

The present work was submitted to the Institute for Data Science in Mechanical Engineering.  
Diese Arbeit wurde vorgelegt am Institut für Data Science im Maschinenbau.

# **Benchmarking Learning-Based Control and Reinforcement Learning**

## **Benchmarking von lernbasierter Regelung und Reinforcement Learning**

Master Thesis  
Masterarbeit

**Presented by / Vorgelegt von**

Tsung Yuan Tseng  
Matr.Nr.: 416164

**Supervised by / Betreut von**

Dr. Siqu Zhou  
(Learning Systems and Robotics Lab)  
Lukas Brunke M.Sc.  
(Learning Systems and Robotics Lab)  
Alexander von Rohr M.Sc.  
(Institute for Data Science in Mechanical Engineering)  
Univ.-Prof. Dr. sc. Sebastian Trimpe  
(Institute for Data Science in Mechanical Engineering)

**1<sup>st</sup> Examiner / 1. Prüfer**

**2<sup>nd</sup> Examiner / 2. Prüfer**

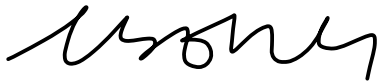
Univ.-Prof. Dr. sc. Angela Schoellig  
(Learning Systems and Robotics Lab)

Aachen, September 29, 2023



### **Eidesstattliche Erklärung**

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne die Benutzung anderer als der angegebenen Hilfsmittel selbstständig verfasst habe; die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe des Literaturzitats gekennzeichnet.

A handwritten signature in black ink, consisting of a series of loops and strokes, likely representing the author's name.

Aachen, den September 29, 2023



## Abstract

To deal with increasingly complex and uncertain environments, both control and reinforcement learning (RL) communities strive to develop decision-making algorithms that are capable of achieving safety and high performance in targeted applications. However, the algorithms from these communities have not been rigorously benchmarked to gauge the progress of this line of development. Hyperparameter optimization (HPO) emerges as a method to identify hyperparameters (HPs) that yield optimal performance for a given algorithm. In our work, HPO serves as an intermediate resort to achieve fair, consistent, and reproducible comparisons. However, standard HPO algorithms do not typically consider highly stochastic evaluations for the HPO objective. That is especially the case for RL algorithms, of which the performance is dramatically influenced by random seeds, making HPO challenging. To elaborate, HPs considered effective in HPO for some seeds may likely perform poorly across unseen seeds. In this thesis, we develop an HPO strategy to address the impact of uncertainty raised by random seeds and HPs. We evaluate our proposed strategy on a range of learning-based control and reinforcement learning algorithms and show that the optimized HPs result in consistent, robust, and reproducible performance under limited computational budget.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Problem Formulation . . . . .	4
1.2	Contributions . . . . .	4
1.3	Structure of the Thesis . . . . .	5
<b>2</b>	<b>Preliminaries</b>	<b>7</b>
2.1	Hyperparameter Optimization . . . . .	7
2.2	Risk Measures . . . . .	8
2.3	Estimators . . . . .	13
2.3.1	Maximum Likelihood Estimator for a Gaussian . . . . .	13
2.3.2	Mean Estimator and CVaR Estimator for Arbitrary Distribution	14
2.4	Decision Making Algorithms . . . . .	14
2.4.1	Reinforcement Learning . . . . .	14
2.4.2	Learning-Based Control . . . . .	18
2.5	Stratified Bootstrap Confidence Intervals . . . . .	20
2.6	Metrics . . . . .	21
<b>3</b>	<b>Related Work</b>	<b>23</b>
<b>4</b>	<b>Method</b>	<b>27</b>
4.1	Reducing Maximization Bias . . . . .	27
4.2	Optimizing over Conditional Value at Risk . . . . .	33
4.3	Summary . . . . .	35
<b>5</b>	<b>Experiments</b>	<b>37</b>
5.1	Synthetic Examples . . . . .	38
5.2	Decision-Making Algorithms . . . . .	44

<b>6 Conclusion and Outlook</b>	<b>53</b>
6.1 Toward Benchmarking Learning-Based Control and Reinforcement Learning . . . . .	54
6.2 Future Work . . . . .	54
<b>List of Algorithms</b>	<b>57</b>
<b>List of Figures</b>	<b>59</b>
<b>List of Tables</b>	<b>61</b>
<b>Bibliography</b>	<b>63</b>



# 1 Introduction

In recent years, robots have been used in many applications, including but not limited to automated inspection [1], transportation [2], and manufacturing [3]. To deal with increasingly complex and uncertain environments, both control and reinforcement learning (RL) communities strive to develop decision-making algorithms that are capable of achieving safety and high performance in targeted applications. In [4], Brunke et al. reviewed the state-of-the-art control and reinforcement learning algorithms for safe robot decision-making under uncertainties. While the survey shows the ever-increasing abilities of decision-making algorithms from both communities, these algorithms have not been rigorously compared on well-defined benchmark problems to gauge the progress of this line of development. One of the first attempts at comparison, *safe-control-gym* (SCG) [5], was developed to speed up robotic controller development and serve as a software platform to potentially compare different methodologies from the two communities. In particular, the work implemented both standard learning-based methods, such as model predictive control with Gaussian processes (GP-MPC) [6], [7], and reinforcement learning (RL) controllers such as the proximal policy optimization (PPO) algorithm [8]. Nonetheless, an important research question for benchmarking is how we compare control algorithms fairly. This is challenging mainly because of (1) different prior assumptions, (2) hyperparameter tuning, (3) and a lack of consistent and fair metrics. This thesis aims to develop tools that facilitate systematic comparisons and support further algorithmic or theoretic designs. To this end, we focus on (2) and we identify the main objective of the thesis: systematic hyperparameter optimization for benchmarking.

For many computational problems, there exist several different algorithms as potential solutions. After selecting an approach, there are additional lower-level choices to be determined to fully specify the algorithm [9], [10]. These choices are often referred to as hyperparameters (HPs) or algorithm configurations [9]. As an example,

consider Newton’s method, a well-known iterative algorithm to solve root-finding problems. Such an algorithm has at least two HPs: (1) a numerical threshold as a stopping criterion and (2) an initial guess for the root. In general, HPs have a great impact on algorithm performance. Consider the aforementioned example, the threshold affects the quality of the solution. In addition, if the initial guess is too far away from the true root, then the algorithm may not converge. The notion of HPs is general. They can be numerical parameters such as a real-valued threshold, categorical parameters (e.g. the type of activations used in neural networks), or binary parameters (e.g. active/inactive algorithm components). Before initiating decision-making algorithms, there exist HPs that need to be determined. The selections of HPs in learning-based control and RL are typically a determining factor of the performance of the algorithms [11]–[14]. The task of finding good HPs that lead to optimal performance of the algorithms can be formulated as hyperparameter optimization (HPO). That is, if the statistic of interest is expected performance, HPO aims to find an optimal set of HPs that maximize the expected performance

$$\mathbf{h}^* = \arg \max_{\mathbf{h} \in \mathcal{H}} \mathbb{E}[\mathbf{G}|\mathbf{h}], \quad (1.1)$$

where  $\mathbf{G}$  is the evaluation metrics of an algorithm,  $\mathbf{h} \in \mathcal{H}$  is the collection of hyperparameters, and  $\mathcal{H}$  is the hyperparameter search space. The optimization is done by training from data  $\{(\mathbf{h}_1, \hat{\mathbf{G}}_{\mathbf{h}_1}), \dots, (\mathbf{h}_M, \hat{\mathbf{G}}_{\mathbf{h}_M})\}$  with  $M$  being the total number of iterations and  $\hat{\mathbf{G}}_{\mathbf{h}_i}$  denoting the approximation for  $\mathbb{E}[\mathbf{G}|\mathbf{h}_i]$  (shorthand  $\mathbb{E}[\mathbf{G}_{\mathbf{h}_i}]$ ) given  $\mathbf{h}_i$  for  $i = \{1, 2, \dots, M\}$ . The estimator  $\hat{\mathbf{G}}_{\mathbf{h}_i}$  is computed based on the samples from the distribution of random variable  $\mathbf{G}_{\mathbf{h}_i}$ . Manual HPO for overall high performance is difficult and tedious [9] as we as humans need to configure  $\mathbf{h}_i$  and then re-configure it based on the feedback of  $\mathbf{G}_{\mathbf{h}_i}$ , and we keep doing this loop until we are satisfied with the performance or running out of time budget. For instance, one would like to have a control algorithm that has a certain tracking accuracy (performance), and he/she has a day (time budget) to tune the HPs to fulfill the specification. HPO tools (e.g. Optuna [15] and SMAC [16]) have been developed to automate this. Utilizing HPO tools in this context offers several advantages, including better reproducibility, less human time, better performance, and the evaluation of a broader range of configurations [17]–[19]. Those advantages can ensure fairer comparisons between algorithms, i.e., benchmarking algorithms [9]. Ideally, after

---

solving the HPO, optimized HPs may exhibit the properties of performance being transferable across (1) environments and (2) multiple runs/seeds [18]. The former property is desirable to have as we do not need to do time-consuming HPO for other environments where we deploy our algorithm. Although the latter attribute is basic, it remains a challenge in HPO for RL.

Overfitting in HPO is a well-known problem, which is also called overconfidence or overtuning [20]. In this case, the performance cannot be generalized. This is typically the case when  $\hat{\mathbf{G}}_{\mathbf{h}_i}$  is not representative of  $\mathbb{E}[\mathbf{G}_{\mathbf{h}_i}]$ , and then the optimizer would pick this particular  $\mathbf{h}_i$  because the biased overestimated approximation  $\hat{\mathbf{G}}_{\mathbf{h}_i}$ . Overfitting in HPO for RL is even more challenging [17] since random seeds can dramatically influence how hyperparameters affect the behaviour of the RL algorithms, resulting in varying outcomes [19], [21]. As a consequence, HPO may overfit the HPs to particular scenarios where the solver obtains high-performance runs given random seeds by chance, but the performance cannot be generalized to unseen seeds. In general, reporting evaluation metrics in an objective from multiple runs/seeds as an estimator of true evaluation metrics would achieve more accurate estimations [13], [22]; thus, this avoids overfitting to some random seeds, but the evaluation would be expensive. If we evaluate in a few-run regime, this leads to overestimation/underestimation of true evaluation metrics [23]. The high variance of performance of RL algorithms across different seeds makes HPO for RL challenging [17] as estimations from a handful of samples tend to overestimate/underestimate the true performance. Overfitting in HPO has a synonym called maximization bias [24]. Particularly, using the maximum value from the estimators as an approximation for the maximum expected value would face the problem of maximization bias. Optimization bias is a problem that is particularly pronounced in high-variance problems such as RL. Nevertheless, maximization bias in HPO for RL has, to the best of our knowledge, attracted little attention, but avoiding this is essential to enable robust and reproducible comparisons between algorithms. Other than maximization bias, from the objective term in HPO problem, optimizing over expectations may not capture the uncertainty or risk well. Therefore, in this work, we enable improved comparisons by accounting for maximization bias and leverage different measures other than expectation in order to capture the desired characteristics of the distribution.

## 1.1 Problem Formulation

In this thesis, we address the impact of random seeds on decision-making algorithms and develop HPO strategies that give us optimized HPs resulting in consistent, reproducible, and generalized performance under a limited computational budget.

Consider the evaluation metrics of a decision-making algorithm  $\mathbf{G}$  on an uncertain environment  $\epsilon$ , and the algorithm hyperparameter  $\mathbf{h} \in \mathcal{H}$ , where  $\mathcal{H}$  is the considered search space. Then, the hyperparameter optimization problem can be generally written as

$$\mathbf{h}^* = \arg \max_{\mathbf{h} \in \mathcal{H}} \rho[\mathbf{G}|\mathbf{h}, \epsilon],$$

where  $\rho$  is a risk functional (risk measure) mapping from the space of the random variable  $\mathbf{G}$  to  $\mathbb{R}$ . Note that expectation is one of the risk functionals (i.e.,  $\rho = \mathbb{E}$ ) that captures the mean characteristic of the distribution. The goal of this work is to extract  $\mathbf{h}^*$  that yields consistent performance (robust against random seeds) by observing data sequences  $\{(\mathbf{h}_1, \hat{\rho}[\mathbf{G}|\mathbf{h}_1, \epsilon]), \dots, (\mathbf{h}_M, \hat{\rho}[\mathbf{G}|\mathbf{h}_M, \epsilon])\}$ , where  $\hat{\rho}[\mathbf{G}|\mathbf{h}_i, \epsilon]$  (or shorthand  $\hat{\rho}[\mathbf{G}_{\mathbf{h}_i}]$ ) is the estimation of  $\rho[\mathbf{G}|\mathbf{h}_i, \epsilon]$  (or shorthand  $\rho[\mathbf{G}_{\mathbf{h}_i}]$ ) for  $i = \{1, 2, \dots, M\}$  and  $M$  is the maximum number of trials. In this thesis, we consider the budget to be the maximum number of iterations (trials) or the maximum number of agent runs. A trial (iteration) denotes one gathering of a pair of data  $(\mathbf{h}_i, \hat{\rho}[\mathbf{G}_{\mathbf{h}_i}])$ . Defining a maximum number of agent runs as the budget is also popular as it is usually the expensive bottleneck for HPO.

## 1.2 Contributions

Our contributions are fourfold:

1. We prove that current methods always suffer from maximization bias when estimators have uncertainty, and we derive a condition to ensure zero maximization bias.
2. Combined with another risk functional, we propose a practical algorithm that mitigates maximization bias and considers the risk while possibly requiring fewer computational resources during optimization.

3. We demonstrate that the proposed algorithm has the capability of reducing maximization bias and considering the risk on synthetic examples.
4. We show that our algorithm can foster fairer comparisons, consistency, and reproducibility for various learning-based control and RL algorithms while requiring less computing time during optimization.

## 1.3 Structure of the Thesis

The thesis is structured as follows:

1. **Preliminaries:** We give a formal definition of hyperparameter optimization in the context of decision-making algorithms, with illustrative block diagrams and a synthetic example illustrating how the optimization operates. Furthermore, we introduce the estimators we use to approximate the considered risk measures. Next, we introduce the selected decision-making algorithms for benchmarking and their core formulas. After that, we introduce stratified bootstrap confidence intervals for estimating uncertainty in sample estimates. Lastly, the selected metrics for the control algorithm’s performance are presented.
2. **Related Work:** We give an overview of developments in the field of hyperparameter optimization and the current research challenges.
3. **Methods:** We study the issue of maximization bias and derive a condition for zero maximization bias. Furthermore, combining the risk functional with the insight from the theoretical analysis allows us to propose a novel objective for HPO.
4. **Results:** We demonstrate that our algorithm indeed reduces maximization bias and takes the risk into account on simulation examples and a range of decision-making algorithms.
5. **Conclusion and Outlook:** We briefly summarize the results from this thesis and give an outlook on future improvements.



## 2 Preliminaries

This chapter introduces the background necessary for the remainder of this thesis. In particular, we first introduce hyperparameter optimization in the context of decision-making algorithms. Secondly, we touch upon one of the risk measures for risk quantification. Thirdly, we review a few common estimators for risk functionals. Next, we give an overview of the selected decision-making algorithms that we aim to benchmark and their corresponding hyperparameters. After that, we introduce a statistical tool: bootstrap confidence intervals to measure the uncertainty in the aggregate performance of control algorithms. Lastly, we present two selected popular metrics and a more robust metric for control algorithms.

### 2.1 Hyperparameter Optimization

Albeit briefly stated in Chapter 1, we give a formal definition of hyperparameter optimization in the context of decision-making algorithms.

**Definition 1.** (*Hyperparameter Optimization*). *Let  $\mathbf{G}$  be the evaluation metrics of a decision-making algorithm on an uncertain environment  $\epsilon$ , and the algorithm's hyperparameter  $\mathbf{h} \in \mathcal{H}$ , where  $\mathcal{H}$  is the considered hyperparameter search space. Consider a risk functional  $\rho$  mapping from a space of the random variable  $\mathbf{G}$  to  $\mathbb{R}$ . Then the hyperparameter optimization problem is*

$$\mathbf{h}^* = \arg \max_{\mathbf{h} \in \mathcal{H}} \rho[\mathbf{G}|\mathbf{h}, \epsilon]. \quad (\text{HPO})$$

Some main characteristics of the problem include (1)  $\mathbf{G}$  is typically expensive to evaluate, (2) the gradients of  $\mathbf{G}$  may not be accessible; thus, the problem is termed a black-box optimization problem, (3) if  $\mathbf{G}$  contains more than one element, then it is under the scope of multi-objective optimization. To automate the optimization

process, there are different methods for solving HPO such as grid search, random search, and Bayesian optimization (BO) type methods. There exist hyperparameter optimization tools (e.g. Optuna [15] and SMAC [16]) where different HPO algorithms are implemented. Core components of HPO tools consist of a sampler and an objective (see Figure 2.1). For example, the HPO toolbox Optuna implements samplers such as random sampler [10] and tree-structured parzen estimator (TPE), a BO type of sampler [25] (to see the complete list <sup>1</sup>). BO-type methods can actively sample the next HPs by leveraging a surrogate model. If one would like to perform HPO for decision-making algorithms, then the objective can be simply replaced by some evaluation metrics to assess an agent’s performance after being trained in a certain environment (see Figure 2.2). A popular evaluation metric is the validation reward of an agent (i.e., the evaluation of a trained policy using random seeds different from the ones used for training [26]). Another choice could be the learning efficiency of an agent [21].

