Gray-Box Gaussian Processes for Automated Reinforcement Learning

Gresa Shala[1], **André Biedenkapp**[1], **Frank Hutter**[1,2], **Josif Grabocka**[1]
[1]University of Freiburg, [2]Bosch Center for Artificial Intelligence
{shalag,biedenka,fh,grabocka}@cs.uni-freiburg.de

Abstract

Despite having achieved spectacular milestones in an array of important real-world applications, most Reinforcement Learning (RL) methods are very brittle concerning their hyperparameters. Notwithstanding the crucial importance of setting the hyperparameters in training state-of-the-art agents, the task of hyperparameter optimization (HPO) in RL is understudied. In this paper, we propose a novel gray-box Bayesian Optimization technique for HPO in RL, that enriches Gaussian Processes with reward curve estimations based on generalized logistic functions. We thus not only reason about the performance of learning algorithms, transferring information across configurations but also about epochs of the learning algorithm. In a very large-scale experimental protocol, comprising 5 popular RL methods (DDPG, A2C, PPO, SAC, TD3), 22 environments (OpenAI Gym: Mujoco, Atari, Classic Control), and 7 HPO baselines, we demonstrate that our method significantly outperforms current HPO practices in RL.

1 Introduction

While Reinforcement Learning (RL) has celebrated amazing successes in many applications [1–5], it remains very brittle [6, 7]. The successes of RL are achieved by leading experts in the field with many years of expertise in the "art" of RL, but the field does not yet provide a technology that broadly yields successes off the shelf. A crucial hindrance for both broader impact and faster progress in research is that an RL algorithm that has been well-tuned for one problem does not necessarily work for another one; especially, optimal hyperparameters are environment-specific and must be carefully tuned in order to yield strong performance.

Despite the crucial importance of strong hyperparameter settings in RL [6, 8–10], the field of hyperparameter optimization (HPO) for RL is understudied. The field is largely dominated by manual tuning, computationally expensive hyperparameter sweeps, or population-based training which trains many agents in parallel that exchange hyperparameters and states [11][1]. While these methods are feasible for large industrial research labs, they are costly, substantially increase the CO2 footprint of artificial intelligence research [12], and make it very hard for smaller industrial and academic labs to partake in RL research. In this paper, we address this gap, developing a computationally efficient yet robust HPO method for RL.

The method we propose exploits the fact that reward curves tend to have similar shapes. As a result, future rewards an agent collects with a given hyperparameter setting can be predicted quite well based on initial rewards, providing a computationally cheap mechanism to compare hyperparameter settings against each other. We combine this insight in a novel gray-box Bayesian optimization method that includes a parametric reward curve extrapolation layer in a neural network for computing a Gaussian process kernel.

[1]For a more in-depth discussion of related work we refer to Appendix A

In a large-scale empirical evaluation using 5 popular RL methods (DDPG, A2C, PPO, SAC, TD3), 22 environments (OpenAI Gym: Mujoco, Atari, Classic Control), and 7 HPO baselines, we demonstrate that our resulting method, the *Reward-Curve Gaussian Process (RCGP)*, yields state-of-the-art performance across the board. In summary, our contributions are as follows:

- We introduce a novel method for extrapolating initial reward curves of RL agents with given hyperparameters based on partial learning curves with different hyperparameters.
- We introduce RCGP, a novel Bayesian optimization method that exploits such predictions to allocate more budget to the most promising hyperparameter settings.
- We carry out the most comprehensive experimental analysis of HPO for RL we are aware of to date (including 5 popular RL agents, 22 environments and 8 methods), concluding that RGCP sets a new state of the art for optimizing RL hyperparameters in low compute budgets.

To ensure reproducibility (another issue in modern RL) and broad use of RGCP, all our code is open-sourced at `https://github.com/releaunifreiburg/RCGP.git`.

2 Method Preliminaries

Hyperparameter Optimization (HPO) focuses on discovering the best hyperparameter configuration $\lambda \in \Lambda$ of a Machine Learning method. Gray-box HPO refers to the concept of cheaper and approximate evaluations of the performance of hyperparameter configurations. For example, we can approximately evaluate the final reward of the configurations of a deep RL method system after every epoch of stochastic gradient descent (gray-box evaluations) [13, 14], instead of waiting for the full convergence (black-box evaluations). The reward after a budget b (i.e. after b epochs) is defined as $R\left(\lambda, b\right) : \Lambda \times \mathbb{N} \rightarrow \mathbb{R}_{+}$. To address the noisieness of reward curves of RL algorithms we smooth the curves using a best-so-far transformation. We average rewards based on windows of h training steps, and select the highest reward at any past window, as:

$$R^{(\max)}\left(\lambda, b\right) = \max_{0 \leq b' < b-h} \frac{1}{h} \sum_{i=1}^{h} R(\lambda, b' + i). \tag{1}$$

From now on, we refer to the smoothed $R^{(\max)}$ as the reward R. In addition, let us define the cost (e.g., wall-clock time) of evaluating a configuration for a specific budget as $C\left(\lambda, b\right) : \Lambda \times \mathbb{N} \rightarrow \mathbb{R}_{+}$. We define the history of N evaluated configurations and the respective budget as $H^{(K)} = \left\{\left(\lambda_1, b_1, R\left(\lambda_1, b_1\right)\right), \ldots, \left(\lambda_K, b_K, R\left(\lambda_K, b_K\right)\right)\right\}$. A gray-box algorithm $\mathcal{A}$ is a policy that recommends the next configuration to evaluate and its budget as $\left(\lambda_{K+1}, b_{K+1}\right) := \mathcal{A}\left(H^{(K)}\right)$ where $\mathcal{A} : \left(\Lambda \times \mathbb{N} \times \mathbb{R}_{+}\right)^{K} \rightarrow \Lambda \times \mathbb{N}$. Gray-box HPO formally focuses on policies $\mathcal{A}$ that are sequentially executed for as many steps (denoted K) as needed to reach a total budget Ω. The best policy discovers the configuration with the largest reward at any budget, as $\arg\max_{\mathcal{A}} \max_{i=1,\ldots,K} R\left(\left(\lambda_{i+1}, b_{i+1}\right) := \mathcal{A}\left(H^{(i)}\right)\right)$ s.t. $K = \max_{j \in \mathbb{N}_{+}} \quad \Omega > \sum_{i=1}^{j} C\left(\lambda_i, b_i\right)$.

Bayesian optimization (BO) is a very popular HPO policy that sequentially recommends hyperparameters to evaluate. BO operates in sequences of two steps: by (i) fitting a probabilistic regression model to approximate the observed performances $R(\lambda, b)$ of the evaluated configurations and budgets in H; and (ii) applying an acquisition to select the next configuration to evaluate.

