AutoRL-Bench 1.0

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

Abstract

It is well established that Reinforcement Learning (RL) is very brittle and sensitive to the choice of hyperparameters. This prevents RL methods from being usable out of the box. The field of automated RL (AutoRL) aims at automatically configuring the RL pipeline, to both make RL usable by a broader audience, as well as reveal its full potential. Still, there has been little progress towards this goal as new AutoRL methods often are evaluated with incompatible experimental protocols. Furthermore, the typically high cost of experimentation prevents a thorough and meaningful comparison of different AutoRL methods or established hyperparameter optimization (HPO) methods from the automated Machine Learning (AutoML) community. To alleviate these issues, we propose the first tabular AutoRL Benchmark for studying the hyperparameters of RL algorithms. We consider the hyperparameter search spaces of five well established RL methods (PPO, DDPG, A2C, SAC, TD3) across 22 environments for which we compute and provide the reward curves. This enables HPO methods to simply query our benchmark as a lookup table, instead of actually training agents. Thus, our benchmark offers a testbed for very fast, fair, and reproducible experimental protocols for comparing future black-box, gray-box, and online HPO methods for RL.

1 Introduction

Reinforcement Learning (RL) applications have made headlines in the past decade, with breakthroughs in a variety of domains such as game playing [see, e.g., 1–4], robotics [5] or real world tasks [6, 7]. These demonstrations of the capabilities of RL algorithms have caused a surge of interest. In spite of such achievements, RL algorithms are still highly sensitive to hyperparameter configurations and implementation details [8–11]. Additionally, the brittleness of RL [12], i.e. configurations drastically failing on some seeds while performing very well on others, results in unreliable comparisons of RL algorithms [13]. Bundled together with a typically high cost of experiments, this makes manual tuning of RL agents highly error-prone, tedious and requires vast expert knowledge.

The quickly growing field of automated reinforcement learning [AutoRL; 14] aims to tackle these problems. Still, efficient hyperparameter optimization of RL algorithms remains an open problem. We argue that this can be largely attributed to the fact that there does not yet exist a commonly used set of benchmarks on which novel AutoRL methods can be easily, fairly and comprehensively studied and compared. Without such benchmarks, it is difficult to develop novel HPO methods for RL, and it is even more challenging to compare fairly to existing HPO methods. Evaluating each hyperparameter configuration is computationally demanding, as each evaluation requires running an RL algorithm's training loop for a sequence of episodes, repeating the process for multiple seeds.

The field of neural architecture search (NAS) in fact had faced very similar issues before the first tabular NAS benchmark NAS-Bench-101 [15] was released. That benchmark had enormous impact in enabling reproducible research and democratizing the field, leading to more than 1 000 new papers on NAS in the last two years alone [16]. Here, we aim at introducing the equivalent of NAS-Bench-101 for the case of AutoRL. Without such benchmarks, while there has been significant success in other areas of Machine Learning (ML) for optimizing the hyperparameters of ML algorithms [17], the RL community has not yet seen many success stories regarding the adoption of HPO methods, besides notable exceptions such as PBT [18] and PB2 [19].[1] In order to facilitate the evaluation of existing and novel HPO methods for RL, we propose **AutoRL-Bench 1.0**, a tabular benchmark for hyperparameter optimization of RL.

AutoRL-Bench 1.0 is the first tabular HPO benchmark for RL and contains reward curves for five popular model-free RL algorithms across 22 environments. We consider three distinct classes of environments from OpenAI Gym [20]: Atari [21], Classic Control, and MuJoCo [22] and focus on five popular RL algorithms: PPO [23], A2C [24], DDPG [25], SAC [26], and TD3 [27]. We evaluate 468 distinct hyperparameter configurations for each environment, resulting in $12\,744$ total training runs. In order to allow for the fast evaluation of black-box, as well as gray-box HPO methods for RL, we provide reward curve information on the performance of different hyperparameter configurations across the environments. Speaking simply, a researcher can now evaluate a new HPO method by just querying our pre-computed reward curves without actually running any RL algorithms (i.e. the evaluation of hyperparameter configuration reduces to $\mathcal{O}(1)$). Further, as dynamic adaptation plays an important role in deep RL [14], for a subset of algorithms and environments, we evaluate 729 schedules of hyperparameters with distinct switching points on each considered environment, which allows us to give a head-to-head comparison between online/dynamic and gray-box HPO methods.

Our contributions are as follows: i) To the best of our knowledge, we are the first to provide a tabular HPO benchmark for RL algorithms. ii) AutoRL-Bench 1.0 is abundant in information, providing evaluations across 22 environments, as well as learning curve information on the performance of RL algorithms. iii) We additionally provide a version of the benchmark that considers hyperparameter schedules with distinct switching points that allow studying dynamic hyperparameter optimization. iv) We empirically demonstrate the usefulness of our benchmark by evaluating multiple black-box, gray-box and online HPO methods on it and providing insights into hyperparameter importance for RL algorithms.

2 Related Work

Tabular Benchmarks Tabular benchmarks have been proposed for other important problems in the (Auto)ML community where evaluations are expensive. For example, in the field of neural architecture search [NAS; 28] evaluating the performance of deep learning architectures can quickly become very resource intensive. Thus, to make novel NAS methods easily and cheaply comparable and reproducible, tabular benchmarks have become an important tool for NAS research [29]. By now, there exist various different tabular benchmarks (see, e.g., NAS-Bench-101 [15], NAS-Bench-1Shot1 [30], NAS-Bench-201 [31], NAS-Bench-301 [32] or NAS-Bench-Suite [33]) all of which provide different search spaces and target applications. The availability of such benchmarks has allowed rapid development of NAS methods. Similarly, tabular benchmarks play important roles in the fields of HPO [34] and multi-fidelity optimization [35, 36] (see, e.g., HPO-B [37] and HPOBench [38]). While RL is similarly or potentially even more expensive to train and evaluate, to the best of our knowledge, there have not yet been any tabular benchmarks on which HPO for RL could be studied.