To provide the intuition, we define a non-convex function  $G$  with a one dimensional  $h$  with  $\mathcal{H} = [-7.5, 7.5]$  (see Figure 2.3) for Optuna to solve. Please note that  $G$  given  $h$  is always constant so the HPO problem simplifies to  $h^* = \arg \max_{h \in \mathcal{H}} G(h)$ . We solve the HPO problem for 100 times, and the TPE sampler roughly takes 350 trials to stably return the global optimum, while the random sampler returns sub-optimum sometimes (see Figure 2.4). With the number of trials increasing, we expect the optimal solution to be returned with a higher probability. Due to the modeling effort of the TPE sampler, the number of trials required is less than the random sampler (see Figure 2.5).

## 2.2 Risk Measures

A common risk measure used in HPO is the expectation  $\mathbb{E}[G|\mathbf{h}, \epsilon]$  [21]. However, the characteristic of the tail distribution of the random variable  $\mathbf{G}$  cannot be captured by its expectation. The risk measure we would like to use in this work is termed conditional value at risk (CVaR). Let cost  $Z$  be a random variable, then  $\text{CVaR}_\alpha(Z)$  stands for the conditional value at risk (VaR) of  $Z$  at the  $\alpha$  level, where  $\text{VaR}_\alpha(Z)$

---

<sup>1</sup><https://optuna.readthedocs.io/en/stable/reference/samplers/index.html>



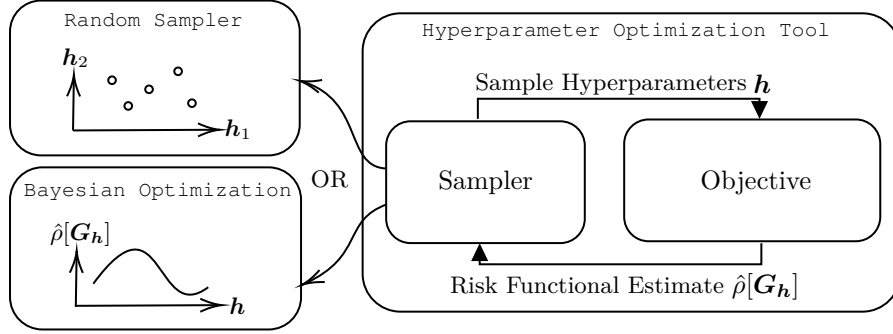


Figure 2.1: Hyperparameter optimization tool. Tools consist of two main components: one is a sampler, and the other is a user-defined objective. Here we list two possible types of samplers: the random sampler and the Bayesian optimization (BO) sampler. A random sampler, as its name suggests, samples randomly in HPs search space in every iteration. BO models the relation between hyperparameters and the risk functional estimate over evaluation metric; hence, it is usually more data efficient.

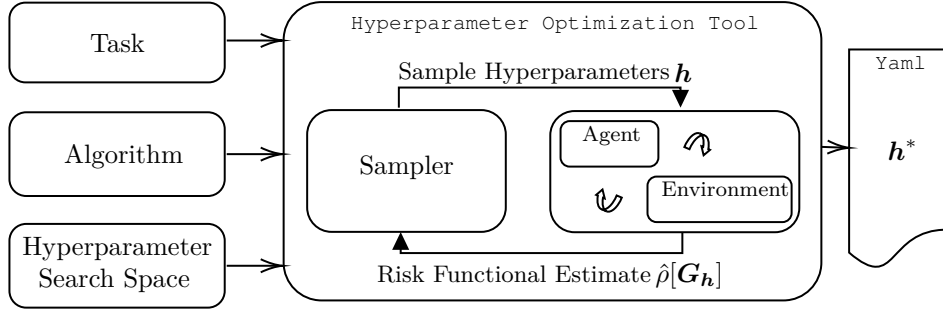


Figure 2.2: Hyperparameter optimization tool applied on learning-based control and reinforcement learning. As we consider learning-based control and RL algorithms, there are interactions between an agent and an environment involved within the objective block. After running out of the number of iterations (trials), the tool will report  $\mathbf{h}^*$  considered the best so far.

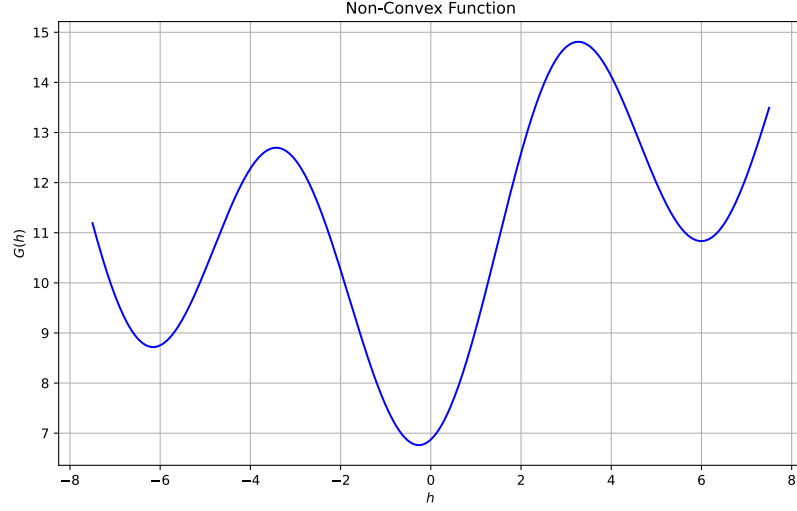


Figure 2.3: A non-convex function  $G$  as a function of  $h$ .  $G(h) = 4\sin\left(\frac{h-3}{3}\right) + \frac{5\cos(h-3)+20}{1.7}$ .

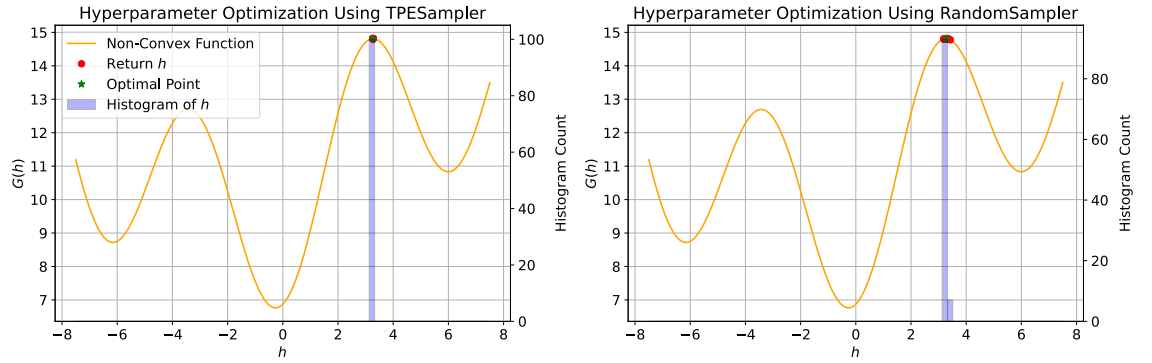


Figure 2.4: HPO results for solving optimization over the non-convex function. The left-hand side plot uses TPE sampler, while the right-hand side plot uses random sampler. The red dots denote the end return  $h^*$  and  $G(h^*)$  for 350 trials for a hundred times. The blue histogram shows how the end returns are distributed.

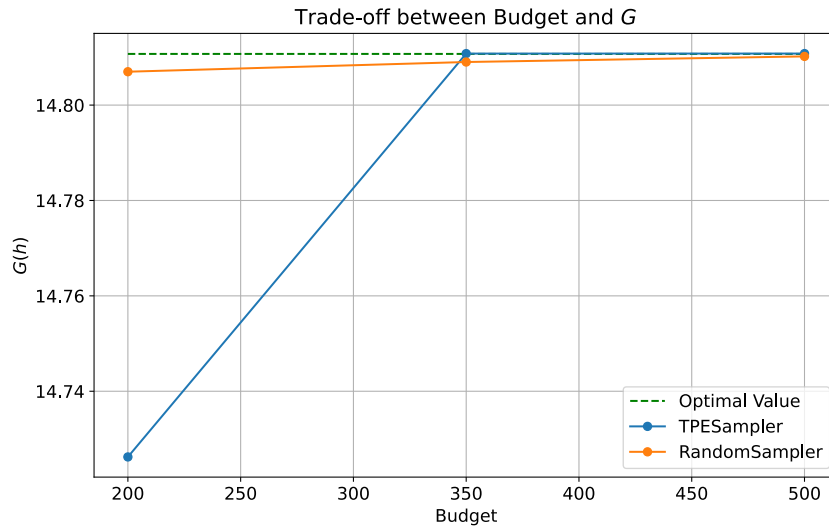


Figure 2.5: Trade-off between budget and  $G$ . The plot shows the average  $G(h)$  returns for a hundred times HPO runs at 200, 350 and 500 trials. It showcases TPE sampler acquires more sub-optimal solutions than the random sampler at the budget of 200 trials. At the budget of 350 trials, the number of trials is large enough and the effect of the modeling in the TPE sampler allow it to outperform the random sampler. Without modeling, the random sampler still cannot always return the optimal point at 500 trials.

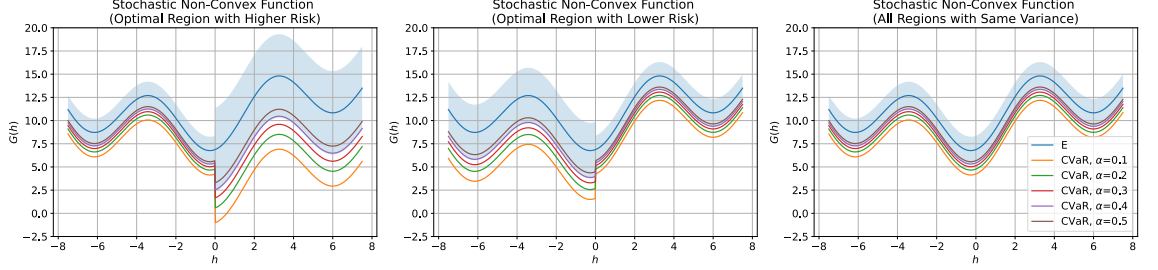


Figure 2.6: CVaR for different  $\alpha \in \{0.1, \dots, 0.5\}$ . The blue line is the expectation of the non-convex function (legend E). The blue-shaded areas are the standard deviation of the noise. The plots show that CVaR indeed quantifies the risk that the region having a larger variance has the uncertainty of getting a lower reward.

refers to the left side  $(1 - \alpha)$ -quantile of  $Z$  [27]:

$$\text{VaR}_\alpha[Z] := \inf\{z \in \mathbb{R} : F_Z(z) \geq 1 - \alpha\}, \quad \alpha \in (0, 1), \quad (2.1)$$

where  $F_Z(z)$  is the cumulative distribution function (CDF) of  $Z$ . This is commonly known as the value at risk of  $Z$  at the  $\alpha$  level. Then,  $\text{CVaR}_\alpha(Z)$  represents the mean value of  $Z$  for values that exceed  $\text{VaR}_\alpha(Z)$  [27]:

$$\text{CVaR}_\alpha[Z] = \frac{1}{\alpha} \int_{1-\alpha}^1 \text{VaR}_{1-s}(Z) ds. \quad (2.2)$$

Synonyms for CVaR are expected shortfall or expected tail loss. Since rewards are commonly used in RL algorithms, we use negated rewards as costs, and the above definition can be applied as usual. Figure 2.7 visualizes risk functionals for expectations, VaR, and CVaR.

To gain intuitions, we use the same non-convex function shown in Figure 2.3 but add Gaussian zero mean noise with different standard deviations. We compute the corresponding value from the closed-form formula of  $\text{CVaR}_\alpha$  for the given normal distribution. By lowering  $\alpha$ , CVaR calculates expectations of more extreme events (see Figure 2.6); thus, it quantifies the risk of tail distributions. Note that if we do not know the distribution, then the analytical solution of CVaR is no longer available. Thus, CVaR needs to be computed from historical data.

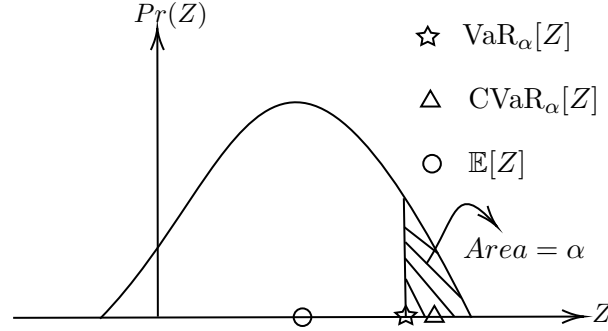


Figure 2.7: Visualization for risk functionals. The plot depicts different characteristics of the distributions of the random cost  $Z$ .  $\mathbb{E}[Z]$  is the expectation of  $Z$ .  $\text{VaR}_\alpha[Z]$  is the left side  $(1 - \alpha)$ -quantile of  $Z$ , while  $\text{CVaR}_\alpha[Z]$  is the conditional value at risk of  $Z$  at level  $\alpha$ .

## 2.3 Estimators

Since  $\rho[\mathbf{G}|\mathbf{h}, \epsilon]$  defined in HPO is usually intractable and expensive to compute (or even no closed-form solution if distributions are unknown), the value is often approximated by some estimators from historical data. Given that, this section reviews estimators for estimating the expectation and CVaR over a random variable.

### 2.3.1 Maximum Likelihood Estimator for a Gaussian

Suppose that we have a dataset of  $N$  observations  $S = \{s_1, \dots, s_N\}$  being independent and identically distributed (i.i.d.) from a Gaussian distribution with  $\mu$  and  $\sigma^2$ . Since each sample is i.i.d., we can express the likelihood function as:

$$p(S|\mu, \sigma^2) = \prod_{i=1}^N \mathcal{N}(s_i|\mu, \sigma^2). \quad (2.3)$$

One could derive a maximum likelihood estimator (MLE) by maximizing the log-likelihood function and solving for  $\mu$ :

$$\max_{\mu} \ln p(S|\mu, \sigma^2) \quad (2.4)$$

$$\implies \mu_{\text{ML}} = \frac{1}{N} \sum_{i=1}^N X_i. \quad (2.5)$$

One can notice that the MLE is equivalent to the sample mean estimator under the assumption that the samples are independent and identically Gaussian distributed. We then introduce estimators by relaxing the assumption of Gaussian distribution.

### 2.3.2 Mean Estimator and CVaR Estimator for Arbitrary Distribution

In Section 2.3.1, we know the maximum likelihood estimator for a Gaussian is just a sample mean estimator. Without loss of generality, the sample mean estimator is unbiased regardless of the underlying distribution of samples. As we have no idea of the true distribution of the performance of decision-making algorithms, we simply use a sample mean estimator to estimate the true mean.

Regarding the CVaR estimator, we can estimate CVaR from samples. That is, the value is computed by taking the average of the samples exceeding a certain quantile, which controls the extent of risk one can undertake.

## 2.4 Decision Making Algorithms

The motivation of this work is to benchmark decision-making algorithms. Hence, this section is for introducing the considered decision-making algorithms and their HPs.

### 2.4.1 Reinforcement Learning

We consider the standard RL problem on a Markov Decision Process (MDP), which is defined by the tuple  $(\mathcal{S}, \mathcal{A}, T, \gamma, r)$ .  $\mathcal{S}$  and  $\mathcal{A}$  are the continuous state and action spaces respectively,  $T$  is the transition dynamics,  $r$  is the reward function, and  $\gamma$  is the discount factor. We note state and action as  $\mathbf{s} \in \mathcal{S}$  and  $\mathbf{a} \in \mathcal{A}$ , respectively.

Besides, we denote  $\mathbf{s}_t$  and  $\mathbf{a}_t$  as the state and action at time step  $t$ .

**Deep Deterministic Policy Gradient:** deep deterministic policy gradient (DDPG) [28] is an algorithm that trains a Q-function and a policy concurrently. It leverages off-policy data and mean-squared Bellman error (MSBE) function to update the Q-function (Equation 2.6), which in turn informs the improvement of the policy (Equation 2.7).

$$Q^\pi(\mathbf{s}, \mathbf{a}) = \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \sim \rho_\pi} \left[ \sum_{t=0}^{N-1} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \mid \mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a} \right]. \quad (2.6)$$

with  $\rho_\pi$  the marginals of trajectory distribution induced by  $\pi$  and  $N$  as the control horizon.

If we parameterize the Q-function by  $\phi$  and the deterministic policy  $\pi$  by  $\theta$ , i.e.,  $\pi_\theta$ , then the policy learning of DDPG is solving the problem

$$\max_{\theta} \mathbb{E}_{\mathbf{s} \sim \mathcal{D}} [Q_\phi(\mathbf{s}, \pi_\theta(\mathbf{s}))]. \quad (2.7)$$

with  $\mathcal{D}$  a set containing experience.

We list HPs and their search spaces in Table 2.1. We define the search spaces by either referring to Stable Baselines3 (SB3)<sup>2</sup> or expanding spaces centered around the default HP tuned by the authors of prior work [5].