In the first step, we train Gaussian Processes [15] to approximate the observed performances (i.e. $R(\lambda, b) \approx \text{GP}\left(\lambda, b; \theta\right)$) by finding the optimal GP parameters θ^{*} via MLE:

$$\theta^{*}\left(H\right) := \arg\max_{\theta} \mathbb{E}_{(\lambda, b, R(\lambda, b)) \sim p_H} \ \log\left(R(\lambda, b) \mid \lambda, b\, ;\, \theta\right) \tag{2}$$

At the second step, we use an acquisition function $\alpha : \Lambda \times \mathbb{N}_{+} \rightarrow \mathbb{R}_{+}$ that scores how "well" a previously unevaluated configuration might perform at a future budget, based on the estimation of the GP fitted above. Typical acquisition functions, such as the Expected Improvement [15] recommend configurations with a high predicted performance (high GP posterior mean), but also explore configurations where the GP is uncertain on their performance (high posterior variance). A naive gray-box BO can be formalized as a special HPO policy, based on a fitted GP with parameters θ^* from a history of i evaluations $H^{(i)}$, that recommends the $(i+1)$-th configuration as:

$$(\lambda_{i+1}, b_{i+1}) := \mathcal{A}^{\text{BO}}\left(\theta^*\left(H^{(i)}\right)\right) := \underset{\lambda\in\Lambda, b\in\mathbb{N}_+}{\arg\max}\, \alpha\left(\lambda, b\,;\, \theta^*\left(H^{(i)}\right)\right) \tag{3}$$

3 Proposed Method

A multi-fidelity Gaussian Process can be modeled as a standard GP with an augmented feature vector $z := [\lambda, b] \in \Lambda \times \mathbb{N}$ [16–18]. We present the preliminaries of our method in Appendix 2. From the history of evaluations $H^{(K)}$ we define the training features $z_i = [\lambda_i, b_i]$ and their respective targets $y_i = R(\lambda_i, b_i)$ for $i \in \{1, \ldots, K\}$. A kernel function measures the similarity of features as $k(z_i, z_j) : (\Lambda \times \mathbb{N})^2 \rightarrow R_+$. The aim of the GP is to estimate the posterior distribution of the unknown target of a new observed test instance $z_* = [\lambda_*, b_*]$. The covariance matrix between training features' pairs is defined as $K(z, z) = [k(z_i, z_j)]_{\forall i,j}$. Similarly, the covariance between test-to-training features is $K(z_*, z) = [k(z_*, z_i)]_{\forall i}$, and the test-to-test one as $K(z_*, z_*) = k(z_*, z_*)$. Ultimately, the posterior prediction of the unknown test target is:

$$\mu(z_*) = K(z_*, z)\left(K(z, z) + \sigma_y^2 I\right)^{-1} y, \tag{4}$$

$$\sigma^2(z_*) = K(z_*, z_*) - K(z_*, z)\left(K(z, z) + \sigma_y^2 I\right)^{-1} K(z_*, z)^T. \tag{5}$$

It was recently pointed out that a sigmoidal relationship exists between the reward curve of Reinforcement Learning methods and the optimization budget [16]. In this paper, we model the reward curve $R(\lambda, b)$ of configuration λ at budget b as a generalized logistic function (Richard's curve) with five coefficients [19]. Furthermore, we do not naively fit one sigmoid function on each reward curve for each hyperparameter configuration. Instead, we propose to condition the sigmoid coefficients on the hyperparameter configurations within a multi-layer perceptron $g : \Lambda \rightarrow \mathbb{R}^5$ with weights w (where $g(\lambda, b; w)_i$ represents the i-th output neuron) as:

$$\hat{R}(\lambda, b; w) = g(\lambda; w)_1 + \frac{g(\lambda; w)_2 - g(\lambda; w)_1}{\left(1 + g(\lambda; w)_3\, e^{-g(\lambda; w)_4 b}\right)^{1/g(\lambda; w)_5}}. \tag{6}$$

In this paper, we propose a novel GP that exploits the pattern of the reward curve of the RL algorithm, by introducing the sigmoidal reward curve of (Equation 6). We augment the feature space with the estimation of the reward curve as $[\lambda, b] \rightarrow [\lambda, b, \hat{R}(\lambda, b; w)]$. Therefore, the kernel becomes:

$$k^{\text{our}}([\lambda_i, b_i], [\lambda_j, b_j]\,;\, w) = k\left([\lambda_i, b_i, \hat{R}(\lambda_i, b_i; w)], [\lambda_j, b_j, \hat{R}(\lambda_j, b_j; w)]\right). \tag{7}$$

We train the parameters w using the established machinery of kernel learning for GPs [20] and then use this GP for gray-box HPO [16, 17]. It is worth pointing out that the acquisition function of Bayesian optimization (BO) queries the GP's estimation of the final performance of an unevaluated configuration. For brevity, we omit the basics of BO here, however, we refer the interested reader to Snoek et al. [15]. In this context, the reward curve model of Equation 6 offers crucial information in estimating the full ($b^{\max}$) performance of an unknown configuration (i.e., $[\lambda, b^{\max}] \rightarrow [\lambda, b^{\max}, \hat{R}(\lambda, b^{\max}; w)]$), and enables the acquisition function of the BO algorithm (expected improvement at convergence) to discover performant hyper-parameter configurations.

Our novel gray-box HPO method is summarized by the pseudocode of Algorithm 1 in Appendix B.

4 Experimental Protocol

We focus on evaluating the performance of our proposed method, RCGP (Reward-Curve GP), for optimizing the hyperparameters of five popular model-free RL algorithms: PPO [21], A2C [22],

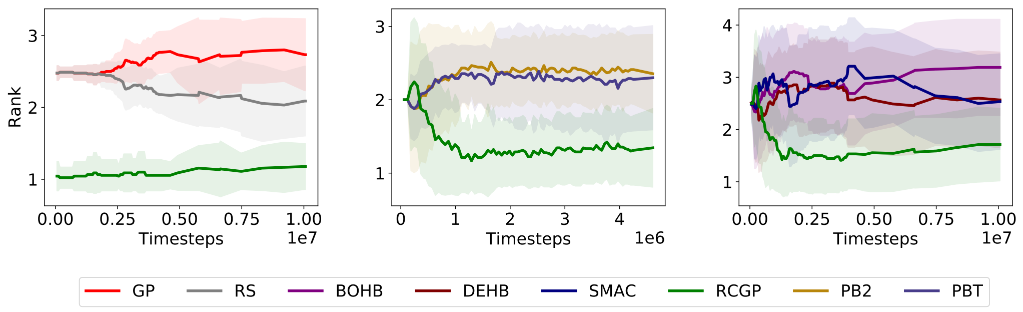

Figure 1: Rank comparison of (i) RS, GP, and RCGP; (ii) PBT, PB2, and RCGP, and (iii) BOHB, SMAC, DEHB, and RCGP for the **PPO** search space in the Atari environments.

