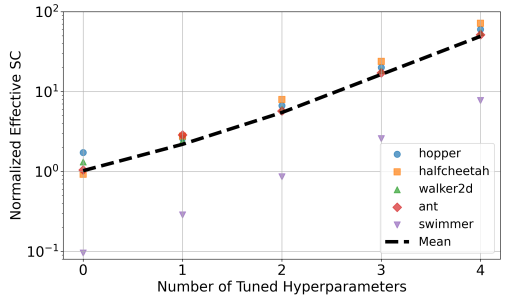
Consider a practitioner deploying RL in a new task similar to MuJoCo. They face a practical question: which hyperparameters should be tuned online, and which can be fixed based on prior knowledge or auxiliary environments? Tuning all hyperparameters may yield the best result in hindsight, but it also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

For each $m = 0, 1, 2, 3, 4$, we consider hyperparameter spaces of size $M = 1, 3^1, 3^2, 3^3, 3^4$, corresponding to zero through four tunable hyperparameters. Starting from the full grid of $M = 3^4$

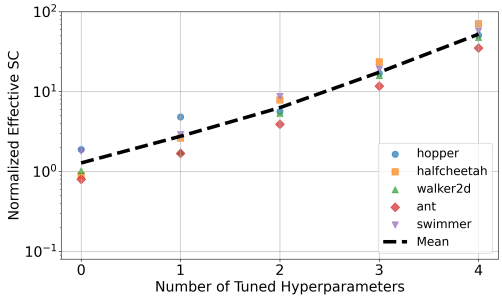


(a) Effective AUC for PPO+GAE

(b) Effective AUC for PPO+ADVN



(c) Effective SC for PPO+GAE



(d) Effective SC for PPO+ADVN

Figure 3: Normalized effective AUC&SC vs. number of tuned hyperparameters.

configurations, we first identify the configuration with the highes average effective AUC (respectively lowest SC) across all environments and use it as the fixed reference point. We then progressively release tunable hyperparameters one by one, following a dynamic order determined by their influence on learning performance, while all unreleased hyperparameters retain their values from the reference configuration. The full procedure is summarized in Algorithm 2 in the Appendix.

For each environment, we evaluate all $M = 3^4$ original configurations using the full trajectory budget $T/1$ and extract 5th and 95th quantile values as normalization anchors. For effective AUC, we define

$$\text{Normalized effective AUC} = \frac{AUC - p_{5,\text{AUC}}(e)}{p_{95,\text{AUC}}(e) - p_{5,\text{AUC}}(e)}, \tag{13}$$

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

Since effective SC can vary substantially across configurations in one environment, we use

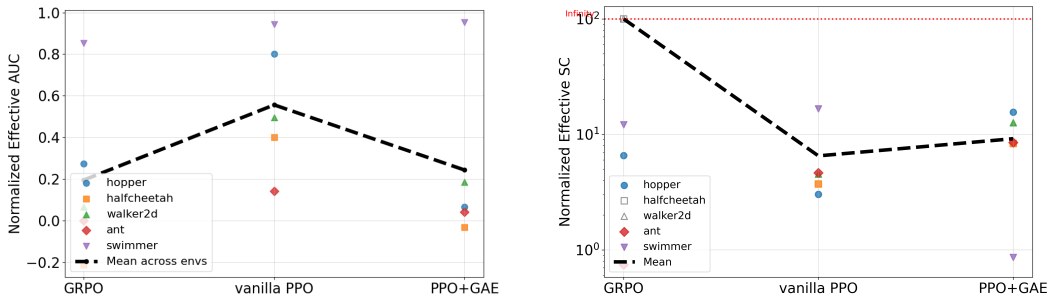
$$\text{Normalized effective SC} = \frac{SC}{p_{5,\text{SC}}(e)}, \tag{14}$$

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

As 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 m, therefore, multiplying by the varying scaling factor leads to a positive correlation between effective SC and m.

We then average the normalized effective AUC (respectively SC) across all five environments. The configuration achieving the highest average normalized effective AUC (respectively lowest SC) is selected as the optimal setup for its corresponding m. This determines which hyperparameters should be fixed across environments and which should be tuned per environment; the resulting choices are shown in Table 2. The performance of the best configuration for each m is shown in Figure 3, where the dashed line indicates the mean normalized effective AUC (respectively SC) across environments.

Figure 3 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.