**Proximal policy optimization:** proximal policy optimization (PPO) [8] is an on-policy RL algorithm that optimizes a surrogate objective function while restricting the distance between current policies  $\pi_{\tilde{\theta}}$  and successive policies  $\pi_\theta$  by Kullback-Leibler (KL) divergence:

$$\max_{\theta} \mathbb{E}_{\mathbf{s}_t, \mathbf{a}_t \sim \pi_{\tilde{\theta}}} \left[ \frac{\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)}{\pi_{\tilde{\theta}}(\mathbf{a}_t | \mathbf{s}_t)} \hat{A}_{\pi_{\tilde{\theta}}}(\mathbf{s}_t, \mathbf{a}_t) - \beta \cdot \text{KL} [\pi_{\tilde{\theta}}(\cdot | \mathbf{s}_t), \pi_\theta(\cdot | \mathbf{s}_t)] \right], \quad (\text{KL-penalized problem})$$

---

<sup>2</sup><https://github.com/DLR-RM/stable-baselines3>

Hyperparameter	Search Space
Hidden Dimensions	{32, 64, 128, 256, 512}
Activations	{Tanh, Relu, Leaky Relu}
Gamma	{0.9, 0.95, 0.98, 0.99, 0.995, 0.999, 0.9999}
Train Interval	{10, 100, 1000}
Train Batch Size	{32, 64, 128, 256, 512}
Maximum Steps	{30000, 54000, 72000}
Warm Up Steps	{500, 1000, 2000, 4000}
Tau	Interval(0.005, 1)
Actor Learning Rate	log(Interval(1e-5, 1))
Critic Learning Rate	log(Interval(1e-5, 1))

Table 2.1: Hyperparameters for DDPG. Hidden Dimensions determine the number of neurons within neuron networks (NNs), while Activations dictate the choice of activation functions used in NNs. Gamma serves as the discount factor in RL algorithms, and Train Interval defines the frequency of training during optimization. Train Batch Size specifies the number of data points in each training update, while Maximum Steps cap the maximum number of training iterations, which can be thought of as necessary steps to converge. Warm Up Steps are the steps before initiating training. Tau represents an interpolation factor for updating a target network. Both Actor Learning Rate and Critic Learning Rate regulate learning rates for updating actor and critic model weights.

where  $\beta$  is adapted based on the hyperparameter  $d_{\text{targ}}$  and  $\hat{A}_{\pi_{\hat{\theta}}}$  is a generalized advantage estimator. Another way to penalize changes to the policy is:

$$\max_{\theta} \mathbb{E}_{s_t, a_t \sim \pi_{\theta}} \left[ \min \left( \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\hat{\theta}}(a_t | s_t)} \hat{A}_{\pi_{\hat{\theta}}}(s_t, a_t), \text{clip} \left( \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\hat{\theta}}(a_t | s_t)}, 1 - \epsilon, 1 + \epsilon \right) \hat{A}_{\pi_{\hat{\theta}}}(s_t, a_t) \right) \right],$$

(Clipped surrogate problem)

where epsilon  $\epsilon$  is a hyperparameter that controls the distance between new and old policy in the Clip function. PPO’s performance is highly relevant for sufficient hyperparameter tuning and the configuration of code-level optimizations [29]. We list the HPs of PPO in Table 2.2. The search spaces again are defined by referencing SB3 <sup>3</sup> or expanding spaces centered around the default HP tuned by the authors of

<sup>3</sup><https://github.com/DLR-RM/stable-baselines3>



prior work [5].

Hyperparameter	Search Space
Hidden Dimensions	{8, 16, 32, 64, 128, 256}
Activations	{Tanh, Relu, Leaky Relu}
Gamma	{0.9, 0.95, 0.98, 0.99, 0.995, 0.999, 0.9999}
Lambda	{0.8, 0.9, 0.92, 0.95, 0.98, 0.99, 1.0}
Clip Parameter	{0.1, 0.2, 0.3, 0.4}
Mini Batch	{32, 64, 128}
Rollout Steps	{50, 100, 150, 200}
Maximum Steps	{30000, 54000, 72000}
Target KL	$\log(\text{Interval}(1\text{e-}8, 0.8))$
Entropy Coefficient	$\log(\text{Interval}(1\text{e-}8, 0.1))$
Actor Learning Rate	$\log(\text{Interval}(1\text{e-}5, 1))$
Critic Learning Rate	$\log(\text{Interval}(1\text{e-}5, 1))$

Table 2.2: Hyperparameters for PPO. The first three HPs have the same definition as the counterparts in Table 2.1. Lambda represents a trade-off parameter in a generalized advantage estimator. Clip Parameter controls the trust region for policy updates, and Mini Batch specifies the number of data points in each training update. Maximum Steps cap the maximum number of training iterations, which can be thought of as necessary steps to converge. Target KL is the parameter associated with KL divergence. Entropy Coefficient is used to control exploration behavior. Both Actor Learning Rate and Critic Learning Rate are defined the same as in Table 2.1.

**Soft actor critic:** soft actor critic (SAC) is a state-of-the-art and off-policy algorithm that optimizes the expected sum of rewards and entropy of the policy  $\mathcal{H}(\pi(\cdot|\mathbf{s}))$  over stochastic policy [30]. We define the Q value function to include the entropy term:

$$Q^\pi(\mathbf{s}, \mathbf{a}) = \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \sim \rho_\pi} \left[ \sum_{t=0}^{T-1} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) + \alpha \sum_{t=1}^{T-1} \gamma^t \mathcal{H}(\pi(\cdot|\mathbf{s}_t)) \mid \mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a} \right] \quad (2.8)$$

with  $\alpha$  the entropy regularization coefficient.

SAC concurrently updates the parameterized Q function by minimizing a modi-

fied MSBE and the parameterized policy by minimizing the KL divergence between the policy distribution and the normalized exponential of the updated Q function. The HPs and their search spaces are defined the same as the previously mentioned DDPG and are summarized in Table 2.3.

Hyperparameter	Search Space
Hidden Dimensions	{32, 64, 128, 256, 512}
Activations	{Tanh, Relu, Leaky Relu}
Gamma	{0.9, 0.95, 0.98, 0.99, 0.995, 0.999, 0.9999}
Train Interval	{10, 100, 1000}
Train Batch Size	{32, 64, 128, 256, 512}
Maximum Steps	{30000, 54000, 72000}
Warm Up Steps	{500, 1000, 2000, 4000}
Tau	Interval(0.005, 1)
Actor Learning Rate	$\log(\text{Interval}(1\text{e-}5, 1))$
Critic Learning Rate	$\log(\text{Interval}(1\text{e-}5, 1))$

Table 2.3: Hyperparameters for SAC. The hyperparameters and the search spaces are defined the same as in Table 2.1.

### 2.4.2 Learning-Based Control

Learning-based control methods enhance a robot’s performance by utilizing historical data to refine the understanding of the actual system dynamics. These methods usually offer assurances regarding the system’s stability and/or its ability to meet constraints. One of the methods is model predictive control using Gaussian process (GP-MPC) [6], [7].

**Model predictive control using Gaussian process:** model predictive control using Gaussian process (GP-MPC) is a control strategy that uses the Gaussian process to improve the model uncertainty and optimizes control actions over a finite time horizon  $N$ . Note that in the following, we follow the convention in the control community that state and action denote as  $x \in \mathcal{S}$  and  $u \in \mathcal{U}$ , respectively. We denote that  $x_k$  is the state of the system at time step  $k$ . The state  $x_k$  represents the system’s state at time step  $k$ . The state  $x_{k+i|k}$  is the predicted system state at time

step  $k + i$ , calculated at time step  $k$  by initiating with  $x_{k|k} = x_k$  and executing the control input sequence  $u_{k|k}, \dots, u_{k+i-1|k}$  in the system model  $x_{k+1|k} = f(x_{k|k}, u_{k|k})$ . Similarly,  $u_{k+i|k}$  represents the control input computed at time  $k$  for use at time step  $k + i$ .

We consider  $f$  as system dynamics, and  $\bar{f}$  as the prior dynamics. GPs are used to model system residual plus some noise  $\delta f(x_k, x_k) + w_k$ :

$$f(x_k, u_k, w_k) = \bar{f}(x_k, u_k) + \delta f(x_k, u_k) + w_k, \quad (2.9)$$

As a result, GP-MPC can be formulated below as a stochastic optimization problem:

$$\min_{u_{k|k}, \dots, u_{k+N-1|k}} \mathbb{E}_{w_k} \left[ g_N(x_{k+N|k}) + \sum_{i=0}^{N-1} g_k(x_{k+i|k}, u_{k+i|k}, w_k) \right] \quad (2.10)$$

$$\text{subject to } x_{k+i+1|k} = \bar{f}(x_{k+i|k}, u_{k+i|k}) + \delta f(x_{k+i|k}, u_{k+i|k}) + w_i, \quad (2.11)$$

$$\forall i = \{0, 1, \dots, N-1\},$$

$$Pr(x_{k+i|k} \in \mathcal{S}) \geq p_x, \quad \forall i = \{0, 1, \dots, N-1\}, \quad (2.12)$$

$$u_{k+i|k} \in \mathcal{U}, \quad \forall i = \{0, 1, \dots, N-1\}, \quad (2.13)$$

$$x_{k|k} = x(0) \in \mathcal{S}, \quad (2.14)$$

with  $g_N$  a terminal cost function,  $g_k$  a cost function, and  $Pr(x_{k+i|k} \in \mathcal{S}) \geq p_x$  a probabilistic constraint that guarantees  $x_{k+i|k} \in \mathcal{S}$  with probability  $p_x$ . To turn the stochastic optimization problem into a deterministic one, one can formulate the probabilistic constraint using the covariance of the GP predictions to tighten the constraints and approximate the expectation of cost by evaluating the cost at the mean state estimate. The optimization yields the optimal input sequence  $\{u_{k|k}^*, \dots, u_{k+N-1|k}^*\}$ , in which the first control input is applied to the system  $u_k = u_{k|k}^*$ . At the next time step, we solve again the finite-horizon constrained optimal control problem at state  $x_{k+1}$  with horizon  $N$ . We do this recursively until the task is finished. We list HPs and the search spaces in Table 2.4. The search spaces are defined by expanding the space centered around the default HP tuned in prior work [5].

Hyperparameter	Search Space
Horizons	{10, 15, 20, 25, 30, 35}
Number of Inducing Points	{30, 40, 50}
Number of Samples	{70, 75, 80, 85}
Number of Epochs	{4, 5, 6, 7, 8}
Kernel	{Matern, RBF}
Optimization Iterations	{2800, 3000, 3200}
Learning Rate	$\log(\text{Interval}(5e-4, 0.5))$

Table 2.4: Hyperparameters for GP-MPC. Horizons are the number of predicted steps in model predicted control. Number of Inducing Points means how many points are used for approximating GPs, while Number of samples represents how many data points are used to compute GPs. Number of Epochs represents how many complete passes through the training dataset. The kernel options here are Matern and Radial Basis Function (RBF), two common choices in GPs. Optimization Iterations determine the number of optimization iterations performed during model training. Learning Rate control how fast to update the parameters in GPs.

## 2.5 Stratified Bootstrap Confidence Intervals

The bootstrap method is a resampling technique used in statistics to estimate the sampling distribution by repeatedly sampling from the observed data with replacement [31]. This method provides sample estimates with measures of accuracy such as confidence intervals (CIs), which is useful for estimating the sample distribution of a control algorithm that performs a single task with  $N$  runs. In deep RL benchmarks, reporting aggregating results across various tasks using point estimates is common [23]. The work [23] proposes to use stratified bootstrap confidence intervals to account for the uncertainty in point estimates. To elaborate, we first organize our data points into buckets (representing tasks), and subsequently, we perform resampling with replacement from each of these buckets, considering their respective sizes. This avoids computing aggregate performance on the subset of tasks. Each time we sample with replacement of the runs, and we compute the metric of interest for those sampled runs. Finally, the CIs are calculated based on the percentiles of the metric for the sampled runs. The percentiles’ method is found to work well in practice [23]. Hence, in this thesis, we report the uncertainty in results using bootstrap confidence

intervals (CIs) following the percentiles' method.

## 2.6 Metrics

The sample mean and sample median are two common metrics to evaluate the performance of a control algorithm. The sample mean is sensitive to outliers as the extreme scores change the value dramatically, while the sample median is not robust in the sense that the value does not vary for the score of 0 on half of the samples. Interquartile mean (IQM) strikes a balance between these two. That is, it computes the mean from the middle 50% scores. It is typically less sensitive to outliers than the mean, provides a better overall performance indicator than the median, and results in narrower confidence intervals (CIs) [23]. Note that different metrics capture different aspects of an algorithm's performance, so it would not be sufficient to consider a single metric for evaluations. As an attempt toward benchmarking, we report evaluations of selected decision-making algorithms based on mean, median, and IQM even if there exist several others.



### 3 Related Work

In the realm of hyperparameter optimization (HPO) methods, a variety of techniques have been developed to enhance the efficiency and effectiveness of the optimization process. These methods start from one of the most basic approaches, random search, which has been proven effective [10]. Bayesian optimization (BO) is introduced to improve the data efficiency. In essence, BO leverages a surrogate model to iteratively search for the optimal solution of a costly and black-box objective function by optimizing acquisition functions given the observations of data. Common choices of acquisition functions are expected improvement (EI) [32] and probability of improvement (PI) [33]. Both can be used in Bayesian optimization (BO) with Gaussian processes (GPs) [34]. However, those acquisition functions have cubic computational complexity in the number of data points if we use GPs to build a surrogate model. Therefore, a range of methods are developed. For instance, a neural network-based approach tailored for scalability is proposed [35]. Furthermore, non-GP-based techniques such as the tree-structured parzen estimator (TPE) [25] have been introduced for less expensive modeling. Some follow-up works are extended to cope with multi-objective optimization, i.e., multi-objective tree-structured parzen estimator (MOTPE) [36], [37]. BO-based methods are less sensitive to noisy evaluations [21] as their acquisition functions (EI and PI) are modeled with the assumption of noisy observations [32], [33]. Nonetheless, excessively noisy evaluations can still affect modeling landscapes, leading to performance degradation [38]. For example, the work [39] shows that noise levels have an impact on acquisition functions for TPE. There are approaches particularly designing the acquisition functions to handle uncertainty [38], [40], while those acquisition functions are typically non-trivial and expensive to compute. In addition, BO-based methods usually assume noisy observations where the noise is distributed uniformly with the same variance [32], [33]. Nevertheless, the distribution given different HPs can have an extremely distinguished distribution (especially for RL algorithms), violating the assumption. In

other research directions, robust evaluation metrics such as interquartile mean are proposed [23] as evaluations involving mean or median can be very unreliable [21]. However, interquartile mean is calculated by discarding extreme cases, which may not be an appropriate practice in HPO because we may extract  $\mathbf{h}^*$  that have good interquartile mean but bad worst-case performance. To the best of our knowledge, there have not been risk functionals other than expectations integrated into HPO to capture the worst-case characteristic of the distribution. As a result, our work focuses on analyzing the effect of risk functionals to report evaluations in order to find  $\mathbf{h}^*$  that have good worst-case performance if that is at all possible.

While reinforcement learning (RL) has demonstrated remarkable capabilities in areas such as game playing, robotics, and autonomous systems, it is not without its challenges. The inherent brittleness of RL algorithms, as highlighted in prior works [13], [23], motivates the research question of how to extract the possible best HPs. HPO for RL, or automated RL (AutoRL) [21], as the emerging field seeks to automate and enhance the process of optimizing RL algorithms over their HPs. Although the standard HPO tools can be used to solve the AutoRL problem, works have been proposed to dynamically optimize over HPs of RL algorithm [18], [41]. They claim that HPs should be dynamically adjusted instead of statically fixed during the training process of RL algorithms and have proven to enhance performance. The issue of HPO for RL is that noisy RL performance evaluations, attributed to the influence of random seeds on natural instabilities in training [21], pose a challenge to the effectiveness of AutoRL [17]. This is related to overconfidence and overfitting [20], or maximization bias [24]. They describe the situation where one may receive sub-optimal solutions from HPO if the observations are not representative. The authors in [20] provide the statistical argument for bias for exponential distributions of the cost and give a relevant reference [42]. Our work revisits the issue of maximization bias and uses synthetic examples to illustrate the effect. The most common objective in HPO for RL is expectation over cumulative rewards or efficiency, which are often approximated by results from multiple training runs or seeds [21]. However, expectations cannot capture the risk of the tail distribution. To the best of our knowledge, there have not been works that combine different risk measures [27] in HPO. In this thesis, we study the effect of replacing expectations with other risk measures over evaluation metrics as the novel HPO objective. Together with risk measures and theoretical insights on the maximization bias, we



---

propose an algorithm that can reduce maximization bias and deal with potential uncertainty.



## 4 Method

Our proposed method tackles two facets: the challenge of (1) maximization bias and (2) risk consideration. First, we start by analyzing the maximization bias and its condition. Thereafter, based on the theoretical insights, we propose an algorithm termed adaptive multiple runs algorithm (AMRA) that aims at tackling the issue of maximization bias. Secondly, we propose to optimize over conditional value at risk (CVaR) to consider risk and uncertainty. These two aspects could be combined as our final proposed algorithm, which is presented in Section 4.3.