Automated Reinforcement Learning (AutoRL) While there are various parts of the RL pipeline that could be automated, e.g., the choice of algorithm, architecture or environment components, in this work we focus on hyperparameter optimization (HPO) for RL[2]. One of the best understood and studied hyperparameter of RL is the discounting factor γ. For example, it is known that smaller values of γ lead to faster convergence but might result in myopic policies [39]. Increasing the discounting value over time can drastically speed up learning [40]. Further, François-Lavet et al. [40] showed that simultaneously decreasing the learning rate while increasing γ improves learning speeds even further. Still, for most algorithms and their hyperparameters it is not clear or understood whether they are best adapted during training or whether they should stay fixed [14] and how they influence the learning dynamics in general. Thus, it is common place that RL practitioners use some default configuration without exploring different types of HPO or AutoRL methods. To alleviate users from having to manually tune their RL agent, various HPO methods for RL have been proposed (see, e.g.,

[1]For a more in-depth discussion of related work we refer to Section 2

[2]For a comprehensive survey on AutoRL we refer to [14]

[18, 41, 42, 19, 43–45]). Still, HPO and AutoRL methods have not yet found widespread adoption by the RL community as methods are often not extensively evaluated.

Reproducibility of RL Comparison and reproducibility of RL experiments remain difficult to tackle problems. RL algorithms' sensitivity to hyperparameters [8, 12] is exacerbated by implementation details that can strongly influence an RL agents performance [9–11]. Further, the cost of typical RL experiments causes studies to often only compare performances on a handful of trials, which is most often not sufficient for a clear comparison [13].

These issues persist when trying to compare AutoRL methods [14]. Thus, novel AutoRL methods are often only evaluated on particular search spaces, across a handful of environments and not compared to existing baselines. Access to cheap-to-evaluate tabular benchmarks can mitigate this issue and may thus help advance the state-of-the-art in AutoRL and help shed light on when hyperparameters are best (not) adapted dynamically.

Benchmarks for RL There exists a plethora of benchmarks to evaluate RL algorithms [see, e.g., 22, 21, 20, 46]. However, these are designed with RL in mind, not HPO for RL or other forms of AutoRL. These benchmarks provide environments for the agents to interact with and to collect training examples on. This makes them prohibitive for use in AutoRL experiments as any RL agent that is being optimized will still have to compute expensive training updates. Our proposed benchmark differs from RL benchmarks in that it provides precomputed reward curves of already trained agents for specific hyperparameter configurations, and is tailored towards AutoRL research.

3 Benchmark Description

The benchmark comprises recorded reward-curves for five commonly used RL algorithms (PPO [23], A2C [24] and DDPG [25]) on 22 environments (see Appendix A Figure 2). For each algorithm, we consider the static configuration space containing the learning rate and discounting factor, and, depending on the algorithm, the clipping value or target network updating frequency (see Appendix A, top half of Table 1). For PPO, SAC, and TD3, we additionally consider a dynamic version in which hyperparameters can change at discrete time-steps *while* the agent is training. Our AutoRL-Bench 1.0 contains the recorded reward-curves for all training runs of each agent with all combinations of hyperparameters in the chosen configuration spaces.

The considered OpenAI Gym [20] environments consist of 15 Atari [21] games, four classic control problems, and three MuJoCo [22] tasks. Most environments have a discrete action space. Only the classic control task *Pendulum* and the MuJoCo environments *Ant*, *Hopper* and *Humanoid* have continuous action spaces. All 15 Atari games have image-based state representations whereas the classic control and MuJoCo environments have vector-based state representations. Most environments have dense reward signals, only the games *Bowling*, *Enduro*, *Pong*, *Skiing* and *Tennis* have mostly sparse reward signals. Finally, all environments were used to compute reward curves with static configurations. Only the classic control tasks and the Enduro game were used to compute reward curves for configuration schedules.

Data Collection For the RL algorithms, we used the implementations from **stable-baselines3** [47]. To be able to provide uncertainty estimates, we ran each configuration (or schedule) for three seeds on a compute cluster using rtx2080 GPUs. We trained all agents for 10^6 steps on each environment and evaluated the performance for 10 episodes every 10^4 steps. The total cost of creating the benchmark amounts to 29 160 GPU hours, or 3.3 GPU years of computational resources. For implementation details about the data format we refer to Appendix B.

When training PPO, SAC, and TD3 with configuration schedules, to avoid a combinatorial explosion, we limited the configuration space to two hyperparameters with three values each and used two discrete switching points after $3 \cdot 10^5$ and $6 \cdot 10^5$ training steps elapsed. This gives rise to a configuration space of $(3^2)^3 = 729$ distinct configuration schedules, all of which were evaluated on five environments for three seeds each. Without this limitation the original PPO configuration space with two switches would already have required $(6 \cdot 6 \cdot 3)^3 = 1\,259\,712$ evaluations. In the pruned space, a third switch would already result in $(3^2)^4 = 6\,561$ schedules.

```python
from benchmark_handler import BenchmarkHandler
import numpy as np
benchmark = BenchmarkHandler(data_path, environment="Pendulum-v0", seed=0,
                             search_space="PPO", static=True)
incumbent_reward, incumbent_config = -np.inf, None
for i in range(100):  # 100 iterations of random search
    candidate = np.random.randint(low=0, high=len(config_space))
    lr, gamma, clip = config_space[candidate]
    config_to_query = {"lr":lr, "gamma": gamma, "clip": clip}
    queried_data = benchmark.get_metrics(config_to_query, budget=100)
    result = queried_data["eval_avg_returns"][-1]
    if result > incumbent_reward:
        incumbent_reward, incumbent_config = result, config_to_query
```