DDPG [23], SAC [24], and TD3 [25]. In total, we consider 22 distinct Gym [26] environments, grouped into the Atari [27], Classic Control, and Mujoco [28] categories. We denote the search spaces for the hyperparameters of each RL algorithm, and the full list of environments and their respective action space types in Appendix C.

We evaluated static hyperparameter optimization (HPO) methods by querying AutoRL-Bench[2], which is a tabular benchmark for AutoRL that contains reward curves for three different random seeds belonging to runs of RL algorithms with every possible combination of hyperparameter values from the search spaces shown in Table 1 in Appendix C. For the dynamic HPO methods (PBT [11] and PB2 [29], details in Appendix C.1), we ran our own evaluations of the RL pipelines. In every environment, we set the budget for all baselines to the run-time equivalent of 10 full training procedures, based on the expected run-time figures of AutoRL-Bench. Each training procedure consists of 10^6 steps on the training environment. All methods are evaluated for three seeds in each environment and RL algorithm. The plots show the mean and standard deviations of the relative ranks of all methods, with the training timesteps in the x-axis. Furthermore, we included the code for evaluating the performance of all the HPO methods in our GitHub repo `https://anonymous.4open.science/r/RCGP-65CC`.

5 Research Hypotheses and Experimental Results

Hypothesis 1: *Using the reward curve information helps in discovering efficient hyperparameters for model-free RL algorithms in the low budget regime.*

We compare the performance of RCGP to RS and GP, as standard HPO baselines which do not utilize learning curve information. We evaluate RS and GP for 10 full RL algorithm runs. The leftmost plot in Figure 1 shows the performance comparison in terms of rank per timesteps in the training environment for the PPO search space in the Atari environments. Initially, RS, GP, and RCGP start the search with the same 4 hyperparameter configurations sampled uniformly at random. RCGP queries the learning curve of evaluation returns of these initial configurations for the smallest budget of 10^5 steps on the training environment. RS, and GP, being black-box optimization methods, query AutoRL-Bench for final evaluation returns after 10^6 training steps, for both the initial and subsequently suggested configurations.

Figures 4 to 6 (Appendix D) show the results on all the (search space, environment) combinations. In all the cases, RCGP outperforms RS and GP within the wall-clock time budget of our experimental protocol. We conclude that gray-box HPO is more efficient than black-box HPO in RL.

Hypothesis 2: *RCGP outperforms state-of-the-art multi-fidelity HPO methods in optimizing the hyperparameters of model-free RL algorithms.*

We compare the performance of RCGP to BOHB, SMAC and DEHB, as state-of-the-art multi-fidelity HPO baselines. We evaluate each method for the equivalent time of 10 full RL algorithm runs. Initially, all four methods start the search with the same 4 hyperparameter configurations

[2]`https://github.com/releaunifreiburg/AutoRL-Bench`

sampled uniformly at random. The middle plot in Figure 1 shows the performance comparison on the PPO search space for the Atari environments. Figures 7 to 9 (Appendix D) include the experiments on the PPO, A2C, DDPG, SAC, and TD3 search spaces for the Atari, Classic Control, and Mujoco environments. In all experiments, our method RCGP on average outperforms BOHB, SMAC, and DEHB in the low budget regime of up to 10 full function evaluations. We therefore conclude that RCGP sets the state-of-the-art in gray-box HPO for RL.

Hypothesis 3: *Our method outperforms PBT and PB2, the state-of-the-art HPO in RL.*

In this experiment, we compare our method RCGP to PBT and PB2, to assess the efficiency of our gray-box HPO technique against state-of-the-art dynamic HPO methods. For ensuring a fair comparison, we evaluated PB2 and PBT using the recommended population size of 4 [29], leading to a budget equivalence of 4 full training routines. In addition, all three optimization methods use an initial design consisting of the same 4 hyperparameter configurations sampled uniformly at random. The rightmost plot in Figure 1 shows the performance comparison on the PPO search space for the Atari environments. The associated experiments of Figures 10 to 12 (Appendix D), include experiments on all the (environment, algorithm) combinations. The plots show RCGP clearly outperforms PBT and PB2 in the low budget regime of up to 4 full function evaluations. Although PBT and PB2 are able to dynamically configure the hyperparameters of an RL algorithm, they require extensive parallel resources, and thus perform sub-optimally on the low budget regime.

6 Conclusion

Reinforcement Learning (RL) is one of the premier research sub-areas of Machine Learning, due to the impressive achievements of modern RL methods. Unfortunately, the performance of trained RL agents depends heavily on the choice of the methods' hyperparameters. In this paper we introduced a novel gray-box HPO method that fits Gaussian Processes (GP) to partially-observed reward curves. Our GP variant fuses hyperparameter configurations, budget information and reward curve models based on generalized logistic functions. In a large-scale experimental protocol we demonstrated that our proposed method significantly advances the state-of-the-art for HPO in RL. Especially, we largely outperform evolutionary search HPO methods in RL (PBT and PB2), as well as existing gray-box HPO techniques.

Acknowledgements

We acknowledge funding by European Research Council (ERC) Consolidator Grant "Deep Learning 2.0" (grant no. 101045765). Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the ERC. Neither the European Union nor the ERC can be held responsible for them. The authors acknowledge funding by The Carl Zeiss Foundation through the research network "Responsive and Scalable Learning for Robots Assisting Humans" (ReScaLe) of the University of Freiburg.

Funded by
the European Union

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015.

[2] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587): 484–489, 2016.

[3] OpenAI. Openai five. `https://blog.openai.com/openai-five/`, 2018.

[4] M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba. Learning dexterous in-hand manipulation. *International Journal of Robotics Research*, 39(1), 2020.

[5] J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de las Casas, C. Donner, L. Fritz, C. Galperti, A. Huber, J. Keeling, M. Tsimpoukelli, J. Kay, A. Merle, Jean-M. Moret, S. Noury, F. Pesamosca, D. Pfau, O. Sauter, C. Sommariva, S. Coda, B. Duval, A. Fasoli, P. Kohli, K. Kavukcuoglu, D. Hassabis, and M. Riedmiller. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022.

[6] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In S. McIlraith and K. Weinberger, editors, *Proceedings of the Thirty-Second Conference on Artificial Intelligence (AAAI'18)*. AAAI Press, 2018.

[7] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry. Implementation matters in deep RL: A case study on PPO and TRPO. In *Proceedings of the International Conference on Learning Representations (ICLR'20)*, 2020. Published online: `iclr.cc`.

[8] Y. Chen, A. Huang, Z. Wang, I. Antonoglou, J. Schrittwieser, D. Silver, and N. de Freitas. Bayesian optimization in alphago. *arXiv:1812.06855 [cs.LG]*, 2018.