### 4.1 Reducing Maximization Bias

For the sake of simplicity, we study maximization bias under the assumption that the risk functional is the expectation

$$\mathbf{h}^* = \arg \max_{\mathbf{h} \in \mathcal{H}} \mathbb{E} [\mathbf{G} | \mathbf{h}, \epsilon].$$

As we are interested in optimizing the HPs of decision-making algorithms, and that the distribution induced by a random variable  $\mathbf{G}_{\mathbf{h}}$  is unknown,  $\mathbb{E}[\mathbf{G} | \mathbf{h}, \epsilon]$  ( $\mathbb{E}[\mathbf{G}_{\mathbf{h}}]$ ) is often approximate by mean estimators  $\hat{\mathbf{G}}_{\mathbf{h}}$ . After observing a sequence of data  $\{(\mathbf{h}_1, \hat{\mathbf{G}}_{\mathbf{h}_1}), \dots, (\mathbf{h}_M, \hat{\mathbf{G}}_{\mathbf{h}_M})\}$ , the optimization is done by:

$$\mathbf{h}^* = \arg \max_{\mathbf{h}} \{\hat{\mathbf{G}}_{\mathbf{h}_1}, \dots, \hat{\mathbf{G}}_{\mathbf{h}_M}\}.$$

It is obvious to see that the value of  $\max\{\hat{\mathbf{G}}_{\mathbf{h}_1}, \dots, \hat{\mathbf{G}}_{\mathbf{h}_M}\}$  has a great impact on final selection of  $\mathbf{h}^*$ . Therefore, we study the question if  $\max\{\hat{\mathbf{G}}_{\mathbf{h}_1}, \dots, \hat{\mathbf{G}}_{\mathbf{h}_M}\}$  is equal to  $\max\{\mathbb{E}[\mathbf{G}_{\mathbf{h}_1}], \dots, \mathbb{E}[\mathbf{G}_{\mathbf{h}_M}]\}$ . For simplicity, we analyze the case where  $\mathbf{G}$  is

a scalar ( $G$ ).

**Lemma 1** (Maximization bias). *Consider a sampler samples a set of hyperparameters  $\mathbf{H} = \{\mathbf{h}_1, \dots, \mathbf{h}_M\}$ , which introduces a set of random variables  $\mathcal{G} = \{G_{\mathbf{h}_1}, \dots, G_{\mathbf{h}_M}\}$  with a set of true means  $\mathbb{E}[\mathcal{G}] = \{\mu_{\mathbf{h}_1}, \dots, \mu_{\mathbf{h}_M}\}$ . A set of unbiased estimators  $\hat{\mathcal{G}} = \{\hat{\mu}_{\mathbf{h}_1}, \dots, \hat{\mu}_{\mathbf{h}_M}\}$  is used to estimate the underlying true means. Then,*

$$\mathbb{E}[\max_{\mathbf{h} \in \mathbf{H}} \hat{\mathcal{G}}] - \max_{\mathbf{h} \in \mathbf{H}} \mathbb{E}[\mathcal{G}] \geq 0.$$

*Proof.* Let  $\mathbf{h}_{\hat{\mu}} = \arg \max_{\mathbf{h} \in \mathbf{H}} \hat{\mathcal{G}}$  with the maximal estimator value  $\max_{\mathbf{h} \in \mathbf{H}} \hat{\mathcal{G}} = \hat{\mu}_{\mathbf{h}_{\hat{\mu}}}$ , and  $\mathbf{h}_{\mu} = \arg \max_{\mathbf{h} \in \mathbf{H}} \mathbb{E}[\mathcal{G}]$  with the maximal true value  $\max_{\mathbf{h} \in \mathbf{H}} \mathbb{E}[\mathcal{G}] = \mu_{\mathbf{h}_{\mu}}$ , then for any  $\hat{\mathcal{G}}$ ,

$$\hat{\mu}_{\mathbf{h}_{\hat{\mu}}} - \mu_{\mathbf{h}_{\mu}} \geq \hat{\mu}_{\mathbf{h}_{\mu}} - \mu_{\mathbf{h}_{\mu}}.$$

The inequality follows from the definition of  $\hat{\mu}_{\mathbf{h}_{\hat{\mu}}}$ . Taking expectations with respect to  $\hat{\mathcal{G}}$  results in

$$\mathbb{E}[\hat{\mu}_{\mathbf{h}_{\hat{\mu}}} - \mu_{\mathbf{h}_{\mu}}] \geq \mathbb{E}[\hat{\mu}_{\mathbf{h}_{\mu}} - \mu_{\mathbf{h}_{\mu}}] = 0.$$

The equality holds because  $\hat{\mu}_{\mathbf{h}_{\mu}}$  is an unbiased estimator. Therefore, we have

$$\mathbb{E}[\hat{\mu}_{\mathbf{h}_{\hat{\mu}}}] - \mu_{\mathbf{h}_{\mu}} \geq 0.$$

As  $\max_{\mathbf{h} \in \mathbf{H}} \hat{\mathcal{G}} = \hat{\mu}_{\mathbf{h}_{\hat{\mu}}}$  and  $\max_{\mathbf{h} \in \mathbf{H}} \mathbb{E}[\mathcal{G}] = \mu_{\mathbf{h}_{\mu}}$ , we have

$$\mathbb{E}[\max_{\mathbf{h} \in \mathbf{H}} \hat{\mathcal{G}}] - \max_{\mathbf{h} \in \mathbf{H}} \mathbb{E}[\mathcal{G}] \geq 0.$$

■

Lemma 1 tells us that if we make a decision based on a set of estimations, we will be disappointed in expectations. The natural follow-up question for the Lemma 1 is when the bias will be 0. Therefore, we derive the sufficient condition for this (see Lemma 2).

**Lemma 2** (Condition for zero maximization bias). *Consider a sampler samples a set of hyperparameters  $\mathbf{H} = \{\mathbf{h}_1, \dots, \mathbf{h}_M\}$ , which introduces a set of independent random variables  $\mathcal{G} = \{G_{\mathbf{h}_1}, \dots, G_{\mathbf{h}_M}\}$  with a set of true means  $\mathbb{E}[\mathcal{G}] = \{\mu_{\mathbf{h}_1}, \dots, \mu_{\mathbf{h}_M}\}$ .*

A set of independent unbiased estimators  $\hat{\mathcal{G}} = \{\hat{\mu}_{\mathbf{h}_1}, \dots, \hat{\mu}_{\mathbf{h}_M}\}$  is used to estimate the underlying true means. Let  $\max_{\mathbf{h} \in \mathbf{H}} \hat{\mathcal{G}} = \hat{\mu}_{\mathbf{h}_{\hat{\mu}}}$  and  $\max_{\mathbf{h} \in \mathbf{H}} \mathbb{E}[\mathcal{G}] = \mu_{\mathbf{h}_j}$ . Then,

$$\mathbb{E}[\hat{\mu}_{\mathbf{h}_{\hat{\mu}}}] - \mu_{\mathbf{h}_j} = 0$$

if for all  $i \neq j$

$$p(\hat{\mu}_{\mathbf{h}_j} - \hat{\mu}_{\mathbf{h}_i} > 0) = 1.$$

holds.

*Proof.* Let  $f_{\hat{\mu}_{\mathbf{h}_k}}(x)$ ,  $F_{\hat{\mu}_{\mathbf{h}_k}}(x)$  be the probability density function (PDF) and cumulative distribution function (CDF) of the random variable  $\hat{\mu}_{\mathbf{h}_k}$  for  $k = 1, \dots, M$ .

Suppose  $p(\hat{\mu}_{\mathbf{h}_j} - \hat{\mu}_{\mathbf{h}_i} > 0) = 1$  for  $i \neq j$ , meaning  $\hat{\mu}_{\mathbf{h}_i} < \hat{\mu}_{\mathbf{h}_j}$  for all  $i \neq j$ . Then, it follows

$$\begin{aligned} f_{\hat{\mu}_{\mathbf{h}_j}}(x) &> f_{\hat{\mu}_{\mathbf{h}_i}}(x), \forall x \quad \forall i \neq j \\ \iff F_{\hat{\mu}_{\mathbf{h}_i}}(x) &= 1, \forall x : F_{\hat{\mu}_{\mathbf{h}_j}}(x) > 0 \quad \forall i \neq j. \end{aligned}$$

Let  $F(x)$  be the CDF of the random variable  $\hat{\mu}_{\mathbf{h}_{\hat{\mu}}}$ , and we know  $\mathbb{E}[\hat{\mu}_{\mathbf{h}_{\hat{\mu}}}] - \mu_{\mathbf{h}_j} = 0$  if

$$\begin{aligned} F(x) &= F_{\hat{\mu}_{\mathbf{h}_j}}(x), \forall x \\ \iff \prod_{i=1}^M F_{\hat{\mu}_{\mathbf{h}_i}}(x) &= F_{\hat{\mu}_{\mathbf{h}_j}}(x) \\ \iff F_{\hat{\mu}_{\mathbf{h}_j}}(x) \prod_{i \neq j} F_{\hat{\mu}_{\mathbf{h}_i}}(x) &= F_{\hat{\mu}_{\mathbf{h}_j}}(x) \\ \iff \prod_{i \neq j} F_{\hat{\mu}_{\mathbf{h}_i}}(x) &= 1, \forall x : F_{\hat{\mu}_{\mathbf{h}_j}}(x) > 0. \end{aligned}$$

Therefore, if  $F_{\hat{\mu}_{\mathbf{h}_i}}(x) = 1, \forall x : F_{\hat{\mu}_{\mathbf{h}_j}}(x) > 0 \quad \forall i \neq j$ , then  $\prod_{i \neq j} F_{\hat{\mu}_{\mathbf{h}_i}}(x) = 1, \forall x : F_{\hat{\mu}_{\mathbf{h}_j}}(x) > 0$ , and thus  $\mathbb{E}[\hat{\mu}_{\mathbf{h}_{\hat{\mu}}}] - \mu_{\mathbf{h}_j} = 0$ . ■

Lemma 2 makes additional assumptions of independence of random variables in the set  $\mathcal{G}$  to ensure the independence of their corresponding estimators. Although those random variables are correlated in practice for BO methods due to active selection of  $\mathbf{h}$ , we can still make this assumption if a hyperparameter space is sufficiently large

and each HPs is sufficiently far away from the others. Lemma 2 indicates that if estimators are certain such that there is no chance that the estimators of alternatives are larger than the estimator of true best value, then maximization bias would be zero. To gain a better understanding of maximization bias, let us consider a simple example that generates zero maximization bias for a simple case (Corollary 1).

**Corollary 1** (Zero maximization bias). *Consider a sampler samples a set of hyperparameters  $\mathbf{H} = \{\mathbf{h}_1, \mathbf{h}_2\}$ , which introduces a set of independent random variables  $\mathcal{G} = \{G_{\mathbf{h}_1}, G_{\mathbf{h}_2}\}$  individually distributed from a Gaussian with means  $\mu_{\mathbf{h}_1}, \mu_{\mathbf{h}_2}$  and variances  $\sigma_{\mathbf{h}_1}^2, \sigma_{\mathbf{h}_2}^2$ , respectively. A set of independent sample mean estimators  $\hat{\mathcal{G}} = \{\hat{\mu}_{\mathbf{h}_1}, \hat{\mu}_{\mathbf{h}_2}\} = \{\frac{\sum_{s \in S_1} s}{N}, \frac{\sum_{s \in S_2} s}{N}\}$  is used to estimate the underlying true means, where  $S_1$  and  $S_2$  are sample sets containing samples drawn from the distributions of  $G_{\mathbf{h}_1}$  and  $G_{\mathbf{h}_2}$ , respectively. Then,*

$$\mathbb{E}[\max_{\mathbf{h} \in \mathbf{H}} \hat{\mathcal{G}}] - \max_{\mathbf{h} \in \mathbf{H}} \mathbb{E}[\mathcal{G}] = 0.$$

*if variances of sample mean estimators are 0.*

*Proof.* We know if  $s \sim \mathcal{N}(\mu, \sigma^2)$ , then the sample mean estimator  $\hat{\mu} \sim \mathcal{N}(\mu, \sigma^2/N)$  where  $N$  is the number of samples taken average. Therefore, we consider  $\hat{\mu}_{\mathbf{h}_1} \sim \mathcal{N}(\mu_{\mathbf{h}_1}, \sigma_{\mathbf{h}_1}^2/N)$  and  $\hat{\mu}_{\mathbf{h}_2} \sim \mathcal{N}(\mu_{\mathbf{h}_2}, \sigma_{\mathbf{h}_2}^2/N)$  with means and variances finite. Let  $i$  be a random variable such that  $\mathbf{h}_i = \arg \max_{\mathbf{h} \in \mathbf{H}} \hat{\mathcal{G}}$  with the maximal estimator value  $\max_{\mathbf{h} \in \mathbf{H}} \hat{\mathcal{G}} = \hat{\mu}_{\mathbf{h}_i}$ , and  $j$  be a constant such that  $\mathbf{h}_j = \arg \max_{\mathbf{h} \in \mathbf{H}} \mathbb{E}[\mathcal{G}]$  with the maximal true value  $\max_{\mathbf{h} \in \mathbf{H}} \mathbb{E}[\mathcal{G}] = \mu_{\mathbf{h}_j}$ , then

$$p(i = j) = p(\hat{\mu}_{\mathbf{h}_j} > \hat{\mu}_{\mathbf{h}_{i \in I \setminus j}}),$$

where  $I$  is a set containing all possible indices of  $i$ .

Let a random variable  $z = \hat{\mu}_{\mathbf{h}_j} - \hat{\mu}_{\mathbf{h}_{i \in I \setminus j}}$ , we know  $z \sim \mathcal{N}(\mu_{\mathbf{h}_j} - \mu_{\mathbf{h}_{i \in I \setminus j}}, \frac{\sigma_{\mathbf{h}_1}^2}{N} + \frac{\sigma_{\mathbf{h}_2}^2}{N})$ .

Then,

$$\begin{aligned} p(i = j) &= p(z > 0) \\ &= 1 - p(z \leq 0) \\ &= 1 - \Phi\left(\frac{-(\mu_{h_j} - \mu_{h_{i \in I \setminus j}})}{\frac{\sigma_{h_1}^2}{N} + \frac{\sigma_{h_2}^2}{N}}\right), \end{aligned}$$

where  $\Phi$  is the cumulative distribution function (CDF) of the standard normal distribution. Due to the fact that  $\Phi(x) \rightarrow 0$  as  $x$  goes to  $-\infty$ , to make  $p(i = j) = 1$  always happen, we need

$$\frac{\sigma_{h_1}^2}{N} + \frac{\sigma_{h_2}^2}{N} \rightarrow 0.$$

If  $p(i = j) = 1$ , based on Lemma 2, then

$$\mathbb{E}[\max_{h \in \mathbf{H}} \hat{\mathcal{G}}] - \max_{h \in \mathbf{H}} \mathbb{E}[\mathcal{G}] = 0.$$

■

Corollary 1 provides another derivation for zero maximization bias to have intuitive observations. One could notice that if variances of sample mean estimators are 0, then  $i$  would always be  $j$ . To relate to Lemma 2, this essentially makes sure the condition stated in Lemma 2 happens. Besides, Corollary 1 discloses that if we use mean estimators, we could attain zero maximization bias in the limit of the infinite number of samples to achieve zero variances of the estimators, which is infeasible.

In reality, we know that estimators typically have finite variances. However, we could only try to reduce the variance of an estimator to reduce maximization bias. A simple way one directly observes from the Corollary 1 is that we could increase  $N$  to reduce the variance of an estimator such that the estimator can be more certain. However, increasing the number of samples for each trial is computationally challenging. Therefore, we suggest the idea of only increasing the runs when necessary. A necessary scenario happens when the current best value is realized. Accordingly, an adaptive multiple runs algorithm (AMRA) is proposed (Algorithm 1). The main steps are explained as follows.

1. Initialize a couple of objects, including an HPO tool  $H$ , a decision-making

algorithm, a task (environment)  $\epsilon$ , and an estimator  $\hat{\rho}_N[\mathbf{G}_h]$  with  $\mathbf{h}$  hyperparameter and  $N$  the number of agent runs used for estimating  $\rho[\mathbf{G}_h]$ .

2. At each trial, the sampler samples  $\mathbf{h}$  and call the estimator  $\hat{\rho}_N[\mathbf{G}_h]$ . Then, it would have two cases:
  - If the estimated value is not the current best, the pair of observations  $(\mathbf{h}, \hat{\rho}[\mathbf{G}_h])$  is then added to the dataset ( $\hat{\rho}[\mathbf{G}_h]$  is assigned by  $\hat{\rho}_N[\mathbf{G}_h]$ ).
  - If the estimated value is the current best, we trigger more agent runs to have a more reliable approximation. Triggering more runs stops if the difference between the new approximation  $\hat{\rho}^{ss}[\mathbf{G}_h]$  and the old approximation  $\hat{\rho}^s[\mathbf{G}_h]$  is smaller than a threshold  $\delta$ , and then new estimate  $\hat{\rho}^{ss}[\mathbf{G}_h]$  is returned. Here we notate a superscript double  $s$  as the estimation from more samples, while a superscript single  $s$  is the estimation from fewer samples.
3. After running out of trials, the optimizer will return  $\mathbf{h}^*$  such that it gives the best  $\hat{\rho}[\mathbf{G}_h]$  collected so far.

Compared with normal settings, one generally uses a fixed number of evaluations at each trial, while we propose that we need to invest more resources into promising HPs to avoid optimization bias. We could potentially save computes by avoiding using too many resources to evaluate unpromising HPs. Manipulating the resource would possibly decrease the total optimization time. The big questions by far are how to design the threshold  $\delta$  and if the algorithm terminates. Let's try to answer this question by looking at Hoeffding's inequality (the Proposition 1).