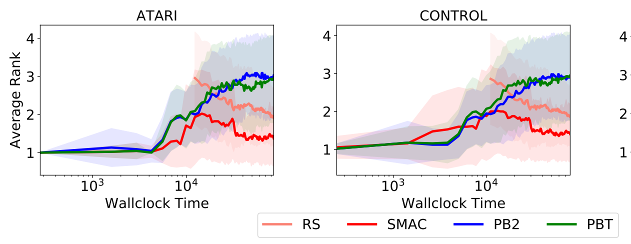
Listing 1: Code snippet for querying AutoRL-Bench 1.0 while optimizing with Random Search

API for AutoRL-Bench 1.0 To ease the accessibility to the data for users, we provide an API which is freely accessible at `https://github.com/releaunifreiburg/AutoRL-Bench`. Once the data is downloaded, a few lines of code suffice to query the metrics of a hyperparameter configuration for a given (environment, search space) combination. An example of querying the benchmark while optimizing with random search is given in Listing 1. The user can also switch between the dynamic or static spaces by modifying the respective attribute in the benchmark object. When querying a dynamic configuration, the user must provide the list of hyperparmeter values that are used in the schedule, where switches are possible at 300k and 600k training steps (see Listing 2 of Appendix B). In our GitHub repository, we provide further examples on advanced ways to query the API to avoid creating an object for every (environment,search space) combination. Moreover, we provide examples on how to couple its functionality with HPO optimizers.

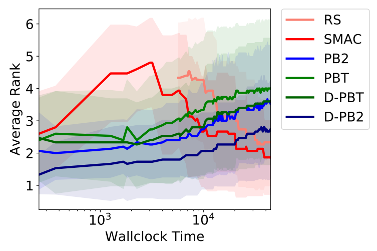
4 Experiments

Setup To demonstrate how HPO methods for RL or other AutoRL approaches could leverage our novel benchmark, here we provide a comprehensive comparison of existing HPO methods and evaluate their usefulness for RL. For a detailed description of the considered baselines we refer to Appendix C. On the static benchmark, we evaluated RS and SMAC4MF for a budget of 4 full RL training procedures for each (algorithm, environment, seed) triple. As PBT and PB2 are designed to optimize hyperparameters dynamically, we only evaluate them on the dynamic version of our benchmark. We denote these results under the labels D-PBT, and D-PB2. Additionally, to get a better understanding of the configuration spaces and to facilitate a comparison between multi-fidelity and dynamic methods, we compare to PBT and PB2 results for real runs on the full configuration spaces. These results are denoted with the labels PBT and PB2 respectively. Following Parker-Holder et al. [19] we used a population of 4 members for both PBT and PB2. For all baselines, we report the average rank of the evaluated methods across environments. For brevity, we group environments following the description of Section 3. We present group-aggregated results for PPO in the main paper; qualitatively similar results for all algorithms and additional results on a per-environment basis are given in Appendix D.

Results In all configuration spaces, we observe that the online HPO methods perform best in the beginning, but random search (RS) and SMAC4MF find static hyperparameter configurations that outperform the schedules optimized by PBT and PB2 (see Figure 1 and Figures 5 and 6in the appendix). In conclusion, PBT and PB2 with 4 workers are efficient in terms of discovering configurations under limited budgets, however, the SMAC4MF outperforms both on the PPO search space, when more than about 3 hours of HPO time per environment is available. Such findings indicate that the community needs novel HPO methods that both converge quickly given a low HPO budget, but also remain competitive compared to RS when more computing time is available. Moreover, the results indicate that SMAC4MF, the only multi-fidelity HPO baseline in our collection, is amongst the best-performing methods across time steps, but it performs worse than PBT and PB2 in the early HPO phases. Thus, we believe the successive halving mechanism is not optimally suitable for the realm of RL because reward curves are noisy. Figure 1 (subplot b) shows that running PBT and PB2 on our



(a) static



(b) dynamic

Figure 1: Average ranks for optimizing PPO on the (a) static and (b) dynamic benchmark

benchmark yields very similar results to running the default PBT/PB2 implementations, therefore validating the correctness of the dynamic benchmark. Overall, the experimental results demonstrated that tuning the hyperparameters of RL algorithms is an open challenge and we believe this benchmark and its eventual successors will be the *de facto* experimental protocol for innovating more efficient HPO methods for RL, which would enable practitioners to deploy RL in an off-the-shelve manner on new environments.

Finally, we used our benchmark to evaluate hyperparameter importance for the considered algorithms. Our results show that the learning rate is of high importance for the considered algorithms. For a more detailed analysis we refer to Appendix D.

5 Conclusion

We presented the first tabular HPO benchmark for AutoRL. Our tabular benchmark drastically reduces the computational requirements for evaluating novel AutoRL methods and, in turn, dramatically lowers the barrier of entry into this field of study. Our benchmark consists of reward curves for five commonly used RL methods across a diverse set of 22 environments. In particular, counter to commonly provided tabular HPO benchmarks, our benchmarks allow to study configuration schedules through the use of distinct switching points. We demonstrated the value of our benchmark by using it to evaluate commonly used HPO methods both from the AutoML as well as the AutoRL community. Lastly, we showed how our benchmark can provide insights to RL practitioners about the influence of hyperparameters on an agent's performance. We believe that our benchmark opens up the doors for the study of novel AutoRL methods and will help advance the field in a similar fashion as tabular benchmarks helped advance research in neural architecture search.

Acknowledgements

This research was partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215. André Biedenkapp and Frank Hutter acknowledge Robert Bosch GmbH for financial support. André Biedeankpp and Frank Hutter 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 additionally 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. URL `http://dx.doi.org/10.1038/nature14236`.

[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] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. Pondé de Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang. Dota 2 with large scale deep reinforcement learning. *arXiv:1912.06680 [cs.LG]*, abs/1912.06680, 2019.

[4] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang. Solving rubik's cube with a robot hand. *arXiv:1910.07113 [cs.LG]*, 2019.

[5] 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.

[6] M. G. Bellemare, S. Candido, P. Samuel Castro, J. Gong, M. C. Machado, S. Moitra, S. S. Ponda, and Z. Wang. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836):77–82, 2020.

[7] 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.