[9] B. Zhang, R. Rajan, L. Pineda, N. Lambert, A. Biedenkapp, K. Chua, F. Hutter, and R. Calandra. On the importance of Hyperparameter Optimization for model-based reinforcement learning. In A. Banerjee and K. Fukumizu, editors, *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS'21)*, pages 4015–4023. Proceedings of Machine Learning Research, 2021.

[10] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R.ël Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, S. Gelly, and O. Bachem. What matters for on-policy deep actor-critic methods? a large-scale study. In *Proceedings of the International Conference on Learning Representations (ICLR'21)*, 2021. Published online: `iclr.cc`.

[11] M. Jaderberg, V. Dalibard, S. Osindero, W. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu. Population based training of neural networks. *arXiv:1711.09846 [cs.LG]*, 2017.

[12] P. Dhar. The carbon impact of artificial intelligence. *Nat. Mach. Intell.*, 2(8):423–425, 2020.

[13] K. Swersky, J. Snoek, and R. Adams. Freeze-thaw Bayesian optimization. *arXiv:1406.3896 [stats.ML]*, 2014.

[14] M. Wistuba, A. Kadra, and J. Grabocka. Supervising the multi-fidelity race of hyperparameter configurations. In *Proceedings of the 35th International Conference on Advances in Neural Information Processing Systems (NeurIPS'22)*. Curran Associates, 2022.

[15] J. Snoek, H. Larochelle, and R. Adams. Practical Bayesian optimization of machine learning algorithms. In P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Proceedings of the 25th International Conference on Advances in Neural Information Processing Systems (NeurIPS'12)*, pages 2960–2968. Curran Associates, 2012.

[16] V. Nguyen, S. Schulze, and M. A. Osborne. Bayesian optimization for iterative learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H. Lin, editors, *Proceedings of the 33rd International Conference on Advances in Neural Information Processing Systems (NeurIPS'20)*. Curran Associates, 2020.

[17] J. Song, Y. Chen, and Y Yue. A general framework for multi-fidelity bayesian optimization with gaussian processes. In K. Chaudhuri and M. Sugiyama, editors, *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS'19)*, volume 89, pages 3158–3167. PMLR, 2019.

[18] K. Swersky, J. Snoek, and R. P. Adams. Multi-task bayesian optimization. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, *Proceedings of the 26th International Conference on Advances in Neural Information Processing Systems (NeurIPS'13)*, volume 26, pages 2004–2012. Curran Associates, 2013.

[19] F. J. Richards. A Flexible Growth Function for Empirical Use. *Journal of Experimental Botany*, 10(2):290–301, 06 1959.

[20] A. G. Wilson, Z. Hu, R. Salakhutdinov, and E. P. Xing. Deep kernel learning. In A. Gretton and C. Robert, editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS'16)*, volume 51, pages 370–378. Proceedings of Machine Learning Research, 2016.

[21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347 [cs.LG]*, 2017.

[22] V. Mnih, A. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In M. Balcan and K. Weinberger, editors, *Proceedings of the 33rd International Conference on Machine Learning (ICML'17)*, volume 48, pages 1928–1937. Proceedings of Machine Learning Research, 2016.

[23] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR'16)*, 2016. Published online: `iclr.cc`.

[24] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, volume 80, pages 1861–1870. Proceedings of Machine Learning Research, 2018.

[25] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, volume 80, pages 1587–1596. Proceedings of Machine Learning Research, 2018.

[26] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI gym. *arXiv:1606.01540 [cs.LG]*, 2016.

[27] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal Artificial Intelligence Research*, 47:253–279, 2013.

[28] E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems (IROS'12)*, pages 5026–5033. ieeecis, IEEE, 2012.

[29] J. Parker-Holder, V. Nguyen, and S. J. Roberts. Provably efficient online Hyperparameter Optimization with population-based bandits. In H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H. Lin, editors, *Proceedings of the 33rd International Conference on Advances in Neural Information Processing Systems (NeurIPS'20)*. Curran Associates, 2020.

[30] F. Hutter, L. Kotthoff, and J. Vanschoren, editors. *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 2019. Available for free at http://automl.org/book.

[31] J. Parker-Holder, R. Rajan, X. Song, A. Biedenkapp, Y. Miao, T. Eimer, B. Zhang, V. Nguyen, R. Calandra, A. Faust, F. Hutter, and M. Lindauer. Automated reinforcement learning (AutoRL): A survey and open problems. *Journal of Artificial Intelligence Research (JAIR)*, 74: 517–568, 2022.

[32] Y. Miao, X. Song, J. D. Co-Reyes, D. Peng, S. Yue, E. Brevdo, and A. Faust. Differentiable architecture search for reinforcement learning. In M. van der Schaar, M. Lindauer, I. Guyon, and H. H. Hoos, editors, *Proceedings of the First Conference on Automated Machine Learning (AutoML'22)*, 2022.

[33] A. Gleave, M. D Dennis, S. Legg, S. Russell, and J. Leike. Quantifying differences in reward functions. In *Proceedings of the International Conference on Learning Representations (ICLR'21)*, 2021. Published online: `iclr.cc`.

[34] M. Feurer and F. Hutter. Hyperparameter Optimization. In Hutter et al. [30], chapter 1, pages 3–38. Available for free at http://automl.org/book.

[35] B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A. Boulesteix, D. Deng, and M. Lindauer. Hyperparameter Optimization: Foundations, algorithms, best practices and open challenges. *arXiv:2107.05847 [stat.ML]*, 2021.

[36] A. Eriksson, G. Capi, and K. Doya. Evolution of meta-parameters in reinforcement learning algorithm. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS'03)*, pages 412–417. IEEE, 2003.

[37] L. Hertel, P. Baldi, and D. L. Gillen. Quantity vs. quality: On hyperparameter optimization for deep reinforcement learning. *arXiv:2007.14604 [cs.LG]*, 2020.

[38] N. M. Ashraf, R. R. Mostafa, R. H. Sakr, and M. Z. Rashad. Optimizing hyperparameters of deep reinforcement learning for autonomous driving based on whale optimization algorithm. *PLOS ONE*, 16(6):1–24, 2021.

[39] J. Franke, G. Köhler, A. Biedenkapp, and F. Hutter. Sample-efficient automated deep reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR'21)*, 2021. Published online: `iclr.cc`.

[40] K. Kandasamy, G. Dasarathy, J. Schneider, and B. Póczos. Multi-fidelity Bayesian Optimisation with Continuous Approximations. In D. Precup and Y. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*, volume 70, pages 1799–1808. Proceedings of Machine Learning Research, 2017.

[41] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: Bandit-based configuration evaluation for Hyperparameter Optimization. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*, 2017. Published online: `iclr.cc`.

[42] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter. Fast bayesian Hyperparameter Optimization on large datasets. *Electronic Journal of Statistics*, 11:4945–4968, 2017.