**Proposition 1** (Hoeffding's inequality). *For independent and identically distributed random variables  $X_1, X_2, \dots, X_N$  where  $a \leq X_i \leq b$  for all  $i$  and  $N$  is the number of samples, let  $\bar{X} = \frac{1}{N} \sum_{i=1}^N X_i$  be the sample mean, and let  $\mu = \mathbb{E}[X_i]$  be the true mean. Then, for any  $\delta > 0$ ,*

$$\Pr \left( \left| \bar{X} - \mu \right| \geq \delta \right) \leq 2 \exp \left( -\frac{2N^2\delta^2}{(b-a)^2} \right).$$

*Proof.* The proof is in [43] ■



This proposition is useful because we know how to bound the probability of distance between the mean estimator and true mean beyond  $\delta$ . We can use the proposition as a heuristic to design  $\delta$ . To have the same upper bound with  $a$  and  $b$  fixed, we need larger  $N$  for smaller  $\delta$ , and smaller  $N$  for larger  $\delta$ . In both cases for  $\delta$ , the bound approaches 0 when  $N$  is large enough, which ensures the termination of our algorithm. To be more specific, if  $N$  is large enough, then both  $\hat{\rho}^s[\mathbf{G}_h]$  and  $\hat{\rho}^{ss}[\mathbf{G}_h]$  have a low probability that they are beyond the underlying mean by  $\delta$ ; it follows that the distance of  $\hat{\rho}^s[\mathbf{G}_h]$  and  $\hat{\rho}^{ss}[\mathbf{G}_h]$  is within  $\delta$  with a high probability. On the other hand, if the distribution is wide ( $|b - a|$  is large), then we need more samples to keep mean estimators accurate by a fixed error. In other words, for high-variance RL algorithms such as PPO, larger  $N$  is expected to probabilistic guarantee that a mean estimator not exceeding its mean by a threshold  $\delta$ . Then, in this case, if we do not have enough budget, probably  $\delta$  should be set to a higher value. To deal with different scales of evaluation metric  $G$ , one could, for example, use a fraction coefficient  $c$  times the upper bound of  $G$  as the threshold  $\delta$ . This allows the threshold to be determined at different scales.

## 4.2 Optimizing over Conditional Value at Risk

In section 2.2, we introduce the concept of conditional value at risk (CVaR). In particular, we use Figure 2.6 to illustrate the impact of alpha  $\alpha$  on CVaR. If one replaces the objective  $\mathbb{E}[\mathbf{G}|\mathbf{h}, \epsilon]$  with  $\text{CVaR}_\alpha[\mathbf{G}|\mathbf{h}, \epsilon]$ , it directly follows that  $\text{CVaR}_\alpha$  is changing the optimization landscape. That being said, by optimizing over CVaR, optimizers prefer the solutions having lower risk rather than having higher expectations. Changing the  $\alpha$  of CVaR to lower values makes regions having high risk less preferable even though their expectations are higher.

Similar to sample mean estimators, CVaR estimators can be computed purely based on historical data without making assumptions about underlying distributions of samples. The computing steps are summarized as follows:

1. Draw samples from a cost random variable.
2. Compute left side  $(1 - \alpha)$ -quantile of the samples, also termed value at risk (VaR) (see Equation 2.1).

---

**Algorithm 1** Adaptive Multiple Runs Algorithm

---

```
1: procedure ADAPTIVE MULTIPLE RUNS ALGORITHM
2:   Initialize an HPO tool  $H$ , estimator  $\hat{\rho}_N[\mathbf{G}_h]$  of an agent given a task  $\epsilon$ 
3:   Initialize number of basic agent runs  $N$  and incremented agent runs  $N^+$ 
4:   Initialize number of trial  $M$ , dataset  $\mathcal{D} = \emptyset$ , and  $\hat{\rho}^{ss}[\mathbf{G}_h] = \inf$ 
5:   for each trial do
6:      $\mathbf{h} \leftarrow H.sampler(\mathcal{D})$ 
7:     Reset  $N$  to the initial value ▷ Reset the value to basic agent runs
8:      $\hat{\rho}^s[\mathbf{G}_h] \leftarrow \hat{\rho}_N[\mathbf{G}_h]$ 
9:      $\hat{\rho}[\mathbf{G}_h] \leftarrow \hat{\rho}^s[\mathbf{G}_h]$ 
10:    if  $\hat{\rho}^s[\mathbf{G}_h]$  is current best trial then ▷ Trigger more runs
11:      while  $|\hat{\rho}^s[\mathbf{G}_h] - \hat{\rho}^{ss}[\mathbf{G}_h]| > \delta$  do
12:         $\hat{\rho}^s[\mathbf{G}_h] \leftarrow \hat{\rho}^{ss}[\mathbf{G}_h]$  except for the first iteration
13:         $N = N + N^+$ 
14:         $\hat{\rho}^{ss}[\mathbf{G}_h] \leftarrow \hat{\rho}_N[\mathbf{G}_h]$  ▷ Approximate with more samples
15:      end while
16:       $\hat{\rho}[\mathbf{G}_h] \leftarrow \hat{\rho}^{ss}[\mathbf{G}_h]$ 
17:    end if
18:     $\mathcal{D} \leftarrow \mathcal{D} \cup (\mathbf{h}, \hat{\rho}[\mathbf{G}_h])$ 
19:  end for
20:  return  $\mathbf{h}^* \leftarrow \arg \max_{\mathbf{h}} \{\hat{\rho}[\mathbf{G}_{h_1}], \dots, \hat{\rho}[\mathbf{G}_{h_M}]\}$ 
21: end procedure
```

---

- 
3. Compute the average for values above VaR (see Equation 2.2).

### 4.3 Summary

Our final proposed algorithm is called the adaptive multiple runs algorithm with conditional value at risk (AMRA with CVaR) (see Algorithm 2). We simply replace the risk functional estimator  $\hat{\rho}_N[\mathbf{G}_h]$  in Algorithm 1 with a CVaR estimator  $\text{CVaR}_N[\mathbf{G}_h]$ .

---

**Algorithm 2** Adaptive Multiple Runs Algorithm with Conditional Value at Risk

---

```

1: procedure ADAPTIVE MULTIPLE RUNS ALGORITHM WITH CONDITIONAL
   VALUE AT RISK
2:   Initialize an HPO tool  $H$ , estimator  $\text{CVaR}_N[\mathbf{G}_h]$  of an agent given a task  $\epsilon$ 
3:   Initialize number of basic agent runs  $N$  and incremented agent runs  $N^+$ 
4:   Initialize number of trial  $M$ , dataset  $\mathcal{D} = \emptyset$ , and  $\text{CVaR}^{ss}[\mathbf{G}_h] = \inf$ 
5:   for each trial do
6:      $\mathbf{h} \leftarrow H.\text{sampler}(\mathcal{D})$ 
7:     Reset  $N$  to the initial value ▷ Reset the value to basic agent runs
8:      $\text{CVaR}^s[\mathbf{G}_h] \leftarrow \text{CVaR}_N[\mathbf{G}_h]$ 
9:      $\text{CVaR}[\mathbf{G}_h] \leftarrow \text{CVaR}^s[\mathbf{G}_h]$ 
10:    if  $\text{CVaR}^s[\mathbf{G}_h]$  is current best trial then ▷ Trigger more runs
11:      while  $|\text{CVaR}^s[\mathbf{G}_h] - \text{CVaR}^{ss}[\mathbf{G}_h]| > \delta$  do
12:         $\text{CVaR}^s[\mathbf{G}_h] \leftarrow \text{CVaR}^{ss}[\mathbf{G}_h]$  except for the first iteration
13:         $N = N + N^+$ 
14:         $\text{CVaR}^{ss}[\mathbf{G}_h] \leftarrow \text{CVaR}_N[\mathbf{G}_h]$  ▷ Approximate with more samples
15:      end while
16:       $\text{CVaR}[\mathbf{G}_h] \leftarrow \text{CVaR}^{ss}[\mathbf{G}_h]$ 
17:    end if
18:     $\mathcal{D} \leftarrow \mathcal{D} \cup (\mathbf{h}, \text{CVaR}[\mathbf{G}_h])$ 
19:  end for
20:  return  $\mathbf{h}^* \leftarrow \arg \max_{\mathbf{h}} \{\text{CVaR}[\mathbf{G}_{h_1}], \dots, \text{CVaR}[\mathbf{G}_{h_M}]\}$ 
21: end procedure

```

---



## 5 Experiments

In this section, all the experiments are conducted using the TPE sampler, a BO-type sampler, as this class is the best for dealing with noisy data generated from decision-making algorithms [21]. We first look at the synthetic examples (Figure 5.1). Subsequently, we evaluate our method on learning-based control (GP-MPC) and RL algorithms (SAC, PPO, and DDPG). The code is available on the GitHub repository <sup>1</sup>.

We compare five different strategies under a maximum budget of  $M$  trials. The five strategies are categorized depending on how they define  $\hat{\rho}[G_{\mathbf{h}_i}]$ , which is described in the following. We use non-bold letters for  $G_{\mathbf{h}_i}$  because we consider a single objective optimization for all experiments. Note that we notate  $S_{G_{\mathbf{h}_i}}^j$  as  $j$ -th sample drawn from the random variable  $G_{\mathbf{h}_i}$  and that superscript  $j$  is omitted if there is only one sample drawn.

At trial  $i$ , after  $\mathbf{h}_i$  is sampled,  $\hat{\rho}[G_{\mathbf{h}_i}]$  is approximated by:

1. Naive Single Run:  $\hat{\rho}[G_{\mathbf{h}_i}] = S_{G_{\mathbf{h}_i}}$ . The risk functional is the expectation, and we use one sample to approximate it ( $\mathbb{E}[G_{\mathbf{h}_i}] \approx S_{G_{\mathbf{h}_i}}$ ).
2. Naive Multiple Runs:  $\hat{\rho}[G_{\mathbf{h}_i}] = \frac{\sum_j S_{G_{\mathbf{h}_i}}^j}{N}$ . The risk functional is the expectation and we use  $N > 1$  samples to approximate it ( $\mathbb{E}[G_{\mathbf{h}_i}] \approx \frac{\sum_j S_{G_{\mathbf{h}_i}}^j}{N}$ ). In this case,  $N$  is fixed *a priori*.
3. AMRA with Expectations (Algorithm 1 using expectation as the risk functional):  $\hat{\rho}[G_{\mathbf{h}_i}] = \frac{\sum_j S_{G_{\mathbf{h}_i}}^j}{N_i}$ . The expectation is estimated by  $N_i > 1$  samples ( $\mathbb{E}[G_{\mathbf{h}_i}] \approx \frac{\sum_j S_{G_{\mathbf{h}_i}}^j}{N_i}$ ), where  $N_i$  will be dynamically adapted.
4. Multiple Runs with CVaR:  $\hat{\rho}[G_{\mathbf{h}_i}] = \hat{\text{CVaR}}_N[G_{\mathbf{h}_i}]$ . Here we use a fixed number of samples  $N > 1$  to approximate the CVaR ( $\text{CVaR}[G_{\mathbf{h}_i}] \approx \hat{\text{CVaR}}_N[G_{\mathbf{h}_i}]$ ).

---

<sup>1</sup><https://github.com/middleleyuan/safe-control-gym>

5. AMRA with CVaR (Algorithm 2):  $\hat{\rho}[G_{\mathbf{h}_i}] = \hat{\text{CVaR}}_{N_i}[G_{\mathbf{h}_i}]$ . The value of CVaR is estimated by  $N_i > 2$  samples ( $\text{CVaR}[G_{\mathbf{h}_i}] \approx \hat{\text{CVaR}}_{N_i}[G_{\mathbf{h}_i}]$ ), where  $N_i$  will be dynamically adapted.

The first two strategies are common practice in HPO. They use a fixed number of agent runs for evaluating each  $\mathbf{h}$  during the HPO. The third strategy is our proposed algorithm (Algorithm 1) using the expectation as the risk functional. The fourth strategy is adapted from the second by changing the expectation to CVaR. The last strategy is our proposed algorithm (Algorithm 2) using CVaR as the risk functional. The HPs are denoted as  $h$  (Section 5.1) and  $\mathbf{h}$  (Section 5.2) depending on  $\mathbf{h}$  being a scalar or a vector, respectively.

## 5.1 Synthetic Examples

In Section 2.1, we showed the optimized HPs returned by HPO tools for 100 HPO runs with 350 trials in Figure 2.4. The TPE solver can consistently give optimal results if the evaluation metric  $G$  is deterministic. However, in practice  $G$  is highly stochastic, especially for decision-making algorithms. We are interested in the impact of noisy evaluations of  $G$  on the final HPs. Therefore, we create three different scenarios with varying noise distributions to compare the strategies discussed above (see Figure 5.1). Below, we discuss optimal/sub-optimal regions in the sense of expectations. That is, an optimal region means the area where it has a maximum expectation value. The left plot describes the case when there exists a sub-optimal region but low risk. The center plot showcases the situation when there exists an optimal region with low risk. The right plot has an objective function with noise uniformly distributed. The desired solution is defined as  $h^*$  such that  $\text{CVaR}_\alpha[G_{h^*}]$  is maximized. In other words, regions with better worst-case events are preferred. We conduct the HPO for 100 times for each strategy with 350 trials. The number of agent runs is set to  $N = 1$  in the first strategy,  $N = 4$  in the second strategy,  $N = 4$  and  $N^+ = 4$  in the third strategy,  $N = 8$  in the fourth strategy, and  $N = 8$  and  $N^+ = 8$  in the fifth strategy. We use more runs for the strategies that use CVaR to have improved estimates of the worst-case averages. We compute results using  $\alpha = 0.1$  and  $0.5$  for strategies using CVaR in order to observe the impact of  $\alpha$ . The threshold is set as the optimal expectation of  $G$  ( $\max \mathbb{E}[G(h)]$ ) times the fraction

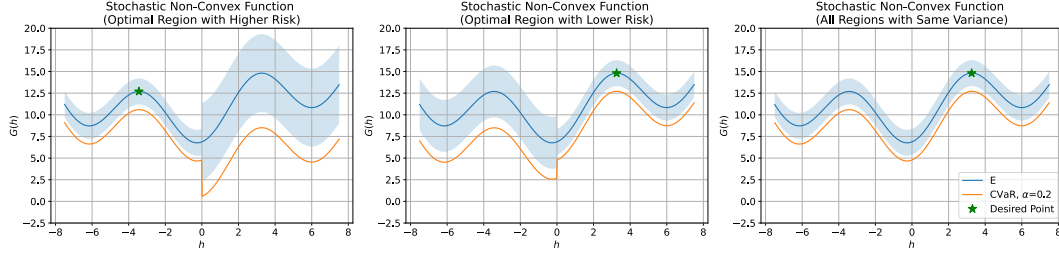


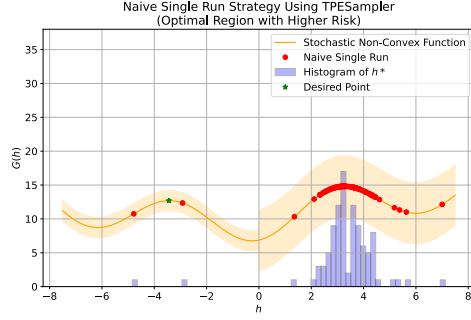
Figure 5.1: Non-convex function with noise. The blue curves are the expectations of the noisy functions, and the shaded areas are the standard deviation of the noise distributions. The orange curves are the lower bounds captured by CVaR. We design three different situations for noise. The green stars are located at  $(h^*, \mathbb{E}[G(h^*)])$  representing the lowest risk of  $h^*$  such that  $h^* = \arg \max_h \text{CVaR}_\alpha[G_h]$ .

coefficient  $c$ . Strategy three uses  $c = 0.01$ , while strategy five uses  $c = 0.001$ . The latter value is lower as CVaR estimators are high-variance, and thus, require more samples to increase the confidence in our approximations.

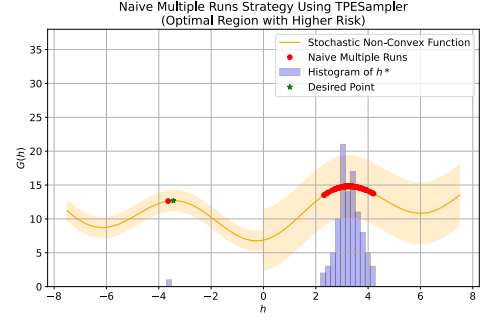
**Optimal region with higher risk:** We first present the first three strategies as they optimize over the expectations (see Figures 5.2a, 5.2b, and 5.2c). Almost all of the  $h^*$  lie in the high-risk region (noisy region) for the first three strategies. This indicates that expectations cannot quantify the risk of tail distributions. The results from strategy four are shown in Figures 5.2d and 5.2e. Indeed, the distribution of  $h^*$  is shifted to the left (low-risk region) compared to those in Figures 5.2a, 5.2b, and 5.2c. With a decreasing  $\alpha$ , the optimizer becomes more risk-aware. Finally, our proposed strategy (Algorithm 2) shows the best results in Figure 5.2g. The image shows 96% (96 out of 100 HPO runs) of solutions in the desired low-risk region, while Figure 5.4f shows less promising results with  $\alpha$  increased to 0.5. Comparing Figures 5.2f and 5.4g to Figures 5.2d and 5.2e, our method is better with the same value  $\alpha$  (more concentrated distributions of  $h^*$  on the low-risk region).