[8] R. Islam, P. Henderson, M. Gomrokchi, and D. Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv:1708.04133 [cs.LG]*, 2017.

[9] 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`.

[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] S. Huang, R. F. J. Dossa, A. Raffin, A. Kanervisto, and W. Wang. The 37 implementation details of proximal policy optimization. In *ICLR Blog Track*, 2022. URL `https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/`. https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/.

[12] 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 Conference on Artificial Intelligence (AAAI'18)*. AAAI Press, 2018.

[13] R. Agarwal, M. Schwarzer, P. Samuel Castro, A. C. Courville, and M. G. Bellemare. Deep reinforcement learning at the edge of the statistical precipice. 2021.

[14] 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. doi: 10.1613/jair.1.13596.

[15] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pages 7105–7114. PMLR, 2019.

[16] Difan Deng and Marius Lindauer. Literature on Neural Architecture Search. `https://www.automl.org/automl/literature-on-neural-architecture-search/`, 2022. [Online; accessed 25-September-2022].

[17] M. Feurer and F. Hutter. Hyperparameter optimization. In F. Hutter, L. Kotthoff, and J. Vanschoren, editors, *Automated Machine Learning: Methods, Systems, Challenges*, chapter 1, pages 3–38. Springer, 2019. Available for free at http://automl.org/book.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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. IEEE, 2012.

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

[24] 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.

[25] 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`.

[26] 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.

[27] 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. Proceedings of Machine Learning Research, 2018.

[28] T. Elsken, J. Metzen, and F. Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

[29] M. Lindauer and F. Hutter. Best practices for scientific research on neural architecture search. *Journal of Machine Learning Research*, 21(243):1–18, 2020.

[30] A. Zela, J. Siems, and F. Hutter. NAS-Bench-1Shot1: Benchmarking and dissecting one-shot neural architecture search. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=SJx9ngStPH`.

[31] X. Dong and Y. Yang. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR'20)*, 2020. Published online: `iclr.cc`.

[32] J. Siems, L. Zimmer, A. Zela, J. Lukasik, M. Keuper, and F. Hutter. NAS-bench-301 and the case for surrogate benchmarks for neural architecture search. *arXiv:2008.09777*, 2020.

[33] Y. Mehta, C. White, A. Zela, A. Krishnakumar, G. Zabergja, S. Moradian, M. Safari, K. Yu, and F. Hutter. NAS-Bench-Suite: NAS evaluation is (now) surprisingly easy. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=0DLwqQLmqV`.

[34] 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.

[35] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter. Fast Bayesian optimization of machine learning hyperparameters on large datasets. In A. Singh and J. Zhu, editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 54. Proceedings of Machine Learning Research, 2017.

[36] K. Jamieson and A. Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In A. Gretton and C. Robert, editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 51. Proceedings of Machine Learning Research, 2016.

[37] S. Pineda-Arango, H. S. Jomaa, M. Wistuba, and J. Grabocka. HPO-B: A large-scale reproducible benchmark for black-box HPO based on openml. volume abs/2106.06257, 2021.

[38] K. Eggensperger, P. Müller, N. Mallik, M. Feurer, R. Sass, A. Klein, N. Awad, M. Lindauer, and F. Hutter. HPOBench: A collection of reproducible multi-fidelity benchmark problems for HPO. In J. Vanschoren, S. Yeung, and M. Xenochristou, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. Curran Associates, 2021.

[39] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic programming*, volume 3 of *Optimization and neural computation series*. Athena Scientific, 1996. ISBN 1886529108.

[40] V. François-Lavet, R. Fonteneau, and D. Ernst. How to discount deep reinforcement learning: Towards new dynamic strategies. *arXiv:1512.02011 [cs.LG]*, 2015.

[41] 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.

[42] 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.

[43] 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.

[44] J. Franke, G. Köhler, A. Biedenkapp, and F. Hutter. Sample-efficient automated deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2021.

[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. ijcai.org, 2021.

[46] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman. Leveraging procedural generation to benchmark reinforcement learning. *arXiv:1912.01588 [cs.LG]*, 2019.

[47] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22 (268):1–8, 2021. URL `http://jmlr.org/papers/v22/20-1364.html`.

[48] 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.

[49] 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.

[50] 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`.

[51] 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.

[52] F. Hutter, H. Hoos, and K. Leyton-Brown. An efficient approach for assessing hyperparameter importance. In E. Xing and T. Jebara, editors, *Proceedings of the 31th International Conference on Machine Learning, (ICML'14)*, pages 754–762. Omnipress, 2014.

[53] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. 2016.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Appendix F
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Appendix G
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments (e.g. for benchmarks)...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See Section 3. For convenience we add the URL here again: `https://github.com/releaunifreiburg/AutoRL-Bench`
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 3
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 3
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Sections 1 and 3
   (b) Did you mention the license of the assets? [Yes] See Appendix E
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Additional Benchmark Details