[43] S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and efficient Hyperparameter Optimization at scale. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, volume 80, pages 1437–1446. Proceedings of Machine Learning Research, 2018.

[44] L. Li, K. G. Jamieson, A. Rostamizadeh, E. Gonina, J. Ben-tzur, M. Hardt, B. Recht, and A. Talwalkar. A system for massively parallel hyperparameter tuning. In I. S. Dhillon, D. S. Papailiopoulos, and V. Sze, editors, *Proceedings of Machine Learning and Systems 2020 (MLSys'20)*. mlsys.org, 2020.

[45] N. Awad, N. Mallik, and F. Hutter. DEHB: Evolutionary hyberband for scalable, robust and efficient Hyperparameter Optimization. In Z. Zhou, editor, *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI'21)*, pages 2147–2153, 2021.

[46] T. Domhan, J. Springenberg, and F. Hutter. Speeding up automatic Hyperparameter Optimization of deep neural networks by extrapolation of learning curves. In Q. Yang and M. Wooldridge, editors, *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*, pages 3460–3468, 2015.

[47] B. Baker, O. Gupta, R. Raskar, and N. Naik. Practical neural network performance prediction for early stopping. *arXiv:1705.10823 [cs.LG]*, 2017.

[48] A. Chandrashekaran and I. Lane. Speeding up Hyper-parameter Optimization by Extrapolation of Learning Curves using Previous Builds. In M. Ceci, J. Hollmen, L. Todorovski, C. Vens, and S. Džeroski, editors, *Machine Learning and Knowledge Discovery in Databases (ECML/PKDD'17)*, volume 10534 of *Lecture Notes in Computer Science*, pages 477–492. Springer, 2017.

[49] A. Klein, S. Falkner, J. Springenberg, and F. Hutter. Learning curve prediction with Bayesian neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*, 2017. Published online: `iclr.cc`.

[50] F. Runge, D. Stoll, S. Falkner, and F. Hutter. Learning to Design RNA. In *Proceedings of the International Conference on Learning Representations (ICLR'19)*, 2019. Published online: `iclr.cc`.

[51] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[52] J. Gardner, G. Pleiss, K. Q. Weinberger, D. Bindel, and A. G. Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems (NeurIPS'18)*. Curran Associates, 2018.

[53] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, editors, *Proceedings of the 24th International Conference on Advances in Neural Information Processing Systems (NeurIPS'11)*, pages 2546–2554. Curran Associates, 2011.

[54] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter. SMAC3: A versatile bayesian optimization package for Hyperparameter Optimization. *Journal of Machine Learning Research (JMLR) – MLOSS*, 23(54):1–9, 2022.

[55] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica. Tune: A research platform for distributed model selection and training. *arXiv:1807.05118 [cs.LG]*, 2018.

A Related Work

RL training pipelines are complex and often brittle [6, 7, 10]. This makes RL difficult to use for novel applications. To mitigate this, automated reinforcement learning [AutoRL; 31] aims to alleviate a human practitioner from the tedious and error prone task of manually setting up the RL pipeline.

While there exist different approaches to automate the choice of algorithm, architecture [32] or even environment components [33], in this work we focus on hyperparameter optimization [HPO; 34, 35] for RL. There exist various approaches in the literature of HPO for RL [see, e.g., 36, 8, 37, 38, for a detailed survey we refer to Parker-Holder et al. [31]]. Due to the non-stationarity of RL training, in recent years, most applications of hyperparameter optimization for RL have focused on dynamically adapting hyperparameters throughout the run. For example, population based training [PBT; 11] and variants thereof [see, e.g., 39, 29] have found more wide-spread use in the community. This style of HPO uses a population of agents to optimize their hyperparameters while training. Parts of the population are used to explore different hyperparameter settings while the rest are kept to exploit the so far best performing configurations. While this has proven a successful HPO method, a drawback of population based methods is that they come with an increased compute cost due to needing to maintain a population of parallel agents. Thus, most extensions of PBT, such as PB2 [29], aim at reducing the required population size. Still, to guarantee sufficient exploration, larger populations might be required which makes such methods hard to use with small compute budgets.

In the field of automated machine learning [AutoML; 30], multi-fidelity optimization has gained popularity to reduce the cost of the optimization procedure. Such methods [see, e.g., 40–45] leverage lower fidelities, such as dataset subsets, lower number of epochs or low numbers of repetitions, to quickly explore the configuration space. For the special case of number of epochs as a fidelity, there also exists a rich literature on learning curve prediction [13, 46–49, 14]. Multi-fidelity optimization typically evaluates the most promising configurations on higher fidelities, including the full budget. This style of optimization has proven a cost-efficient way of doing HPO for many applications. Still, multi-fidelity optimization has been explored only little in the context of RL. We are only aware of three such works: Runge et al. [50] used a multi-fidelity optimizer to tune the hyperparameters of a PPO agent [21] that was tasked with learning to design RNA, allowing the so-tuned agent to substantially improve over the state of the art. Nguyen et al. [16] also modelled the training curves, providing a signal to guide the search. In the realm of model-based RL, it was shown that dynamic tuning methods such as PBT can produce well-performing policies but often fail to generate robust results whereas static multi-fidelity approaches produced much more stable configurations that might not result in as high final rewards [9]. Crucially, however, these previous studies did not evaluate how multi-fidelity and PBT style methods compare in the low budget regime, a setting that is more realistic for most research groups.

B RCGP Pseudocode

We show the pseudocode in Algorithm 1. We stress that we are concurrently training one agent for each hyperparameter configuration, but we advance only one training procedure at a time, using an intelligent selection mechanism based on Bayesian optimization. In the first stage, we fit a GP (line 3) using the aforementioned novel kernel function that combines hyperparameter configurations, budgets, and estimated rewards (Equations 4-7). Afterward, we select the next configuration with the highest estimated acquisition at the end of the convergence (line 4). Then we train the RL agent corresponding to the selected configuration for one more budget increment (e.g. continue training for $\Delta_b = 10^b$ more training steps) and measure the observed reward at the end of the next budget (lines 5-6). Note that line 5 defines the next budget for both new configurations ($\nexists \lambda^{\text{next}} : (\lambda^{\text{next}}, \cdot, \cdot) \in H$) as well as existing ones. We add the evaluation to the history (line 7) and continue the BO procedure until no budget is left (line 3).

```
Algorithm 1: Gray-Box HPO for RL
Input  : Search space $\Lambda$, initial design $H^{(0)}$, budget increment $\Delta_b$, max budget per agent $b^{\max}$
Output: Best hyperparameter configuration $\lambda^*$
1 Evaluate initial configurations and budgets $H := H^{(0)}$ ;
2 while still budget do
3     Fit a GP on $H$ using Equations 4-7, for estimating $\mu(\lambda, b), \sigma^2(\lambda, b)$;
4     Run acquisition $a(\mu, \sigma)$ to select $\lambda^{\text{next}} := \arg\max_{\lambda \in \Lambda} a\left(\mu(\lambda, b^{\max}), \sigma^2(\lambda, b^{\max})\right)$;
5     Define the next budget until which to train the selected agent $\lambda^{\text{next}}$:
      $b^{\text{next}} := \min\left(b^{\max}, \begin{cases} \Delta_b & \nexists \lambda^{\text{next}} : (\lambda^{\text{next}}, \cdot, \cdot) \in H \\ \Delta_b + \max\limits_{(\lambda^{\text{next}}, b, \cdot) \in H} b, & \text{otherwise} \end{cases}\right)$ ;
6     Resume training agent with $\lambda^{\text{next}}$ until $b^{\text{next}}$ and measure reward $R\left(\lambda^{\text{next}}, b^{\text{next}}\right)$;
7     Append to history $H \leftarrow H \cup \{(\lambda^{\text{next}}, b^{\text{next}}, R\left(\lambda^{\text{next}}, b^{\text{next}}\right))\}$;
8 end
9 return Best configuration $\lambda^*$ with highest reward $\max\limits_{(\lambda^*, b, R(\lambda^*, b)) \in H} R\left(\lambda^*, b\right)$ ;
```