(a) Effective AUC

(b) Effective SC

Figure 4: Normalized Effective AUC/SC vs different algorithms. Infinite sample complexity values are represented by (10^2) for visualization purposes.

This suggests that under this evaluation protocol, tuning fewer hyperparameters 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), and `PPO+GAE` (4).

In standard RL benchmarking, algorithms are often compared based on the performance of their best hyperparameter configuration, as shown in Figure 2a. This comparison 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 4), the ranking under this tuning-aware protocol changes substantially. 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 an often underrepresented aspect of RL research: the statistical and computational cost of hyperparameter optimization. Unlike SL, which incurs a $\log(|\Theta|)$ overhead, online RL can suffer a $|\Theta|$ penalty. Our results show that ignoring this cost can change algorithm comparisons and lead to a different interpretation of empirical efficiency. To address this, we introduce two tuning-aware metrics, *Effective Sample Complexity* and *Effective AUC*, that quantify an algorithm's learning efficiency while accounting for the cost of hyperparameter tuning. We show on MuJoCo that many hyperparameters are not worth tuning, as their tuning cost outweighs the resulting benefit. Consequently, more complex algorithms do not always outperform simpler baselines when comparisons account fairly for hyperparameter tuning. These results motivate routine HPO-aware evaluation and the development of **hyperparameter-free algorithms**. A broader discussion of the scope and limitations is provided in the appendix.

AI USE STATEMENT

In this work, we used generative AI tools to assist with reviewing and refining the presentation of parts of the mathematical proofs. We did not use generative AI tools to generate synthetic datasets, develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, clean or reformat datasets, or interpret results. Translation and qualitative or thematic data analysis were not applicable to this work. Additionally, we used generative AI tools for improving the readability and grammar of the manuscript. All AI-assisted content was independently reviewed by the authors. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not raise ethical considerations that require specific discussion.

REPRODUCIBILITY STATEMENT

To support reproducibility, we provide implementation details and experimental settings in the main text and appendix. The code used for the experiments is made available in the *Software and Data* section of the appendix.

REFERENCES

Yasin Abbasi-Yadkori, Aldo Pacchiano, and My Phan. Regret balancing for bandit and rl model selection. *arXiv preprint arXiv:2006.05491*, 2020.

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

Alekh Agarwal, Haipeng Luo, Behnam Neyshabur, and Robert E Schapire. Corralling a band of bandit algorithms. In *Conference on Learning Theory*, pp. 12–38. PMLR, 2017.

Raman Arora, Teodor V Marinov, and Mehryar Mohri. Corralling stochastic bandit algorithms. *arXiv preprint arXiv:2006.09255*, 2020.

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

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

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

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

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

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

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

Ashok Cutkosky, Christoph Dann, Abhimanyu Das, Claudio Gentile, Aldo Pacchiano, and Manish Purohit. 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.

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

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

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

Theresa Eimer, Carolin Benjamins, and Marius Lindauer. Hyperparameters in Contextual RL are Highly Situational. *CoRR*, abs/2212.10876, 2022.

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

Dylan Foster, Claudio Gentile, Mehryar Mohri, and Julian Zimmert. Adapting to misspecification in contextual bandits. In *Advances in Neural Information Processing Systems*, 2020.

Jörg KH Franke, Gregor Köhler, André Biedenkapp, and Frank Hutter. Sample-Efficient Automated Deep Reinforcement Learning. In *International Conference on Learning Representations (ICLR)*, 2021.

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

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining Improvements in Deep Reinforcement Learning. In *Conference on Artificial Intelligence (AAAI)*, 2018.

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

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

Jonathan N Lee, Aldo Pacchiano, Vidya Muthukumar, Weihao Kong, and Emma Brunskill. Online model selection for reinforcement learning with function approximation. *arXiv preprint arXiv:2011.09750*, 2020.

Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

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

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-Level Control through Deep Reinforcement Learning. *Nature*, 518:529–533, 2015.

Johan Obando-Ceron, João GM Araújo, Aaron Courville, and Pablo Samuel Castro. On the consistency of hyper-parameter selection in value-based deep reinforcement learning. *arXiv preprint arXiv:2406.17523*, 2024.

Maillard Odalric and Rémi Munos. 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.

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

Aldo Pacchiano, Christoph Dann, Claudio Gentile, and Peter Bartlett. Regret bound balancing and elimination for model selection in bandits and rl. *arXiv preprint arXiv:2012.13045*, 2020.