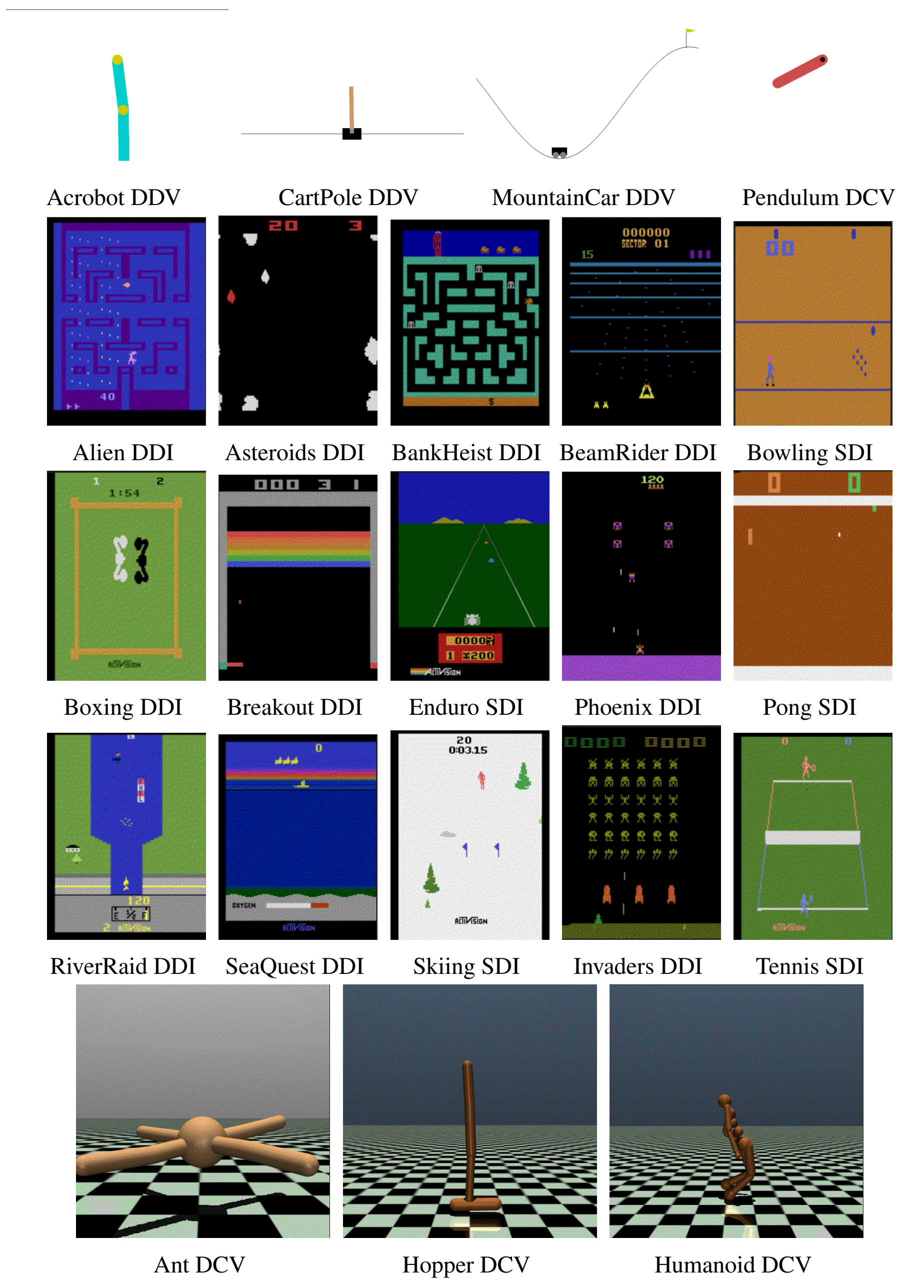


Figure 2: Used environments. We provide reward, action- and state-space classification. Rewards are either **D**ense or **S**parse. Action-spaces are either **D**iscrete or **C**ontinuous and state-spaces are either **I**mage based or **V**ector state-spaces. For example, Pong has a sparse reward (only scoring points carries reward information), a dense action space and an image based state-representation. For detailed descriptions of each environment we refer to `https://www.gymlibrary.ml`

Table 1: Configuration spaces of the considered RL algorithms

Type of Space	Algorithm	Hyperparameters	Hyperparameter Values
Static	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	$0.0001, 0.001, 0.005$
Dynamic	PPO	learning rate ($\log_{10}$)	$-5, -4, -3$
		γ	$0.95, 0.98, 0.99$
	SAC	learning rate ($\log_{10}$)	$-5, -4, -3$
		γ	$0.95, 0.98, 0.99$
	TD3	learning rate ($\log_{10}$)	$-5, -4, -3$
		γ	$0.95, 0.98, 0.99$

B Data Format and Implementation Details

We store the benchmark data as a set of JSON files, separated in folders per search space and environment. Every file contains the reward curves for a hyperparameter configuration (or schedule in the case of the dynamic search space) in a given environment and search space. Specifically, the JSON file has the following fields: i) *returns_train* – the reward list returned during training, ii) *timestamps_train* – the timestamp (in seconds) associated with the training reward, iii) *timesteps_train* – the time step associated with the reward, iv) *returns_eval* – the rewards observed during evaluation and its associated measurements, v) *std_returns_eval* – the standard deviation of the evaluation reward, vi) *timestamps_eval* – the timestamp associated with the evaluation reward, vii) *timesteps_eval* – the time step associated with the evaluation reward.

We use the following naming convention for all the files in the benchmark:
*%env_name%-%search_space%**_random_**%hp1%_val1%hp2%_val2_%**seed**%seedval%**eval.json***,
where we apply bold fonts for fixed strings. For instance, a filename is:
BeamRider-v0_A2C_random_lr_-6_gamma_0.95_seed0_eval.json.

We trained each configuration and seed tuple on an environment for 10^6 steps. Every 10^4 steps, we evaluated the agent for 10 episodes and recorded the mean and standard deviation of the obtained evaluation returns.

C Baseline Descriptions

Random Search (RS) is a simple and standard HPO baseline. It selects hyperparameter configurations uniformly at random.

SMAC4MF (SMAC) [48] implements a variant of the multi-fidelity optimizer BOHB [49] which combines Hyperband [50] with Bayesian optimization [BO; 51]. The Hyperband component allows to quickly discard under-performing configurations on smaller budgets (i.e., few epochs or number of training samples), whereas the BO component identifies well-performing regions of hyperparameters

```
from benchmark_handler import BenchmarkHandler

bench = BenchmarkHandler(data_path, environment="Pendulum-v0", seed = 0,
                         search_space="PPO", static=True)

#querying static configuration
config_to_query = {"lr":-6, "gamma": 0.8, "clip": 0.2}
queried_data = bench.get_metrics(config_to_query, budget=50)

#querying dynamic configuration
bench.static = False
config_to_query = {"lr":[-3,-4], "gamma": [0.8,0.99], "clip": [0.2,0.2]}
queried_data = bench.get_metrics(config_to_query, budget=50)
```

Listing 2: Code snippet for querying AutoRL-Bench 1.0

from which to sample. SMAC4MF differs from the original BOHB by fitting a Random Forest for the BO component.