Table 1: Search spaces for HPO of PPO, A2C, DDPG, SAC, and TD3.

Algorithm	Hyperparameters	Hyperparameter Values
PPO	Learning rate ($\log_{10}$)	$\{-6, -5, -4, -3, -2, -1\}$
	γ	$\{0.8, 0.9, 0.95, 0.98, 0.99, 1.0\}$
	Clip	$\{0.2, 0.3, 0.4\}$
A2C	Learning rate ($\log_{10}$)	$\{-6, -5, -4, -3, -2, -1\}$
	γ	$\{0.8, 0.9, 0.95, 0.98, 0.99, 1.0\}$
DDPG	Learning rate ($\log_{10}$)	$\{-6, -5, -4, -3, -2, -1\}$
	γ	$\{0.8, 0.9, 0.95, 0.98, 0.99, 1.0\}$
	τ	$\{0.0001, 0.001, 0.005\}$
SAC	Learning rate ($\log_{10}$)	$\{-6, -5, -4, -3, -2, -1\}$
	γ	$\{0.8, 0.9, 0.95, 0.98, 0.99, 1.0\}$
	τ	$\{0.0001, 0.001, 0.005\}$
TD3	Learning rate ($\log_{10}$)	$\{-6, -5, -4, -3, -2, -1\}$
	γ	$\{0.8, 0.9, 0.95, 0.98, 0.99, 1.0\}$
	τ	$\{0.0001, 0.001, 0.005\}$

C Experimental Setup

C.1 Baselines

We focus on comparing the performance of RCGP to existing HPO approaches within a given time budget. We compare against three types of baselines. The first type includes standard baselines that do not utilize fidelity information during optimization, namely:

- **Random Search (RS)** [51] is a simple and common HPO baseline. It optimizes hyperparameters by selecting configurations uniformly at random.
- **Bayesian optimization with Gaussian Proccesses (GP)** [15] is another standard HPO baseline, using GPs as the surrogate model in standard blackbox Bayesian optimization. We used a GPytorch [52] implementation with a Matern $5/2$ kernel.

The second type of baselines consists of multi-fidelity baselines which exploit intermediate learning (a.k.a. reward) curve information. Concretely, we compare against the following multi-fidelity HPO techniques:

Table 2: List of environments for the experiments.

Environment Class	Environment Name	Action Space
Atari	Pong-v0 Alien-v0 BankHeist-v0 BeamRider-v0 Breakout-v0 Enduro-v0 Phoenix-v0 Seaquest-v0 SpaceInvaders-v0 Riverraid-v0 Tennis-v0 Skiing-v0 Boxing-v0 Bowling-v0 Asteroids-v0	Discrete
Classic Control	CartPole-v1 MountainCar-v0 Acrobot-v1	Discrete
	Pendulum-v0	Continuous
MuJoCo	Ant-v2 Hopper-v2 Humanoid-v2	Continuous

- **BOHB** [43] is a multi-fidelity HPO baseline that combines Bayesian optimization and Hyperband [41]. It uses tree-based Parzen estimators (TPE) [53] as a surrogate model for Bayesian optimization. We used the source code provided by the authors.
- **SMAC** [54] is a recent variant of BOHB that uses Random Forests (RF) as a surrogate model. Here again, we used the implementation provided by the authors.
- **DEHB** [45] is a state-of-the-art multi-fidelity HPO baseline that combines Differential Evolution and Hyperband. We used the source code released by the authors.

The third type of baselines includes online HPO techniques (which apply different hyperparameter configurations within a single RL agent training procedure). We compare against two state-of-the-art online HPO methods in RL, that are based on evolutionary search:

- **Population-Based Training (PBT)** [11] is an evolutionary HPO method that dynamically optimizes the hyperparameters during the run of the algorithm (i.e., RL agent training). It discards the worst-performing members of the population after a number of steps and replaces them with new hyperparameter configurations that are generated by perturbing the best-performing configuration. We used the PBT implementation in the *Ray Tune* library [55]. To facilitate a fair comparison on a small compute budget we follow the protocol of [29] and use a population of 4 individuals.
- **Population-based bandits (PB2)** [29] is a PBT-like dynamic HPO method. It replaces the random perturbation with a time-varying GP, as a mechanism to identify well-performing regions of the search space. Again, we used the implementation of PB2 in the *Ray Tune* library [55] with a population of 4 individuals.

D Additional Experimental Results

D.1 Ablating our design choices

Throughout the paper we based our technical novelty on the hypothesis that reward curves have predictable shapes, and as a result, we can model them accurately with generalized logistic functions (Equation 6). In this section, we ablate the effect of enriching the feature-space of our surrogate with the reward curve estimations, i.e. $[\lambda, b] \rightarrow [\lambda, b, \hat{R}(\lambda, b; w)]$. The ablations of Figure 2-3 demonstrate that using our novel reward curve modeling offers a major boost on the quality of the optimization. In addition, we ablate the effect of the max-smoothing transformation of the reward curves (Equation 1). The empirical results further demonstrate that smoothing the noisy reward curves improves the performance of RCGP in the low-budget regime, especially as the early segments of reward curves are very noisy.