Andrew Patterson, Samuel Neumann, Martha White, and Adam White. Empirical Design in Reinforcement Learning. *CoRR*, abs/2304.01315, 2023.

Andrew Patterson, Samuel Neumann, Raksha Kumaraswamy, Martha White, and Adam M White. The Cross-Environment Hyperparameter Setting Benchmark for Reinforcement Learning. In *Reinforcement Learning Conference (RLC)*, 2024.

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

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

Tianhe Yu, Aviral Kumar, Rafael Rafailov, Aravind Rajeswaran, Sergey Levine, and Chelsea Finn. Combo: Conservative offline model-based policy optimization. *Advances in neural information processing systems*, 34:28954–28967, 2021.

A Proof of Theorem 3.2

Let

$$L(f) \coloneqq \mathbb{E}_{(x,y)\sim P}\big[(f(x)-y)^2\big].$$

Proof. Split the dataset into D_{train} and D_{val}, each of size $N/2$. For each $\theta \in \Theta$, let

$$f_\theta = \mathcal{A}_\theta(D_{\text{train}})$$

be the predictor trained using hyperparameter θ, and define the empirical validation loss

$$\widehat{L}_{\text{val}}(f) \coloneqq \frac{2}{N} \sum_{(x,y)\in D_{\text{val}}} (f(x)-y)^2.$$

Algorithm 1 returns

$$\hat{\theta} \in \arg\min_{\theta\in\Theta} \widehat{L}_{\text{val}}(f_\theta), \qquad \hat{f} \coloneqq f_{\hat{\theta}}.$$

For each fixed $\theta \in \Theta$, since $\mathcal{A}_\theta$ is a PAC learner trained on $N/2$ samples, with probability at least $1-\delta/(2|\Theta|)$,

$$L(f_\theta) - \inf_{f\in\mathcal{F}} L(f) \le \epsilon_\theta(N/2, \delta/(2|\Theta|)).$$

Applying a union bound over Θ, with probability at least $1-\delta/2$, the above holds simultaneously for all $\theta \in \Theta$.

Now let

$$\theta^\star \in \arg\min_{\theta\in\Theta} \epsilon_\theta(N/2, \delta/(2|\Theta|)).$$

Next, because the candidates $\{f_\theta\}_{\theta\in\Theta}$ are fixed after training on D_{train}, standard uniform concentration over the validation split implies that, with probability at least $1-\delta/2$,

$$\sup_{\theta\in\Theta}\left|\widehat{L}_{\text{val}}(f_\theta) - L(f_\theta)\right| \le \alpha,$$

where

$$\alpha = O\left(\sqrt{\frac{\log|\Theta| + \log(1/\delta)}{N}}\right).$$

Therefore, with probability at least $1-\delta$, both events hold simultaneously.

Since $\hat{\theta}$ minimizes $\widehat{L}_{\text{val}}(f_\theta)$, we have

$$\widehat{L}_{\text{val}}(f_{\hat{\theta}}) \le \widehat{L}_{\text{val}}(f_{\theta^\star}),$$

and therefore

$$\begin{aligned} L(\hat{f}) &= L(f_{\hat{\theta}}) \\ &\le \widehat{L}_{\text{val}}(f_{\hat{\theta}}) + \alpha \\ &\le \widehat{L}_{\text{val}}(f_{\theta^\star}) + \alpha \\ &\le L(f_{\theta^\star}) + 2\alpha. \end{aligned}$$

Combining this with the PAC guarantee for $\theta^\star$, we obtain

$$L(\hat{f}) - \inf_{f\in\mathcal{F}} L(f) \le \epsilon_{\theta^\star}(N/2, \delta/(2|\Theta|)) + 2\alpha.$$

Finally, under the standard PAC rate

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

the validation concentration term α is of the same order up to constants and logarithmic factors, so it can be absorbed into the leading term. Hence

$$L(\hat{f}) - \inf_{f\in\mathcal{F}} L(f) \le 2\min_{\theta\in\Theta}\epsilon_\theta(N/2, \delta/|\Theta|)$$

up to constants, and therefore

$$L(\hat{f}) - \inf_{f\in\mathcal{F}} L(f) \approx \tilde{O}\left(\min_{\theta\in\Theta}\sqrt{\frac{\log|\Theta|\, C_{\mathcal{F}_\theta}}{N}}\right).$$

□

B Proof of Theorem 3.4

Proof. The naive RL HPO procedure allocates $T/|\Theta|$ episodes to each base agent $\mathcal{A}_\theta$, runs all agents independently, and then returns the policy produced by the empirically best configuration.