Population Based Training (PBT) [18] is an evolutionary method for HPO that allows to dynamically change hyperparameters during training. PBT maintains a population of RL agents. Every N steps (a user-defined value) the worst members in the population are replaced with the best ones. Simultaneously the hyperparameters of these replaced agents are perturbed to explore if new hyperparameter values might improve the performance further. To work well, PBT typically requires large populations between 40 and 80 members.

Population based bandits (PB2) [19] extends the PBT framework and replaces the random hyperparameter perturbations with predictions from a time-varying Gaussian process. This change enables PB2 to perform a more informed search over hyperparameters. As a consequence, PB2 typically requires drastically fewer members (i.e., only 4 to 8) in the population compared to PBT.

D Additional Results

D.1 Hyperparameter Importance for RL Agents

In order to determine the importance of hyperparameters for the considered RL agents, we address the following questions. i) Which static hyperparameter configurations result in the best final reward for each considered algorithm? ii) Which hyperparameters are more important, i.e. have a higher influence on the final reward?

To answer the first question we compute the average rank of all considered static configurations per environment and for each configuration space separately. Based on these averages, we select the four best and worst configurations in each configuration space. Figure 3 depicts the results as box-plot, where lower ranks indicate better final rewards. Generally, lower learning rates result in better final rewards than configurations with high learning rates. Further, our results indicate that PPO and A2C configurations are less robust than DDPG ones, i.e. generally poorly-performing configurations can in some environments result in very good final rewards and vice versa. Contrary to practitioners' hopes, there exists no *silver bullet* hyper-parameter configuration that is clearly optimal in the vast majority of the environments. The take-home message is that optimal hyperparameters are environment-specific and must be carefully tuned.

To answer the second question we make use of the fANOVA hyperparameter importance method [52], which aims to quantify how strongly the change in a hyperparameters value influenced the final observed reward. fANOVA attributes higher importance to those hyperparameters that have a stronger influence in the final performance, i.e. reward. In Figure 4 we compute the average rank over all environments. Our results confirm that the learning rate is instrumental in achieving optimal performance for the considered algorithms. Furthermore, γ is very influential in the case of PPO.

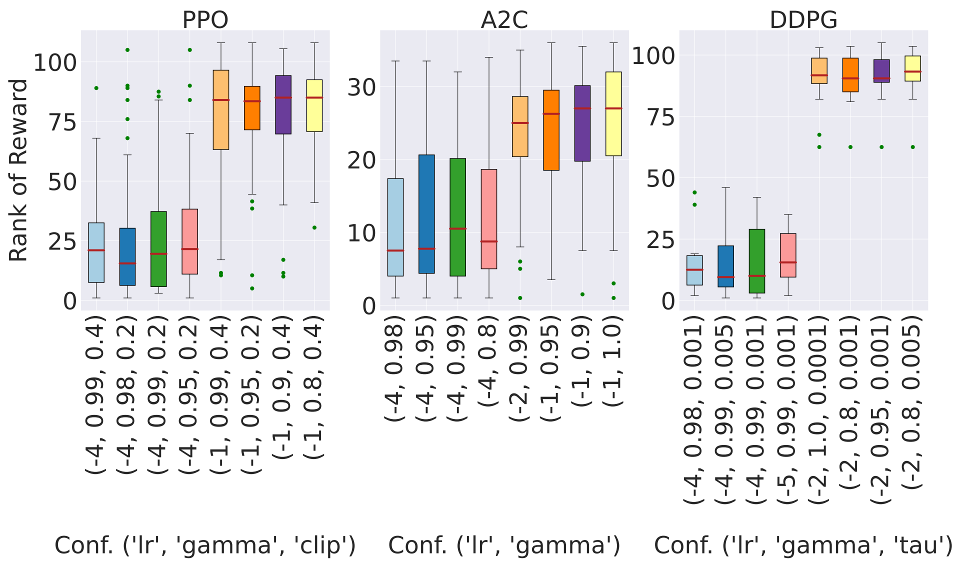


Figure 3: Average rank of the final reward across environments (four best and worst configurations)

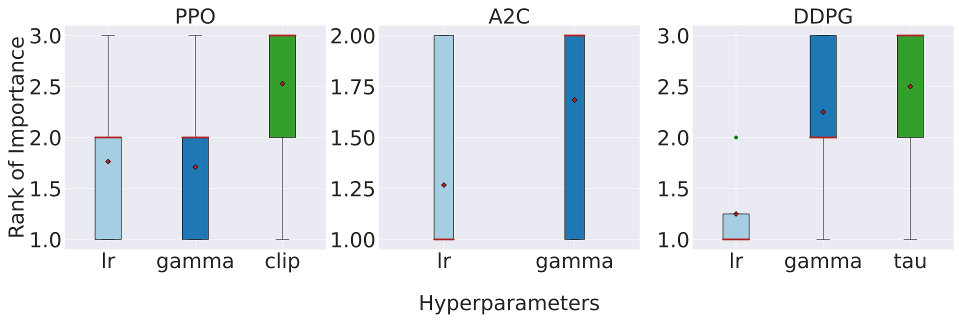


Figure 4: Hyperparameter importance per search space (red diamond=mean, red line=median)