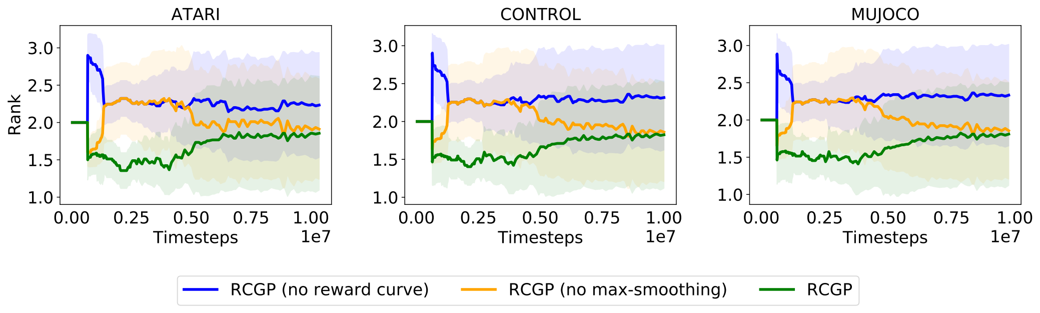

Figure 2: Rank comparison of RCGP (i) without reward curve model, (ii) with raw reward curve information, and (iii) with max-smoothing of the reward curve, for the **PPO** search space in the Atari, Classic Control, and Mujoco environments.

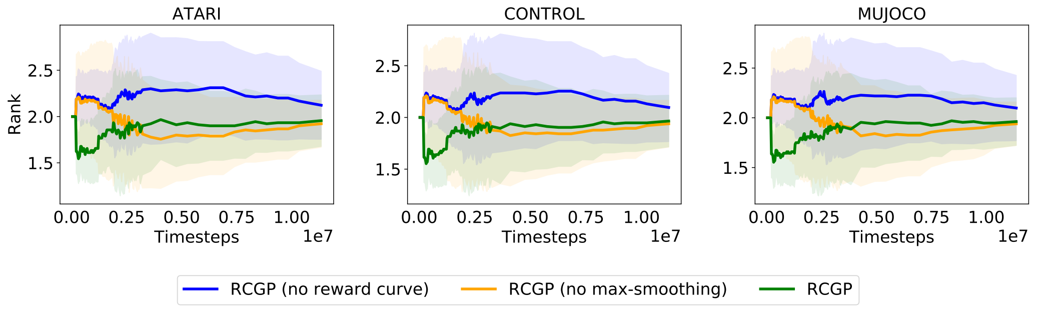

Figure 3: Rank comparison of RCGP (i) without reward curve model, (ii) with raw reward curve information, and (iii) with max-smoothing of the reward curve curve, for the **A2C** search space in the Atari, Classic Control, and Mujoco environments.

D.2 Comparison to Baselines

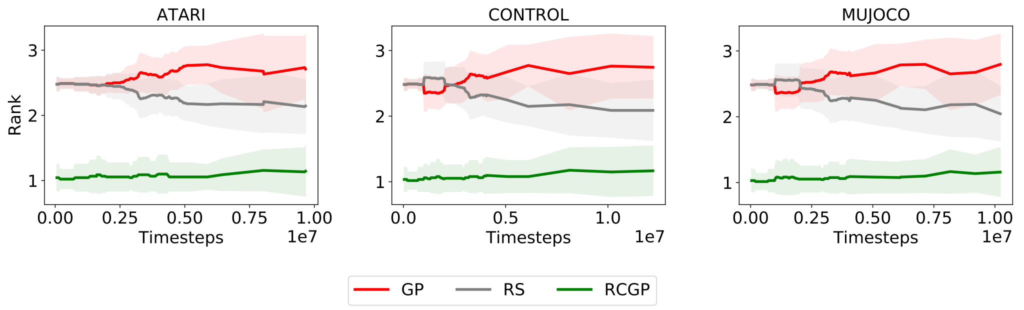

Figure 4: Rank comparison of RS, GP, and RCGP for the **PPO** search space in the Atari, Classic Control, and Mujoco environments.

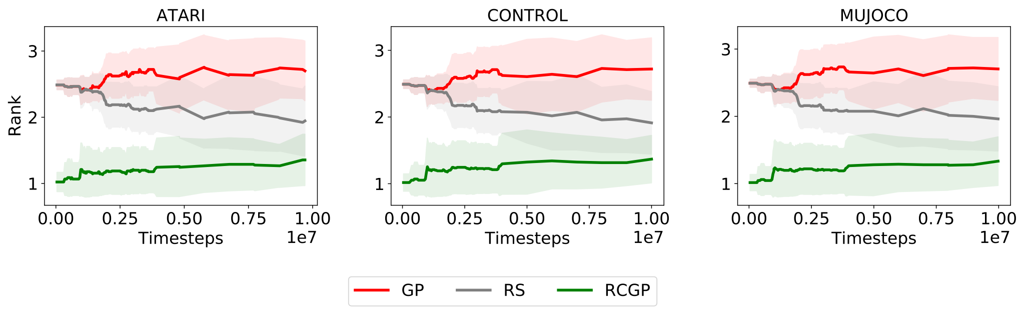

Figure 5: Rank comparison of RS, GP, and RCGP for the **A2C** search space in the Atari, Classic Control, and Mujoco environments.

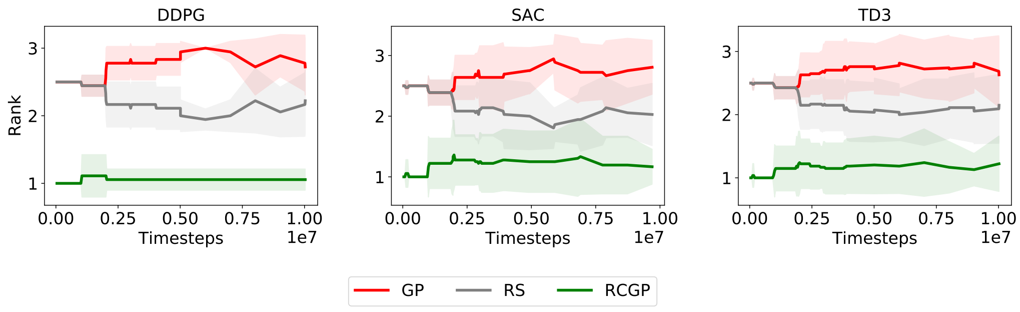

Figure 6: Rank comparison of RS, GP, and RCGP in the MuJoCo enviroments for the **DDPG, SAC, and TD3** search spaces.

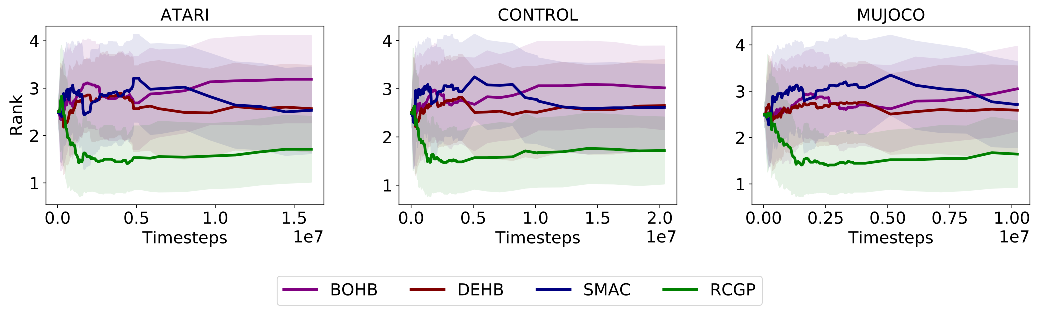

Figure 7: Ranks of BOHB, SMAC, DEHB, and RCGP in Atari, Classic control and MuJoCo enviroments for the **PPO** search space..