**Optimal region with lower risk:** Under this setting, in Figure 5.3a, we can observe that we attain a large proportion of sub-optimal solutions since an estimator using the naive single run strategy has a high chance of overestimating true expectations. This issue can be effectively addressed by either increasing the runs per

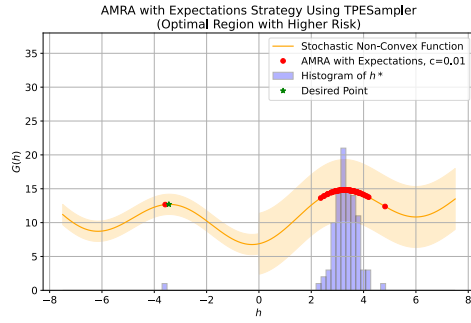
## 5 Experiments



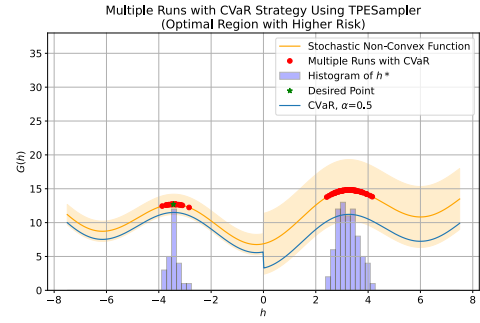
(a) Naive Single Run.



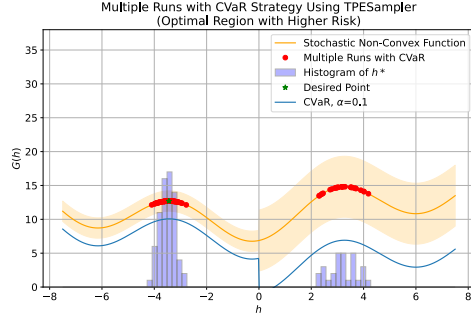
(b) Naive Multiple Runs.



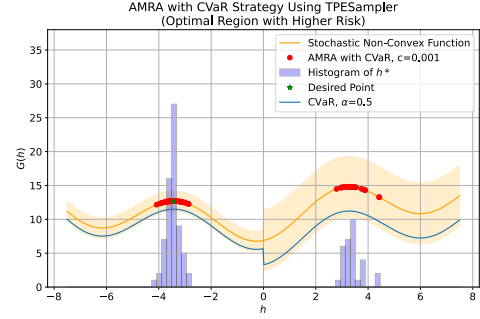
(c) AMRA with Expectations.



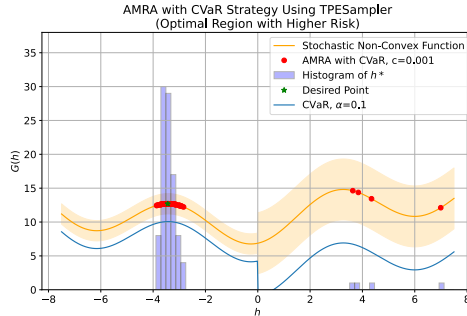
(d) Multiple Runs with CVaR ( $\alpha = 0.5$ ).



(e) Multiple Runs with CVaR ( $\alpha = 0.1$ ).



(f) AMRA with CVaR ( $\alpha = 0.5$ ).



(g) AMRA with CVaR ( $\alpha = 0.1$ ).

Figure 5.2: Strategy comparisons (optimal region with higher risk). The plots show that CVaR is capable of quantifying the risk of tail distributions and that CVaR with lower  $\alpha$  leads to more robust solutions.



configuration (Figure 5.3b) or our proposed Algorithm 1 using expectations as the risk functional (Figure 5.3c). However, this still leads to a few sub-optimal solutions. To deal with this, we can utilize the CVaR risk functional which yields a comparable robust performance as shown in Figures 5.3d and 5.3e as well as Figures 5.3f and 5.3g.

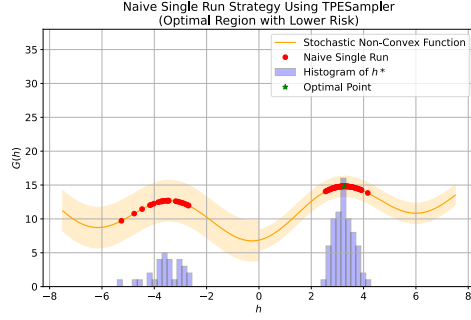
**All regions with the same variance:** This setting exactly matches the assumption of data for BO-based samplers, i.e., the observations are disturbed by some process noise with constant variance. It is worth noting that using CVaR compared to the expectation has no impact as the optimization landscape is just shifted equally in space from the counterpart of expectations. That is the reason why almost all strategies perform similarly except for the first strategy (see Figures 5.4a, 5.4b, 5.4c, 5.4d, 5.4e, 5.4f, and 5.4g ). This highlights that additional samples per set of HPs can reduce the optimization bias.

In the three cases, we demonstrate that our proposed methods are capable of finding low-risk solutions while reducing the maximization bias. If the noise distributions in decision-making algorithms are as in the third scenario, conventional approaches can achieve comparable results. However, if the distribution profiles match the first or second cases, then common approaches may not work well. We summarize our recommendations in Table 5.1. Our method to increase the agent runs when a promising set of HPs is drawn reduces the noise in the observation data, which is fed back to the BO sampler. This helps the BO sampler to construct an improved acquisition function to be used to actively select the next set of HPs, yielding a positive feedback loop.

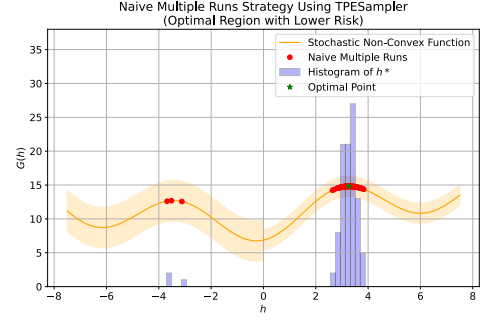
Noise Scenario	Our Recommendation
Optimal region with higher risk	AMRA with CVaR
Optimal region with lower risk	All strategies with CVaR
All regions with same variance	All strategies except for the first

Table 5.1: Our recommendations for different noise scenarios. Although we give recommendations for each noise scenario, the noise scenario at hand is usually unknown *a priori*.

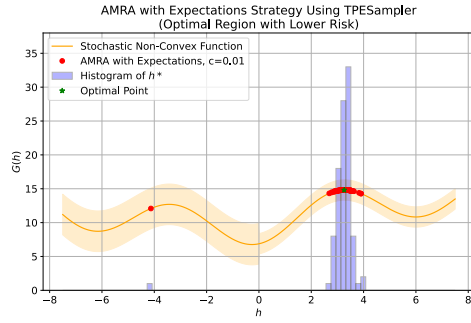
## 5 Experiments



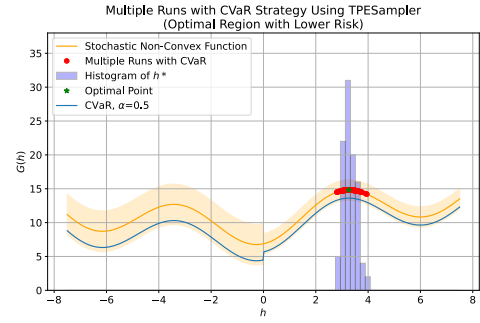
(a) Naive Single Run.



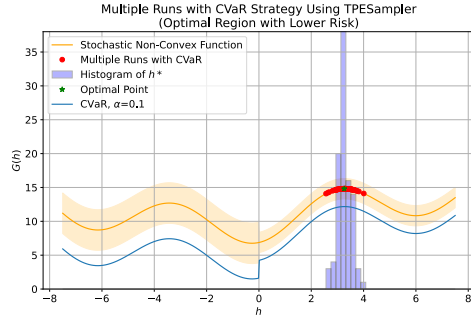
(b) Naive Multiple Runs.



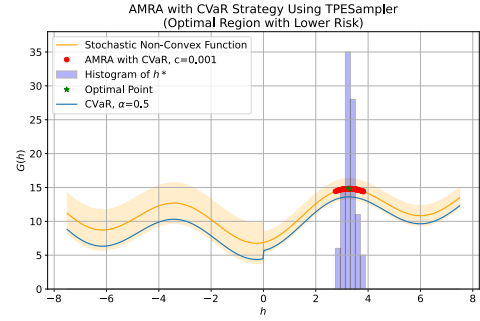
(c) AMRA with Expectations.



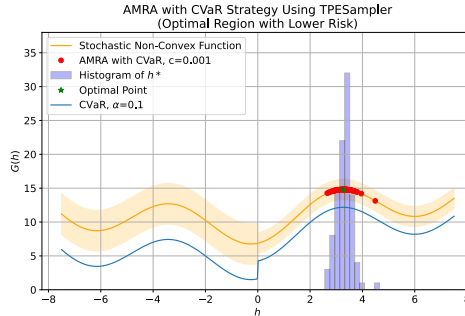
(d) Multiple Runs with CVaR ( $\alpha = 0.5$ ).



(e) Multiple Runs with CVaR ( $\alpha = 0.1$ ).

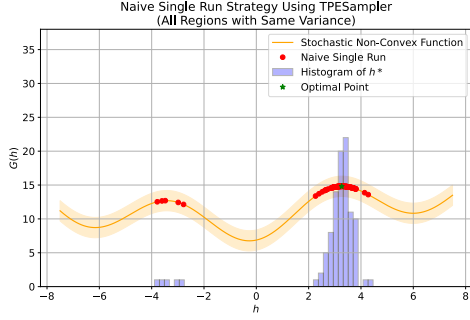


(f) AMRA with CVaR ( $\alpha = 0.5$ ).

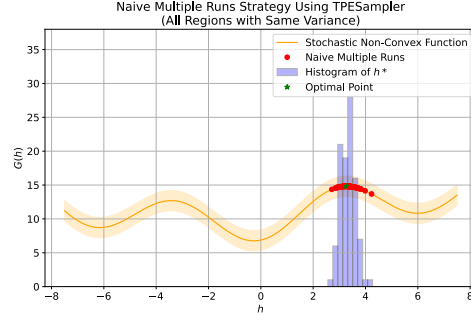


(g) AMRA with CVaR ( $\alpha = 0.1$ ).

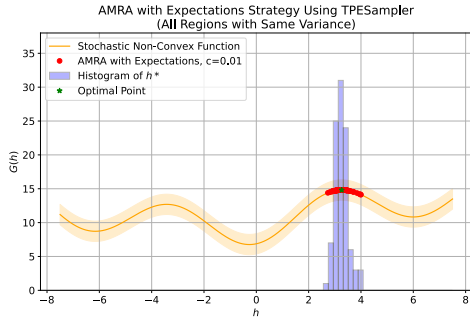
Figure 5.3: Strategy comparisons (optimal region with lower risk). Figures 5.3b and 5.3c demonstrate that the results are improved if one does not use a single evaluation per trial (Figure 5.3a). The solutions consistently lie in the low-risk region if CVaR is used (Figures 5.3d, 5.3e, 5.3f, and 5.3g).



(a) Naive Single Run.



(b) Naive Multiple Runs.



(c) AMRA with Expectations.

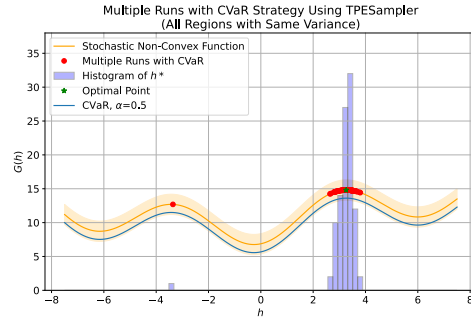
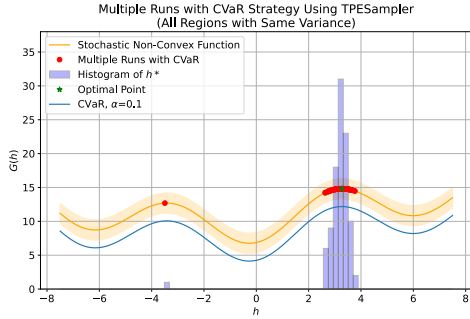
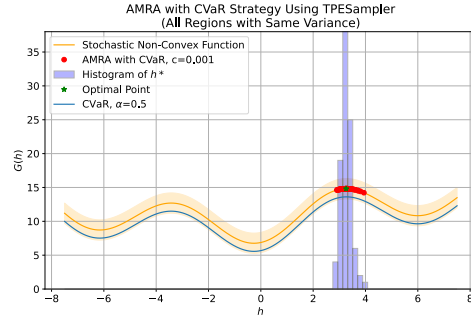
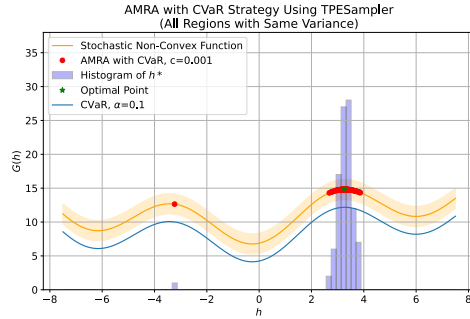
(d) Multiple Runs with CVaR ( $\alpha = 0.5$ ).(e) Multiple Runs with CVaR ( $\alpha = 0.1$ ).(f) AMRA with CVaR ( $\alpha = 0.5$ ).(g) AMRA with CVaR ( $\alpha = 0.1$ ).

Figure 5.4: Strategy comparisons (all regions with the same variance).

The first strategy gives acceptable results demonstrating that BO methods are able to deal with data disturbed by noise with uniform variance. Common approaches (i.e., the second strategy) can achieve good results in such scenarios. Figure 5.4f shows that our method leads to a narrower distribution of  $h^*$  at  $\alpha = 0.5$ .

## 5.2 Decision-Making Algorithms

To perform HPO for learning-based control and RL algorithms, we need to unify their evaluation metric  $G$ . According to the prior work in [5], we define the reward per step  $i$  as

$$J_i^R = \exp\left(-\left(\mathbf{x}_i^T \mathbf{Q} \mathbf{x}_i + \mathbf{u}_i^T \mathbf{R} \mathbf{u}_i\right)\right) \quad (5.1)$$

with  $\mathbf{x}_i$  and  $\mathbf{u}_i$  as the state and action for the system at step  $i$ , and  $\mathbf{Q}$  and  $\mathbf{R}$  are the cost matrices. By doing so, we can bound the reward per step on the interval  $[0, 1]$ , which enables more stable RL training. In our experiments,  $\mathbf{Q}$  and  $\mathbf{R}$  have the same values across all the algorithms for fairness and consistency for learning-based control and RL approaches. As a consequence, the evaluation metric we consider is

$$G = \sum_{i=1}^L J_i^R \quad (5.2)$$

with  $L > 0$  the control horizon so that the upper bound of  $G$  is  $L$ .

We evaluate and compare our five strategies on the platform *safe-control-gym* (SCG)<sup>2</sup>. The experiment pipeline is the following (see also Figure 5.5):

1. As a start, we specify a task, an algorithm, and its hyperparameter search space. In our experiments, the task is the Cartpole system. The decision-making algorithms are DDPG, PPO, SAC, and GP-MPC, and their corresponding HP search spaces are given in Section 2.4.
2. The HPO tool is then used to perform HPO until it runs out of the budget, which is defined as the maximum number of trials. Regarding the sampler, we use the TPE sampler, which samples one  $\mathbf{h}$  per trial. Afterward, the agent is trained on a given environment (task) given sampled HPs as one agent run, of which a validation reward is reported by taking the average over multiple episodes as one sample drawn from the random variable of  $G_{\mathbf{h}}$ . As the last step, the risk functional estimate  $\hat{\rho}[G_{\mathbf{h}}]$  is returned back to the sampler after a certain number of agent runs. The number of agent runs depends on  $N$  and  $N_i$ , which are introduced previously for the five strategies.

---

<sup>2</sup><https://github.com/utiasDSL/safe-control-gym>

3. After HPO is finished, the final  $\mathbf{h}^*$  is extracted. To compute the validated performance of  $\mathbf{h}^*$ , first of all, we use  $\mathbf{h}^*$  to train the control algorithm using unseen seeds. Secondly, the trained policy is evaluated with respect to the evaluation metric defined in Equation 5.2 using another pool of unseen seeds.

**Setup:** We run 3 HPO runs per strategy per decision-making algorithm. For naive single run, naive multiple runs, and multiple runs with CVaR, the maximum number of trials is defined such that their maximum numbers of agent runs are equal. For example, these three strategies for GP-MPC use 240 agent runs (trials times the number of agent runs per trial). For AMRA with expectations and AMRA with CVaR, the maximum number of trials is set such that their maximum number of agent runs is likely below the maximum number of agent runs due to our proposed convergence condition. For instance, the 80 trials are set to these two strategies such that the total number of agent runs is likely below 240 (see Table 5.2). This allows us to show the effectiveness of our algorithm and answer the question of the trade-off between quantity and quality. To elaborate, the second (naive multiple runs) and fourth (multiple runs with CVaR) approaches use a fixed number of evaluations per trial to ensure the quality of estimations, while the first strategy (naive single run) focuses on quantity and does not concern the high variability in a single evaluation per configuration. Our method suggests mixing these two extreme worlds, i.e., we allocate some agent runs to evaluations for less promising configurations, and we invest more runs into promising ones. The number of trials is set differently across different algorithms as they require different amounts of compute time for one agent run. The control frequency is set to 15 Hz, and the control horizon is set to 10 seconds; thus, the upper bound of the reward is 150. The threshold is computed by the upper bound of the reward times a fraction coefficient:  $\delta = 150 \cdot c$  so that  $\delta$  can be determined relative to the upper bound of a reward. Table 5.2 lists the overall summary for the parameters of each strategy of each algorithm. Each HPO run for a strategy for a decision-making algorithm returns an  $\mathbf{h}^*$ . We then evaluate the validated performance using another 50 unseen seeds for RL methods and 20 unseen seeds for GP-MPC (the evaluation pipeline described in Figure 5.5) as the current implementation for GP-MPC is computationally expensive.