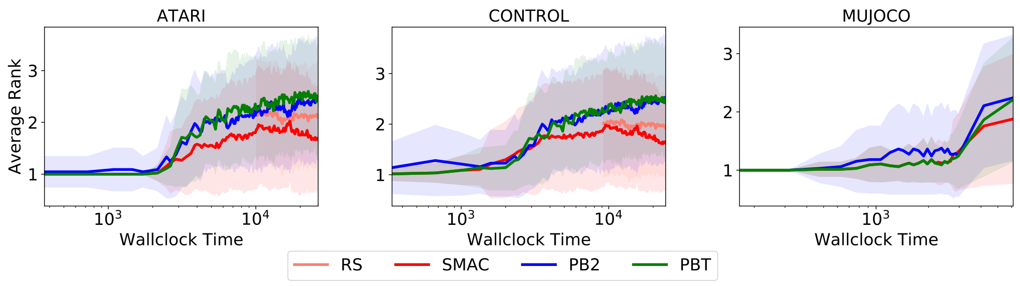


Figure 5: Average Rank for A2C Search Space

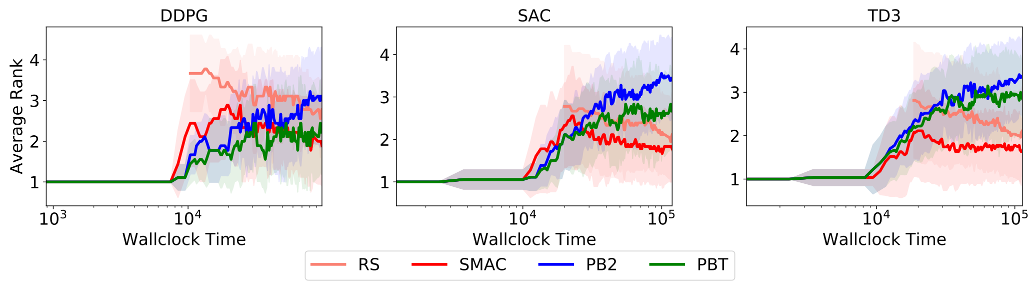


Figure 6: Average Rank for DDPG, SAC, and TD3 Search Space

D.2 Regret Plots

We present the regret plots of the experiments presented in the main paper in Figures 7 to 9.

We compute the normalized regret for an observed reward r within an environment as :

$$\text{normalized regret} = \frac{r_{\max} - r}{r_{\max} - r_{\min}} \tag{1}$$

where $r_{\max}$ and $r_{\min}$ are the maximum and minimum reward observed for any algorithm in that environment.

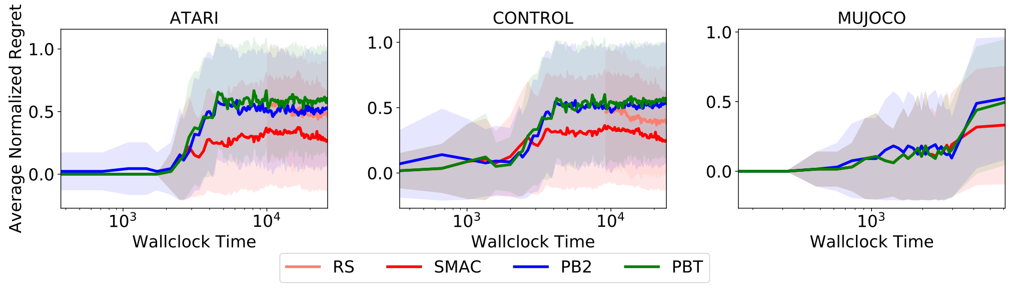
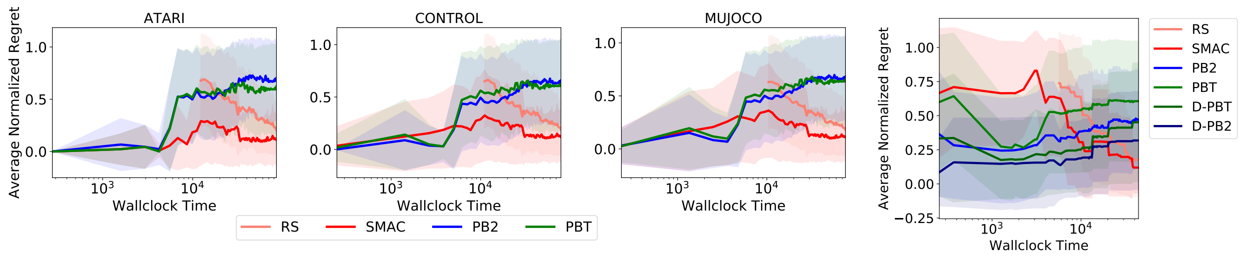


Figure 7: Average normalized regret for A2C Search Space

D.3 Rank plots per Environment

Figures 10 to 14 show the average rank of each evaluated HPO method on the individual environments.



(a) static

(b) dynamic

Figure 8: Average normalized regret for optimizing PPO on the (a) static and (b) dynamic benchmark