For a fixed $\theta\in\Theta$, since $\mathcal{A}_\theta$ is a PAC agent, when run for $T/|\Theta|$ episodes it returns a policy π_θ satisfying

$$\max_\pi V^\pi - V^{\pi_\theta} \le \epsilon_\theta(T/|\Theta|, \delta/|\Theta|)$$

with probability at least $1-\delta/|\Theta|$.

Applying a union bound over all $\theta\in\Theta$, we obtain that with probability at least

$$1 - \sum_{\theta\in\Theta}\frac{\delta}{|\Theta|} = 1-\delta,$$

the above guarantee holds simultaneously for every $\theta \in \Theta$. Therefore, on this event, there exists in particular the best hyperparameter

$$\theta^\star \in \arg\min_{\theta\in\Theta} \epsilon_\theta(T/|\Theta|, \delta/|\Theta|)$$

such that

$$\max_\pi V^\pi - V^{\pi_{\theta^\star}} \le \min_{\theta\in\Theta} \epsilon_\theta(T/|\Theta|, \delta/|\Theta|).$$

Since the naive HPO procedure returns the best policy among the trained configurations, its output $\hat{\pi}$ is no worse than $\pi_{\theta^\star}$. Hence,

$$\max_\pi V^\pi - V^{\hat{\pi}} \le \min_{\theta\in\Theta} \epsilon_\theta(T/|\Theta|, \delta/|\Theta|).$$

Finally, substituting the standard PAC-RL rate

$$\epsilon_\theta(T, \delta) = O\left(\sqrt{\frac{C_{\mathcal{F}_\theta} \log(1/\delta)}{T}}\right)$$

yields

$$\max_\pi V^\pi - V^{\hat{\pi}} \approx \tilde{O}\left(\min_{\theta\in\Theta} \sqrt{\frac{|\Theta|\, C_{\mathcal{F}_\theta}}{T}}\right).$$

□

C Proof of Theorem 3.6

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

$$L \coloneqq \left\lfloor \frac{2T}{M} \right\rfloor.$$

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 algorithm-agnostic HPO succeeds on at most one $\mathcal{M}_i$ with probability at most $1/2 + 1/M$.

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

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

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 $\theta \in \{1, \dots, M\}$ 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 $n_\theta \ge L$, 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 $i \in \{1, \dots, M\}$. 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,

$$\min_{\theta \in \Theta} \epsilon_\theta(L, 1/2) = 0 \qquad \text{on } \mathcal{M}_i. \tag{15}$$

Since $L = \lfloor 2T/M \rfloor \leq 2T/M$ and each valid PAC optimality-gap function $\epsilon_\theta(\cdot, 1/2)$ is non-increasing in its first argument, we have

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

Since optimality gaps are nonnegative, it follows that

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

Thus the right-hand side of equation 7 equals 0 on $\mathcal{M}_i$.

Consequently, achieving equation 7 with probability $> 1/2 + 1/M$ would require outputting an *optimal* policy with probability $> 1/2 + 1/M$ on $\mathcal{M}_i$.

Step 4: A budget constraint on "fully trained" configurations. We focus on the regime $T > M^2$. Consider any algorithm-agnostic 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

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

By the choice $L = \lfloor 2T/M \rfloor$, we have

$$L \geq \frac{2T}{M} - 1.$$

Since $T > M^2$, at most $\frac{M}{2} + 1$ base learners can be allocated at least L episodes, i.e.,

$$|S| \leq \left\lfloor \frac{M}{2} \right\rfloor + 1.$$

Step 5: Success on $\mathcal{M}_i$ requires $i \in S$. Fix $i \in \{1, \ldots, M\}$. 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 $N_\theta \geq L$ 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,

$$\Pr\left(\hat{\pi} \text{ is optimal on } \mathcal{M}_i\right) \ \leq \ \Pr(i \in S). \tag{16}$$

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

$$\Pr(I \in S \mid S) = \frac{|S|}{M} \leq \frac{\lfloor M/2 \rfloor + 1}{M} \leq \frac{1}{2} + \frac{1}{M}.$$

Taking expectation over S,

$$\Pr(I \in S) = \mathbb{E}\left[\Pr(I \in S \mid S)\right] \leq \frac{1}{2} + \frac{1}{M}.$$