The results for each considered algorithm are shown in Figure 5.6. We use boxen

plots to visualize the validated performance distribution for  $\mathbf{h}^*$  extracted from each strategy. The first strategy, naive single run, extracts HPs that have wide distributions for PPO and GP-MPC. The second strategy, naive multiple runs, gives a risky performance for some unseen seeds for DDPG. The third strategy, AMRA with expectations, shows generally unreliable HPs. The fourth strategy, multiple runs with CVaR, demonstrates a more risk-aware optimization as it only performs badly for GP-MPC likely due to too few agent runs. This is because extreme events typically can be captured well if the sample size is sufficiently large. Our proposed method, AMRA with CVaR (Algorithm 2), is the only one that has consistent tight distributions across all selected control algorithms.

One may challenge that what if we remove the extremely bad cases? Does the proposed method still be better? If we repeat the experiments, what results are expected to be observed? To answer these questions, we use robust statistics, stratified bootstrap confidence intervals, proposed by [23] to account for uncertainty in aggregate performance across HPO runs. To elaborate, we divide agent runs into three buckets, and each bucket represents one single HPO run. Each bucket has 50 agent runs (20 for GP-MPC) evaluating optimized HPs given by the corresponding HPO run. We sample with replacement each of those buckets considering their individual sizes. This avoids computing aggregate performance on the subset of HPO runs. We use the percentiles' method provided by their open-source library RLiable<sup>3</sup> to calculate bootstrap 95% confidence intervals (CIs) in terms of the metrics of median, interquartile mean (IQM), and mean. This quantifies the confidence in terms of the performance captured by each selected metric for given  $\mathbf{h}^*$ . The CIs for different control algorithms for different strategies are reported in Figure 5.7. We show that, in general, our strategy leads to small CIs, meaning that if we use our method to perform HPO for decision-making algorithms, then we can be confident about the performance described by selected metrics given extracted  $\mathbf{h}^*$ . Although the aggregate performance for statistical metrics we consider is not necessarily the best, it is generally robust and consistent across unseen seeds, while other strategies may be unreliable in some cases. Moreover, the authors in [23] propose to report the more reliable statistical metric, IQM, because it results in narrower CIs than the median for the widely-used RL benchmark ALE [44]. In general, the statement that

---

<sup>3</sup><https://github.com/google-research/rliable/tree/master>

IQM yields smaller CIs holds true for the first four strategies. Nonetheless, Figure 5.7 shows that our method AMRA with CVaR leads to similar sizes of CIs for the median and IQM, which suggests that our method could enable robust evaluation even if the median is used as the metric. Most importantly, the optimization time for our strategy remains consistently the lowest or second lowest among all strategies (see Figure 5.8), highlighting that there exists a sweet spot for trading off quality and quantity.

Despite the promising results of our method, our algorithm indeed introduces another set of HPs (listed in Table 5.2), still, the soundness of the results appears to be not particularly sensitive to these additional HPs. For example, using the limited budget for the number of agent runs for GP-MPC, 240 agent runs, to the much greater budget of 1600 agent runs for PPO in both settings, our strategy leads to small CIs among five strategies within each algorithm. Regarding  $\alpha$ , in general, the lower the value the  $\alpha$  is, the more samples per trial are needed to have more reliable estimates. In terms of the fraction coefficient, we test from  $c \approx 0.017$  for DDPG and SAC to  $c \approx 0.033$  for PPO, and we obtain similarly promising results. Note that the smaller the value of  $c$  is, the more optimization time is required (see Figure 5.8 to compare) since the procedure needs more samples to satisfy the convergence criteria.

In terms of the difficulty of HPO across selected control algorithms, our experiments show that obtaining robust HPs for GP-MPC and PPO is more difficult. They require more optimization time (see Figure 5.8), but the performance is generally not robust against unseen seeds (see Figure 5.6). In addition, the validated distributions of  $\mathbf{h}^*$  for GP-MPC are typically wider than the rest of the selected control algorithms, suggesting that its HPs may be more sensitive to unseen seeds. On the other hand, HPs of SAC seem to be easier to optimize since the extracted  $\mathbf{h}^*$  is typically more generalized (see Figure 5.6), while requiring a reasonable amount of HPO time (see Figure 5.8).

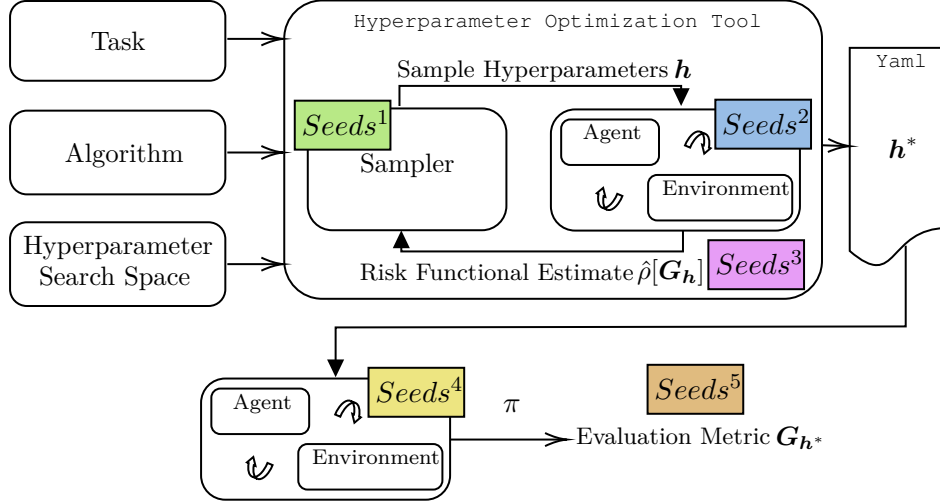
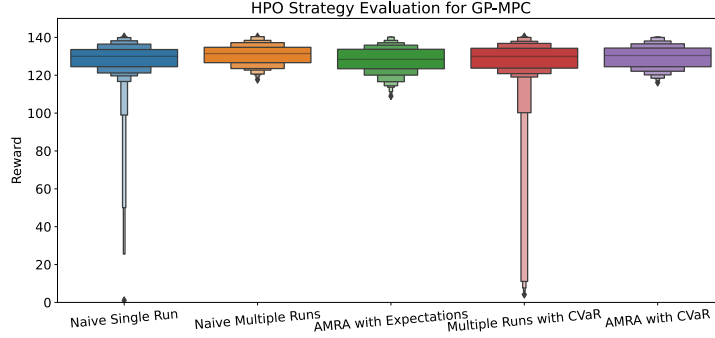
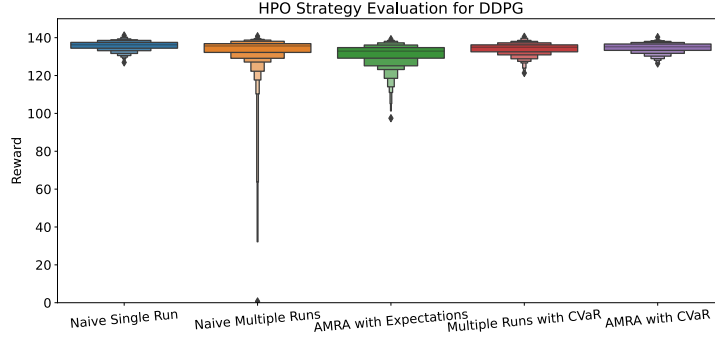


Figure 5.5: Evaluation pipeline for hyperparameter optimization strategy. An HPO tool takes a task, a control algorithm, and HP search spaces for the control algorithm as inputs. Given a finite number of trials, the tool returns  $h^*$ , which is evaluated by assessing the validation  $G_{h^*}$  using unseen seeds. The pool of  $Seeds^1$  is for controlling the randomness in a sampler. The disjoint sets of  $Seeds^2$ ,  $Seeds^3$ ,  $Seeds^4$ , and  $Seeds^5$  are created for the purpose of computing validation performance of  $h^*$ . To elaborate, during HPO, an agent is trained on a task using a seed  $\in Seeds^2$ , and the trained policy is evaluated using a seed  $\in Seeds^3$ . After HPO, an agent is trained given  $h^*$  using a seed  $\in Seeds^4$ , and the trained policy is evaluated using a seed  $\in Seeds^5$ . This gives us a pipeline to test the robustness of  $h^*$  against unseen seeds.

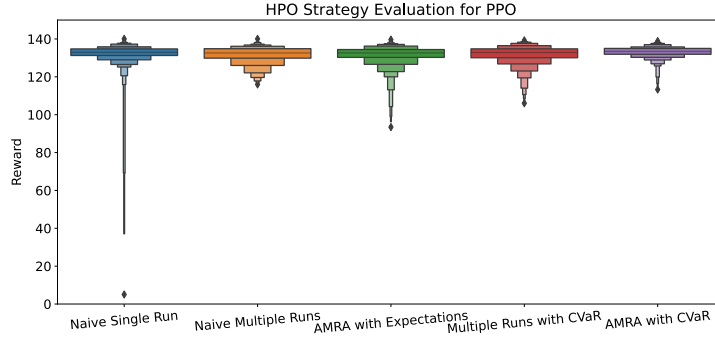




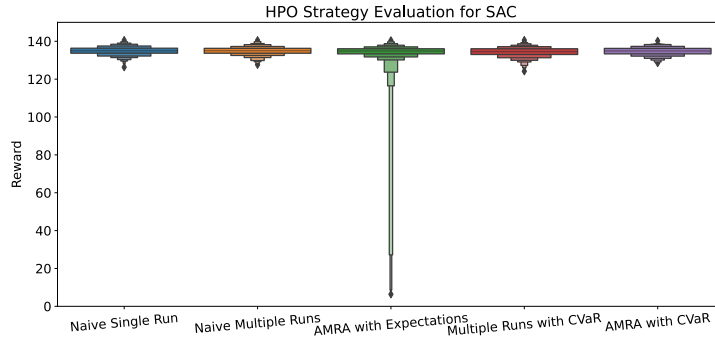
(a) HPO Strategy Evaluation for GP-MPC.



(b) HPO Strategy Evaluation for DDPG.



(c) HPO Strategy Evaluation for PPO.



(d) HPO Strategy Evaluation for SAC.

Figure 5.6: HPO strategy evaluation for GP-MPC, DDPG, PPO, SAC. Our proposed method, AMRA with CVaR, is the only one that has consistent tight distributions across all control algorithms.

## 5 Experiments

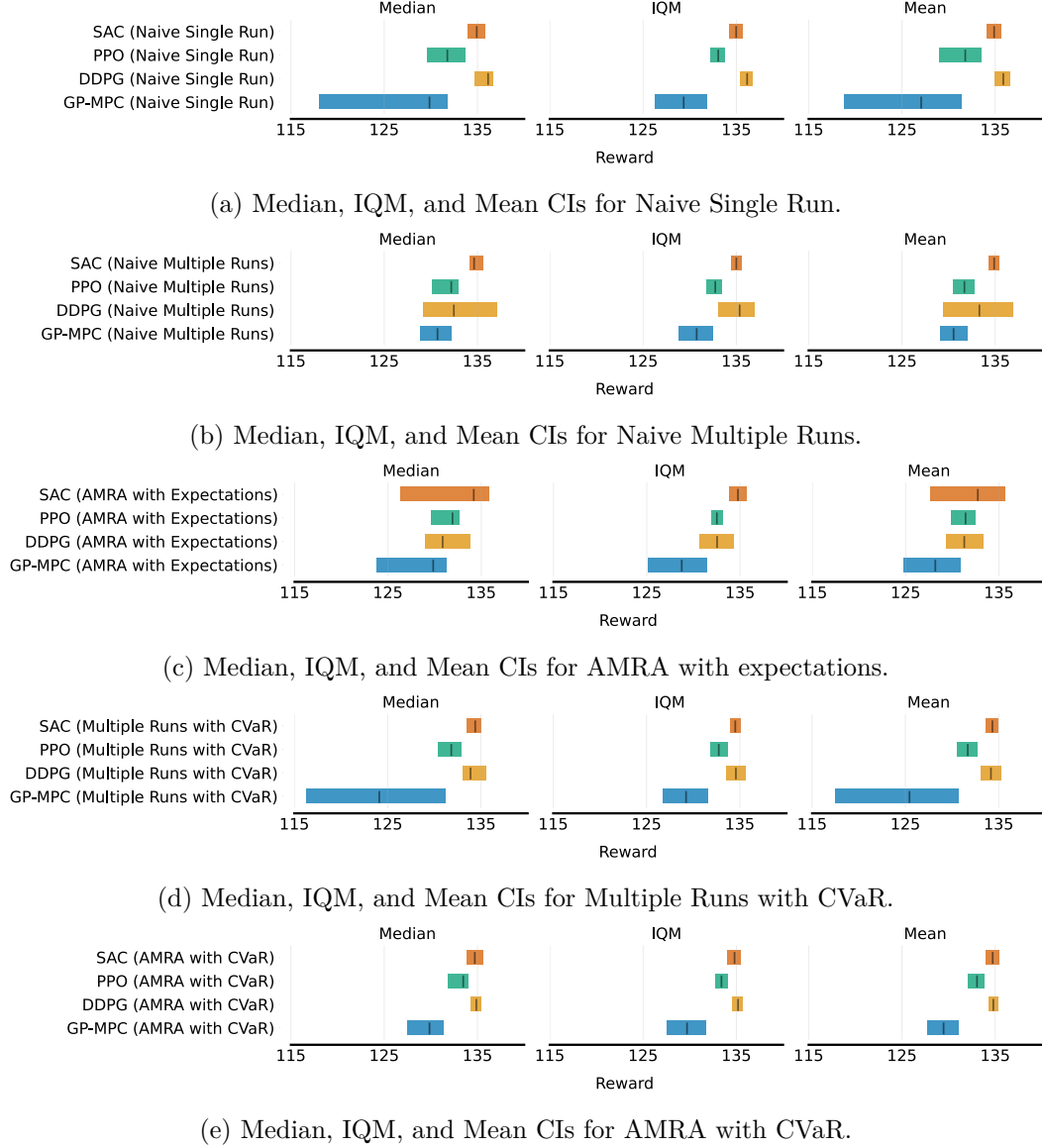
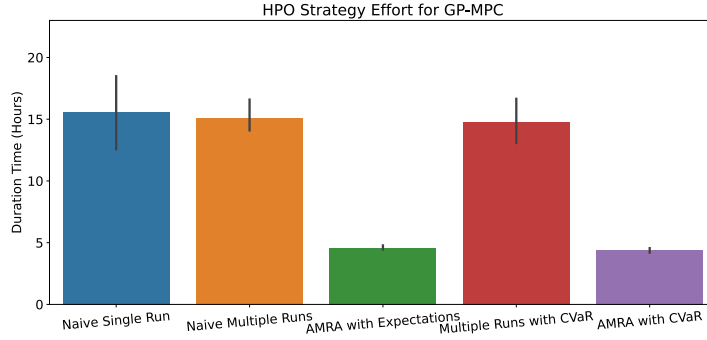
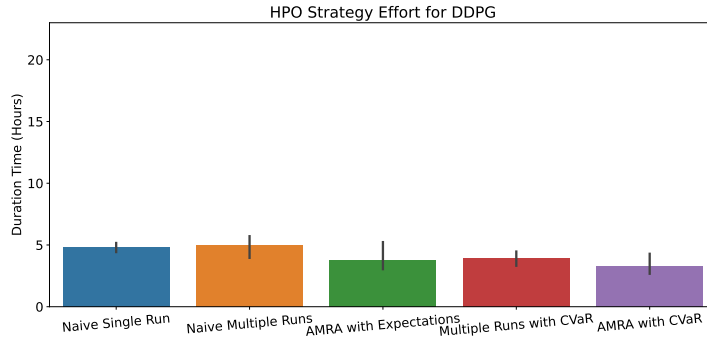


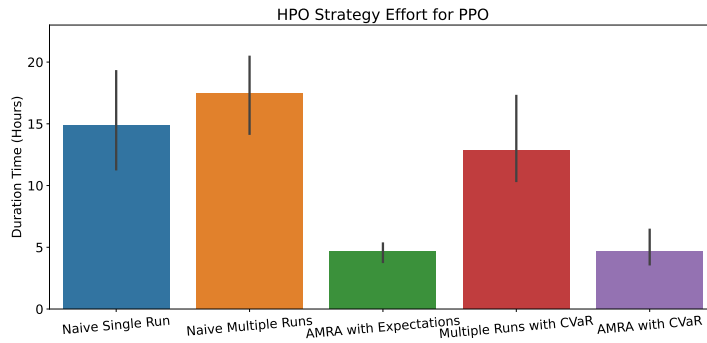
Figure 5.7: Median, IQM, and Mean CIs across strategies. AMRA with CVaR demonstrates consistent small CIs across strategies, control algorithms, and the selected metrics; thus, it is the most robust method.



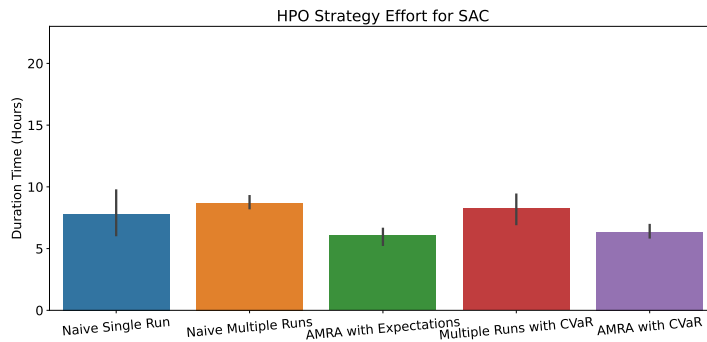
(a) HPO Strategy Effort for GP-MPC.