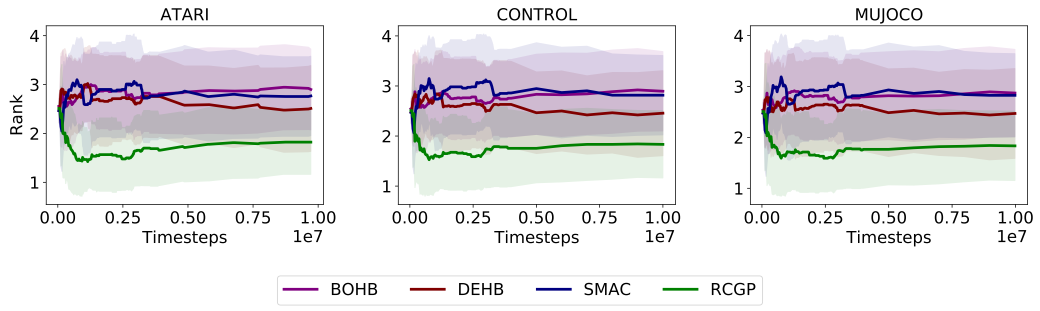

Figure 8: Ranks of BOHB, SMAC, DEHB, and RCGP in Atari, Classic control and MuJoCo enviroments for the **A2C** search space.

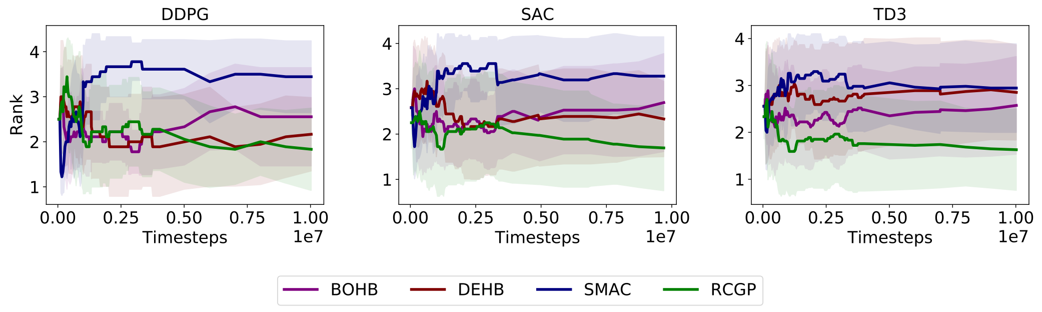

Figure 9: Ranks of BOHB, SMAC, DEHB, and RCGP in the MuJoCo enviroments for the **DDPG, SAC, and TD3** search space.

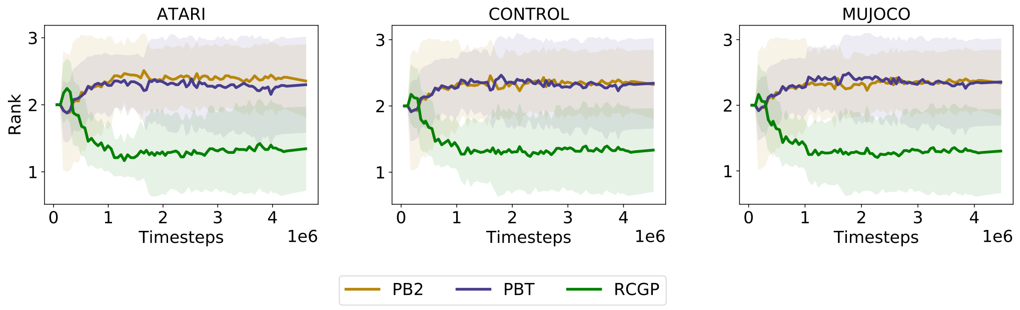

Figure 10: Ranks of PBT, PB2, and RCGP for the **PPO** search space in the Atari, Classic Control, and Mujoco environments.

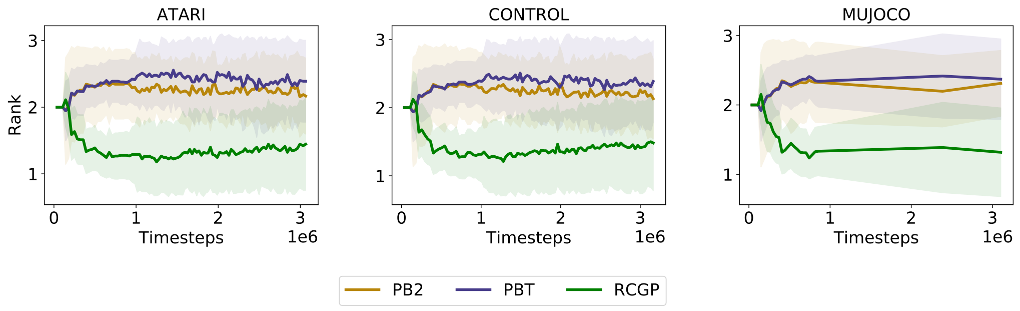

Figure 11: Ranks of PBT, PB2, and RCGP for the **A2C** search space in the Atari, Classic Control, and Mujoco environments.

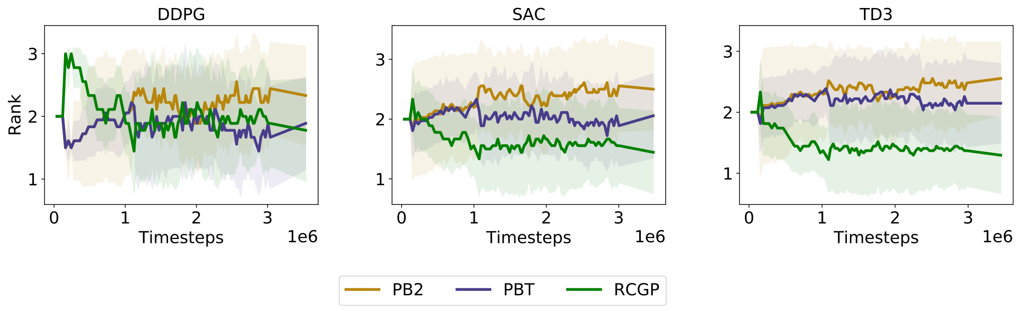

Figure 12: Ranks of PBT, PB2, and RCGP in the Mujoco enviroments for the **TD3** search space.