Combining with equation 16 and the law of total expectation,

$$\Pr\left(\hat{\pi} \text{ is optimal on } \mathcal{M}_I\right) \leq \frac{1}{2} + \frac{1}{M}.$$

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

$$\Pr\left(\hat{\pi} \text{ is optimal on } \mathcal{M}_{i^\star}\right) \leq \frac{1}{2} + \frac{1}{M}.$$

On $\mathcal{M}_{i^\star}$, the benchmark in equation 7 equals 0 by equation 15, so satisfying equation 7 is equivalent to outputting an optimal policy. Hence,

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

which shows that no algorithm-agnostic HPO algorithm can achieve equation 7 with probability strictly greater than $1/2 + 1/M$ on all MDPs. This completes the proof.

□

D Proof of Corollary 3.7

Proof. Let $M \coloneqq |\Theta|$ and, for simplicity, assume $2T/M \in \mathbb{N}$. Set

$$L \coloneqq \frac{2T}{M}.$$

We reuse the construction from Theorem 3.6. Namely, for each $i \in [M]$, define an episodic MDP $\mathcal{M}_i$ with horizon $H = 1$, action set $\{0, 1, \ldots, M\}$, and deterministic reward

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

Thus the unique optimal action in $\mathcal{M}_i$ is i, and every other action has optimality gap 1.

For each $\theta \in [M]$, define the base agent $\mathcal{A}_\theta$ as follows: it plays action 0 for its first L executions, and then switches permanently to action θ. Therefore, on $\mathcal{M}_i$, every episode before the switch incurs unit gap, while after the switch the gap is 0 if $\theta = i$ and 1 otherwise.

In particular, if $\mathcal{A}_i$ is run alone for L episodes on $\mathcal{M}_i$, then its cumulative regret is exactly

$$\mathrm{Reg}_i(L, 1/2) = L.$$

Hence on $\mathcal{M}_i$,

$$\min_{\theta \in \Theta} \mathrm{Reg}_\theta(L, 1/2) = L.$$

Now consider any algorithm-agnostic HPO meta-algorithm with total budget T. Let N_θ be the number of episodes allocated to configuration θ, and define

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

Since $\sum_{\theta=1}^{M} N_\theta = T$, we have

$$|S| L \leq T, \qquad \text{so} \qquad |S| \leq \frac{T}{L} = \frac{M}{2}.$$

Fix an instance $\mathcal{M}_i$. If $i \notin S$, then the correct agent $\mathcal{A}_i$ is never trained for L episodes, so it always plays action 0. Any other fully trained agent $\mathcal{A}_\theta$ with $\theta \neq i$ plays action $\theta \neq i$. Therefore, throughout all T episodes, the meta-algorithm never executes the optimal action i, and thus incurs unit gap in every episode. Hence

$$\mathrm{Reg}_{\mathsf{HPO}}(T) = T \qquad \text{whenever } i \notin S.$$

Equivalently,

$$\Pr\big(\mathrm{Reg}_{\mathsf{HPO}}(T) < T \text{ on } \mathcal{M}_i\big) \leq \Pr(i \in S).$$

Now sample I uniformly from $[M]$. Conditioning on S,

$$\Pr(I \in S \mid S) = \frac{|S|}{M} \leq \frac{1}{2}.$$

Taking expectation gives

$$\Pr(I \in S) \leq \frac{1}{2}.$$

Therefore,

$$\Pr\big(\mathrm{Reg}_{\mathsf{HPO}}(T) < T \text{ on } \mathcal{M}_I\big) \leq \frac{1}{2}.$$

So there exists some $i^\star \in [M]$ such that

$$\Pr\big(\mathrm{Reg}_{\mathsf{HPO}}(T) < T \text{ on } \mathcal{M}_{i^\star}\big) \leq \frac{1}{2}.$$

Finally, since $L = 2T/M$ and

$$\min_{\theta \in \Theta} \mathrm{Reg}_\theta(L, 1/2) = L,$$

we obtain

$$T = \frac{M}{2}\, L = \frac{|\Theta|}{2} \min_{\theta \in \Theta} \mathrm{Reg}_\theta(2T/|\Theta|, 1/2).$$

Hence no algorithm-agnostic HPO algorithm can achieve a regret bound strictly smaller than