(b) HPO Strategy Effort for DDPG.



(c) HPO Strategy Effort for PPO.



(d) HPO Strategy Effort for SAC.

Figure 5.8: HPO strategy effort for GP-MPC, DDPG, PPO, SAC. In general, adaptive multiple runs strategies (AMRA with expectations and AMRA with CVaR) can lead to more efficient optimizations.

Algorithm	Strategy	Trials	$N$	$N^+$	$\alpha$	$c$	$\delta$
GP-MPC	Naive Single Run	240	1				
	Naive Multiple Runs	80	3				
	AMRA with Expectations	80	1	1		0.023	3.5
	Multiple Runs with CVaR	80	3		0.3		
	AMRA with CVaR	80	1	1	0.3	0.020	3
DDPG	Naive Single Run	400	1				
	Naive Multiple Runs	80	5				
	AMRA with Expectations	80	3	3		0.016	2.5
	Multiple Runs with CVaR	80	5		0.2		
	AMRA with CVaR	80	3	3	0.2	0.016	2.5
PPO	Naive Single Run	1600	1				
	Naive Multiple Runs	160	10				
	AMRA with Expectations	160	4	4		0.033	5
	Multiple Runs with CVaR	160	10		0.5		
	AMRA with CVaR	160	4	4	0.5	0.033	5
SAC	Naive Single Run	400	1				
	Naive Multiple Runs	80	5				
	AMRA with Expectations	80	3	3		0.016	2.5
	Multiple Runs with CVaR	80	5		0.2		
	AMRA with CVaR	80	3	3	0.2	0.016	2.5

Table 5.2: Parameter summary for strategies. For the strategies with a constant number of agent runs, they require the maximum number of agent runs as their budget (i.e., the numbers of trials times  $N$  are equal). For the strategies dynamically adapting the number of agent runs,  $N$ ,  $N^+$ , and  $\delta(=150 \cdot c)$  are set such that they do not necessarily require the maximum number of agent runs.

## 6 Conclusion and Outlook

Robust HPs are essential to decision-making algorithms for two main reasons. Firstly, imagine we would like to do learning on real hardware; optimized HPs from simulations ideally exhibit robustness against randomness so that HPs can be better transferred from simulations to the real world. Secondly, when it comes to benchmarking, aggregate results across different tasks are usually reported to assess the advancement of RL algorithms, and it is only feasible to run a handful of agent runs per task for most research projects [23]. In this scenario, if we have HPs that give us a wide distribution with high-expectation performance, then a handful of agent runs cannot represent the true ability of the algorithm, hindering fair comparisons. This is because one may sample data points from tail distributions, meaning overestimation/underestimation of the performance of algorithms, and it is hard to argue and draw a conclusion for comparisons. Those are the reasons why optimized HPs that give generalized, robust, and consistent performance are of crucial importance.

In this thesis, we show that our algorithm is able to extract robust HPs on synthetic examples, especially in the case when a sub-optimal region has a lower risk. After demonstrating our proposed algorithm on synthetic examples, we evaluate our algorithms on GP-MPC, DDPG, PPO, and SAC, where GP-MPC is a representative learning-based control algorithm and the rest are popular RL algorithms. To test the robustness of extracted  $\mathbf{h}^*$  against randomness, the evaluation pipeline (Figure 5.5) is applied. The results show that our method is capable of producing  $\mathbf{h}^*$  that is robust against unseen seeds, while others may result in poor performance given some random seeds. We then investigate the uncertainty in the performance given optimized HPs. This is addressed by using a robust statistic tool: stratified bootstrap confidence intervals (CIs). The technique uses sampling with replacement to mimic the replications of experiments, which is robust to outliers as they are less likely to be drawn. We demonstrate that our method leads to small CIs across all selected control algorithms and the selected metrics, meaning that the aggre-

gate performance of the control algorithms is more certain. From this, we conclude that  $\mathbf{h}^*$  gives consistently good performance. This yields the desired property for benchmarking. In Section 6.1, we discuss what is still missing for benchmarking. In Section 6.2, we summarize the possible future improvements for this work.

## 6.1 Toward Benchmarking Learning-Based Control and Reinforcement Learning

In this thesis, we tackle one of the challenges for fairer comparisons of learning-based control and RL algorithms. However, there are at least two challenges ahead. First of all, it is still unclear how to quantify the impact of prior knowledge on learning-based control and RL algorithms because different approaches usually make different assumptions about the prior knowledge of the system and/or the available experimental data. For instance, GP-MPC is equipped with mathematical differential equations of the model as the prior knowledge. In RL, prior usually comes in the form of a policy [45]. This policy prior can be a pre-defined/pre-trained policy used to guide the exploration of a student policy. For example, the authors in [46] propose a general algorithm termed jump-start RL to encode a policy prior to RL algorithms. There is a lack of metrics and methods that support us in translating a model prior to a policy prior, making prior studies for learning-based control and RL algorithms challenging. Second, different metrics usually emphasize different aspects of the control algorithm's performance; it will generally not be sufficient to report only the restricted number of metrics. The authors in [23] propose metrics such as IQM, performance profile, optimality gap, and probability of improvement and summarize alternative metrics such as difficulty progress and superhuman probability. It will be worthwhile to consider those metrics in future comparisons.

## 6.2 Future Work

In this work, we propose a general and practical algorithm to reduce maximization bias and risk with only a few extra lines of code needed to be added to conventional approaches. We perform a theoretical analysis on maximization bias and the sufficient condition for zero maximization bias. The condition is derived under the

assumption of independent random variables  $\mathbf{G}_h$ . It would be interesting to relax the assumption as it violates the behavior of BO-type methods due to the active selection of  $\mathbf{h}$ . Moreover, we do not derive the bound for maximization bias, but we believe that one could derive bounds using the tail probability bound of multivariate normal distributions [47], [48] if  $\mathbf{G}_h$  is assumed to be Gaussian distributed, which we leave to future works. The main idea of our algorithm is to increase the number of runs when the current best happens, and it stops increasing the number of runs when the distance between new and old approximations is smaller than a pre-defined threshold. This stopping criterion can be changed to more sophisticated approaches. For instance, the bound in Hoeffding's inequality can be used to determine the confidence of a sample mean not exceeding the underlying mean, and then the procedure breaks the loop once the value of the bound is lower than a pre-defined threshold. Nevertheless, using the Hoeffding bound requires prior knowledge of the range of the random variable, which is often unknown ahead of time. In our work, we show the powerfulness of combining the technique of reducing maximization bias and the risk functional (CVaR). Leveraging different risk functional such as exponential utility can be another explore direction. Another interesting future work is to evaluate our method on well-established and widely-used RL platforms such as OpenAI gym and compare the CIs reported in [23]. Finally, our work is motivated by the challenge of benchmarking learning-based control and RL algorithms. The natural next step is to utilize the technique developed in this thesis to systemically extract optimized HPs that yield consistent good performance for each algorithm for fairer comparisons. In this situation, the parameters chosen for our strategy (see Table 5.2) need to be fair across all decision-making algorithms. That is to say,  $N$ ,  $N^+$ ,  $\alpha$ , and  $\delta$  should be set equally so that we can justify that all  $\mathbf{h}^*$  are extracted by using the same technique for reducing maximization bias while considering the same level of risk at the same time.





# List of Algorithms

1	Adaptive Multiple Runs Algorithm . . . . .	34
2	Adaptive Multiple Runs Algorithm with Conditional Value at Risk .	35



## List of Figures

2.1	Hyperparameter optimization tool . . . . .	9
2.2	Hyperparameter optimization tool applied on learning-based control or reinforcement learning . . . . .	9
2.3	Non-convex function . . . . .	10
2.4	HPO results for non-convex function . . . . .	10
2.5	Trade-off between budget and $G$ . . . . .	11
2.6	CVaR for different $\alpha$ . . . . .	12
2.7	Visualization for risk functionals . . . . .	13
5.1	Non-convex function with noise . . . . .	39
5.2	Strategy comparisons (optimal region with higher risk) . . . . .	40
5.3	Strategy comparisons (optimal region with lower risk) . . . . .	42
5.4	Strategy comparisons (all regions with the same variance) . . . . .	43
5.5	Evaluation pipeline for hyperparameter optimization strategy . . . . .	48
5.6	HPO strategy evaluation for GP-MPC, DDPG, PPO, SAC . . . . .	49
5.7	Median, IQM, and Mean CIs across strategies . . . . .	50
5.8	HPO strategy effort for GP-MPC, DDPG, PPO, SAC . . . . .	51



# List of Tables

2.1	Hyperparameters for SAC . . . . .	16
2.2	Hyperparameters for PPO . . . . .	17
2.3	Hyperparameters for SAC . . . . .	18
2.4	Hyperparameters for GP-MPC . . . . .	20
5.1	Our recommendations for different noise scenarios . . . . .	41
5.2	Parameter summary for strategies . . . . .	52



## Bibliography

- [1] C. Gehring, P. Fankhauser, L. Isler, *et al.*, “Anymal in the field: Solving industrial inspection of an offshore hvdc platform with a quadrupedal robot,” in *Field and Service Robotics: Results of the 12th International Conference*, Springer, 2021, pp. 247–260.
- [2] C. Duan, S. Junginger, J. Huang, K. Jin, and K. Thurow, “Deep learning for visual slam in transportation robotics: A review,” *Transportation Safety and Environment*, vol. 1, no. 3, pp. 177–184, 2019.
- [3] D. S. Smys and D. G. Ranganathan, “Robot assisted sensing, control and manufacture in automobile industry,” *Journal of IoT in Social, Mobile, Analytics, and Cloud*, vol. 1, no. 3, pp. 180–187, 2019.
- [4] L. Brunke, M. Greeff, A. W. Hall, *et al.*, “Safe learning in robotics: From learning-based control to safe reinforcement learning,” *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, pp. 411–444, 2022.
- [5] Z. Yuan, A. W. Hall, S. Zhou, *et al.*, “Safe-control-gym: A unified benchmark suite for safe learning-based control and reinforcement learning in robotics,” *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 11 142–11 149, 2022.
- [6] C. J. Ostafew, A. P. Schoellig, and T. D. Barfoot, “Robust constrained learning-based nmpe enabling reliable mobile robot path tracking,” *The International Journal of Robotics Research*, vol. 35, no. 13, pp. 1547–1563, 2016.
- [7] L. Hewing, J. Kabzan, and M. N. Zeilinger, “Cautious model predictive control using gaussian process regression,” *IEEE Transactions on Control Systems Technology*, vol. 28, no. 6, pp. 2736–2743, 2019.
- [8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.

- [9] F. Hutter, “Automated configuration of algorithms for solving hard computational problems,” Ph.D. dissertation, University of British Columbia, 2009.
- [10] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” *Journal of machine learning research*, vol. 13, no. 2, 2012.
- [11] T. Yu and H. Zhu, “Hyper-parameter optimization: A review of algorithms and applications,” *arXiv preprint arXiv:2003.05689*, 2020.
- [12] B. Bischl, M. Binder, M. Lang, *et al.*, “Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges,” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, e1484, 2021.
- [13] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, “Deep reinforcement learning that matters,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.
- [14] R. Islam, P. Henderson, M. Gomrokchi, and D. Precup, “Reproducibility of benchmarked deep reinforcement learning tasks for continuous control,” *arXiv preprint arXiv:1708.04133*, 2017.
- [15] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A next-generation hyperparameter optimization framework,” in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2623–2631.
- [16] M. Lindauer, K. Eggensperger, M. Feurer, *et al.*, “Smac3: A versatile bayesian optimization package for hyperparameter optimization,” *J. Mach. Learn. Res.*, vol. 23, no. 54, pp. 1–9, 2022.
- [17] T. Eimer, M. Lindauer, and R. Raileanu, “Hyperparameters in reinforcement learning and how to tune them,” *arXiv preprint arXiv:2306.01324*, 2023.
- [18] B. Zhang, R. Rajan, L. Pineda, *et al.*, “On the importance of hyperparameter optimization for model-based reinforcement learning,” in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2021, pp. 4015–4023.
- [19] K. Eggensperger, M. Lindauer, and F. Hutter, “Pitfalls and best practices in algorithm configuration,” *Journal of Artificial Intelligence Research*, vol. 64, pp. 861–893, 2019.



- 
- [20] F. Hutter, H. H. Hoos, and T. Stützle, “Automatic algorithm configuration based on local search,” in *Aaai*, vol. 7, 2007, pp. 1152–1157.
  - [21] J. Parker-Holder, R. Rajan, X. Song, *et al.*, “Automated reinforcement learning (autorl): A survey and open problems,” *Journal of Artificial Intelligence Research*, vol. 74, pp. 517–568, 2022.
  - [22] C. Colas, O. Sigaud, and P.-Y. Oudeyer, “How many random seeds? statistical power analysis in deep reinforcement learning experiments,” *arXiv preprint arXiv:1806.08295*, 2018.
  - [23] R. Agarwal, M. Schwarzer, P. S. Castro, A. C. Courville, and M. Bellemare, “Deep reinforcement learning at the edge of the statistical precipice,” *Advances in neural information processing systems*, vol. 34, pp. 29 304–29 320, 2021.
  - [24] A. Patterson, S. Neumann, M. White, and A. White, “Empirical design in reinforcement learning,” *arXiv preprint arXiv:2304.01315*, 2023.
  - [25] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, “Algorithms for hyperparameter optimization,” *Advances in neural information processing systems*, vol. 24, 2011.
  - [26] M. Samvelyan, R. Kirk, V. Kurin, *et al.*, “Minihack the planet: A sandbox for open-ended reinforcement learning research,” *arXiv preprint arXiv:2109.13202*, 2021.
  - [27] Y. Wang and M. P. Chapman, “Risk-averse autonomous systems: A brief history and recent developments from the perspective of optimal control,” *Artificial Intelligence*, vol. 311, p. 103 743, 2022.
  - [28] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” in *International conference on machine learning*, Pmlr, 2014, pp. 387–395.
  - [29] L. Engstrom, A. Ilyas, S. Santurkar, *et al.*, “Implementation matters in deep policy gradients: A case study on ppo and trpo,” *arXiv preprint arXiv:2005.12729*, 2020.
  - [30] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *International conference on machine learning*, PMLR, 2018, pp. 1861–1870.

- [31] B. Efron, “Bootstrap methods: Another look at the jackknife,” in *Breakthroughs in statistics: Methodology and distribution*, Springer, pp. 569–593.
- [32] D. R. Jones, M. Schonlau, and W. J. Welch, “Efficient global optimization of expensive black-box functions,” *Journal of Global optimization*, vol. 13, pp. 455–492, 1998.
- [33] H. J. Kushner, “A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise,” 1964.
- [34] J. Snoek, H. Larochelle, and R. P. Adams, “Practical bayesian optimization of machine learning algorithms,” *Advances in neural information processing systems*, vol. 25, 2012.
- [35] J. T. Springenberg, A. Klein, S. Falkner, and F. Hutter, “Bayesian optimization with robust bayesian neural networks,” *Advances in neural information processing systems*, vol. 29, 2016.
- [36] Y. Ozaki, Y. Tanigaki, S. Watanabe, and M. Onishi, “Multiobjective tree-structured parzen estimator for computationally expensive optimization problems,” in *Proceedings of the 2020 genetic and evolutionary computation conference*, 2020, pp. 533–541.
- [37] Y. Ozaki, Y. Tanigaki, S. Watanabe, M. Nomura, and M. Onishi, “Multi-objective tree-structured parzen estimator,” *Journal of Artificial Intelligence Research*, vol. 73, pp. 1209–1250, 2022.
- [38] B. Letham, B. Karrer, G. Ottoni, and E. Bakshy, “Constrained bayesian optimization with noisy experiments,” 2019.
- [39] S. Watanabe, “Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance,” *arXiv preprint arXiv:2304.11127*, 2023.
- [40] J. Wu and P. Frazier, “The parallel knowledge gradient method for batch bayesian optimization,” *Advances in neural information processing systems*, vol. 29, 2016.
- [41] M. Jaderberg, V. Dalibard, S. Osindero, *et al.*, “Population based training of neural networks,” *arXiv preprint arXiv:1711.09846*, 2017.

- [42] M. Birattari, “The problem of tuning metaheuristics as seen from a machine learning perspective,” 2004.
- [43] W. Hoeffding, “Probability inequalities for sums of bounded random variables,” *The collected works of Wassily Hoeffding*, pp. 409–426, 1994.
- [44] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, “The arcade learning environment: An evaluation platform for general agents,” *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.
- [45] D. Tirumala, A. Galashov, H. Noh, *et al.*, “Behavior priors for efficient reinforcement learning,” *The Journal of Machine Learning Research*, vol. 23, no. 1, pp. 9989–10 056, 2022.
- [46] I. Uchendu, T. Xiao, Y. Lu, *et al.*, “Jump-start reinforcement learning,” *arXiv preprint arXiv:2204.02372*, 2022.
- [47] I. R. Savage, “Mills’ ratio for multivariate normal distributions,” *J. Res. Nat. Bur. Standards Sect. B*, vol. 66, no. 3, pp. 93–96, 1962.
- [48] E. Hashorva and J. Hüsler, “On multivariate gaussian tails,” *Annals of the Institute of Statistical Mathematics*, vol. 55, pp. 507–522, 2003.