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

with probability greater than $1/2$. □

E EFFECTIVE AUC AND SC

Algorithm 2 Progressive Hyperparameter Release for Tuning-Aware Evaluation

Input: Environments $\mathcal{E}$, algorithm a, full-grid configurations Θ_{full}, hyperparameters $\mathcal{H} = \{h_1, h_2, h_3, h_4\}$, Metric $\in \{\text{AUC}, \text{SC}\}$, where AUC and SC denote Effective AUC and Effective SC, respectively
Compute $p_{5,\text{AUC}}(e)$ and $p_{95,\text{AUC}}(e)$ from the full-grid effective AUC values
Compute $p_{5,\text{SC}}(e)$ from the full-grid effective SC values before scaling by M
Normalize effective AUC and SC of configuration θ in environment e ($\text{NormMetric}(e, \theta)$)
Compute normalized effective AUC/SC values $\text{NormMetric}(e, \theta)$ for all $e \in \mathcal{E}$ and $\theta \in \Theta_{\text{full}}$
Choose the reference configuration θ_{ref} that is best on average over $\mathcal{E}$
$U_0 \leftarrow \emptyset$ ▷ released hyperparameters
$\Gamma_0 \leftarrow \{\theta_{\text{ref}}\}$
for $m = 1, 2, 3, 4$ **do**
  for $h \in \mathcal{H} \setminus U_{m-1}$ **do**
    $\Gamma_h \leftarrow \{\theta : \theta$ varies on $U_{m-1} \cup \{h\}$ and matches θ_{ref} otherwise$\}$
    for $e \in \mathcal{E}$ **do**
      if Metric $=$ AUC **then**
        $v_e(h) \leftarrow \max_{\theta \in \Gamma_h} \text{NormMetric}(e, \theta)$
      else
        $v_e(h) \leftarrow \min_{\theta \in \Gamma_h} \text{NormMetric}(e, \theta)$
      end if
    end for
    $s(h) \leftarrow \dfrac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} v_e(h)$
  end for
  if Metric $=$ AUC **then**
    $h_m^\star \leftarrow \arg\max_{h \in \mathcal{H} \setminus U_{m-1}} s(h)$
  else
    $h_m^\star \leftarrow \arg\min_{h \in \mathcal{H} \setminus U_{m-1}} s(h)$
  end if
  $U_m \leftarrow U_{m-1} \cup \{h_m^\star\}$
  $\Gamma_m \leftarrow \{\theta : \theta$ varies on U_m and matches θ_{ref} otherwise$\}$
end for
for $m = 0, 1, 2, 3, 4$ **do**
  if Metric $=$ AUC **then**
    $\theta_m^\star \leftarrow \arg\max_{\theta \in \Gamma_m} \dfrac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \text{NormMetric}(e, \theta)$
  else
    $\theta_m^\star \leftarrow \arg\min_{\theta \in \Gamma_m} \dfrac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \text{NormMetric}(e, \theta)$
  end if
  Choose $\theta_m^\star$ as the best configuration in Γ_m averaged over $\mathcal{E}$
end for
Return: released sets $\{U_m\}_{m=0}^4$, candidate spaces $\{\Gamma_m\}_{m=0}^4$, and selected setups $\{\theta_m^\star\}_{m=0}^4$

F SCOPE AND LIMITATIONS

Our theoretical results are worst-case and are derived in an algorithm-agnostic online RL setting, so they do not model all practical tuning workflows. The proposed metrics are designed for settings in which tuning budget is part of the evaluation criterion, and they should not be interpreted as universal measures of algorithm quality. More generally, the relevance of tuning cost depends on the deployment context. In particular, when tuning is performed once and then shared across many deployments, a larger upfront tuning cost may be justified, and alternative evaluation criteria may be more appropriate. Our goal is therefore not to discourage tuning, but to improve transparency by making its cost explicit when that cost is relevant to the decision maker.

G Broader Impacts

This work may have a positive social impact by encouraging more transparent and sample-efficient evaluation of RL systems, especially in settings where environment interaction is costly and computational resources are limited. By making tuning costs explicit, the proposed perspective can also help reduce wasteful experimentation and make empirical comparisons more honest and reproducible.

At the same time, our metrics should not be used as a universal replacement for existing evaluation protocols. If applied without care, tuning-aware evaluation could bias practitioners against methods with larger one-time tuning costs even in settings where that upfront cost is acceptable, or encourage under-tuning in applications where reliability matters more than interaction efficiency. We therefore view the proposed metrics as complementary tools whose relevance depends on the deployment objective and risk profile.

H Software and Data

Code used in this work can be found at `https://anonymous.4open.science/r/Cost-D88D/README.md`