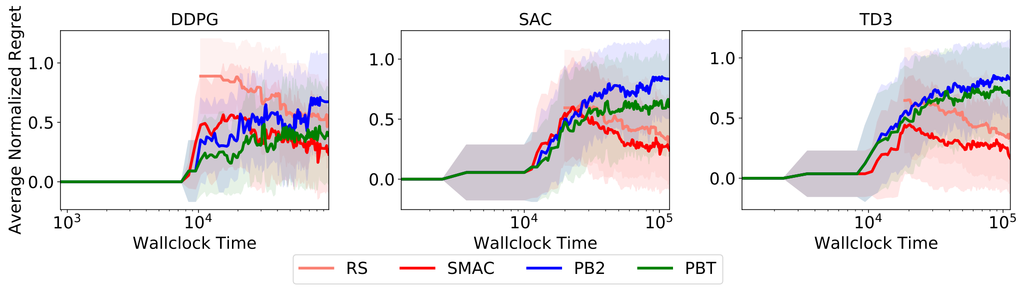


Figure 9: Average normalized regret for DDPG, SAC, and TD3 Search Space

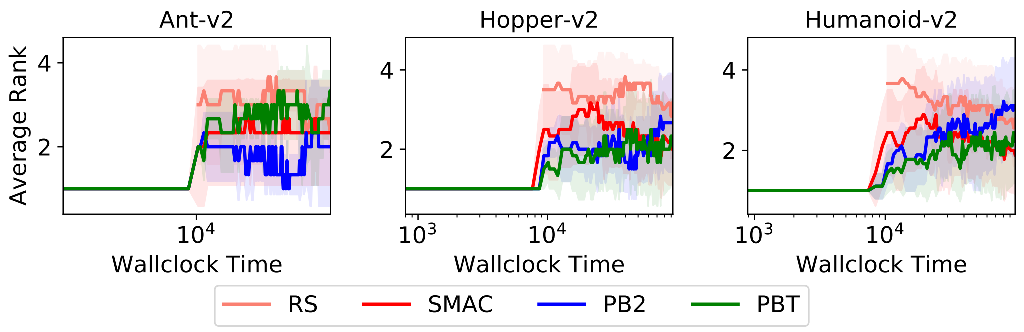


Figure 10: Average Rank for DDPG Search Space per Environment

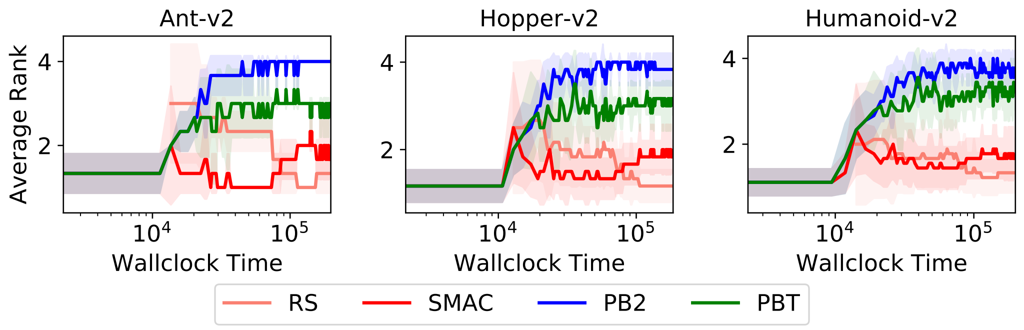


Figure 11: Average Rank for SAC Search Space per Environment

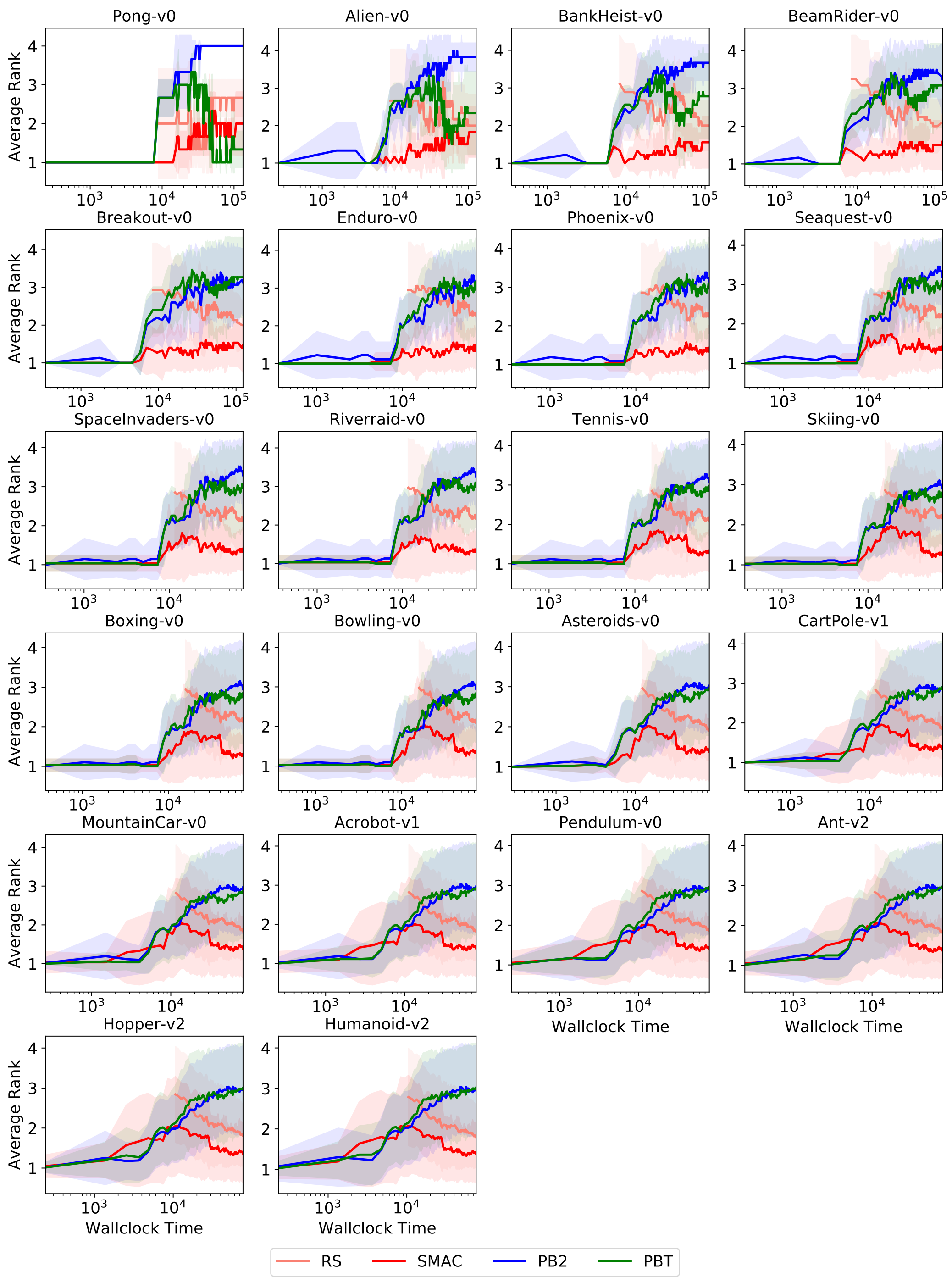


Figure 12: Average Rank for PPO Search Space per Environment

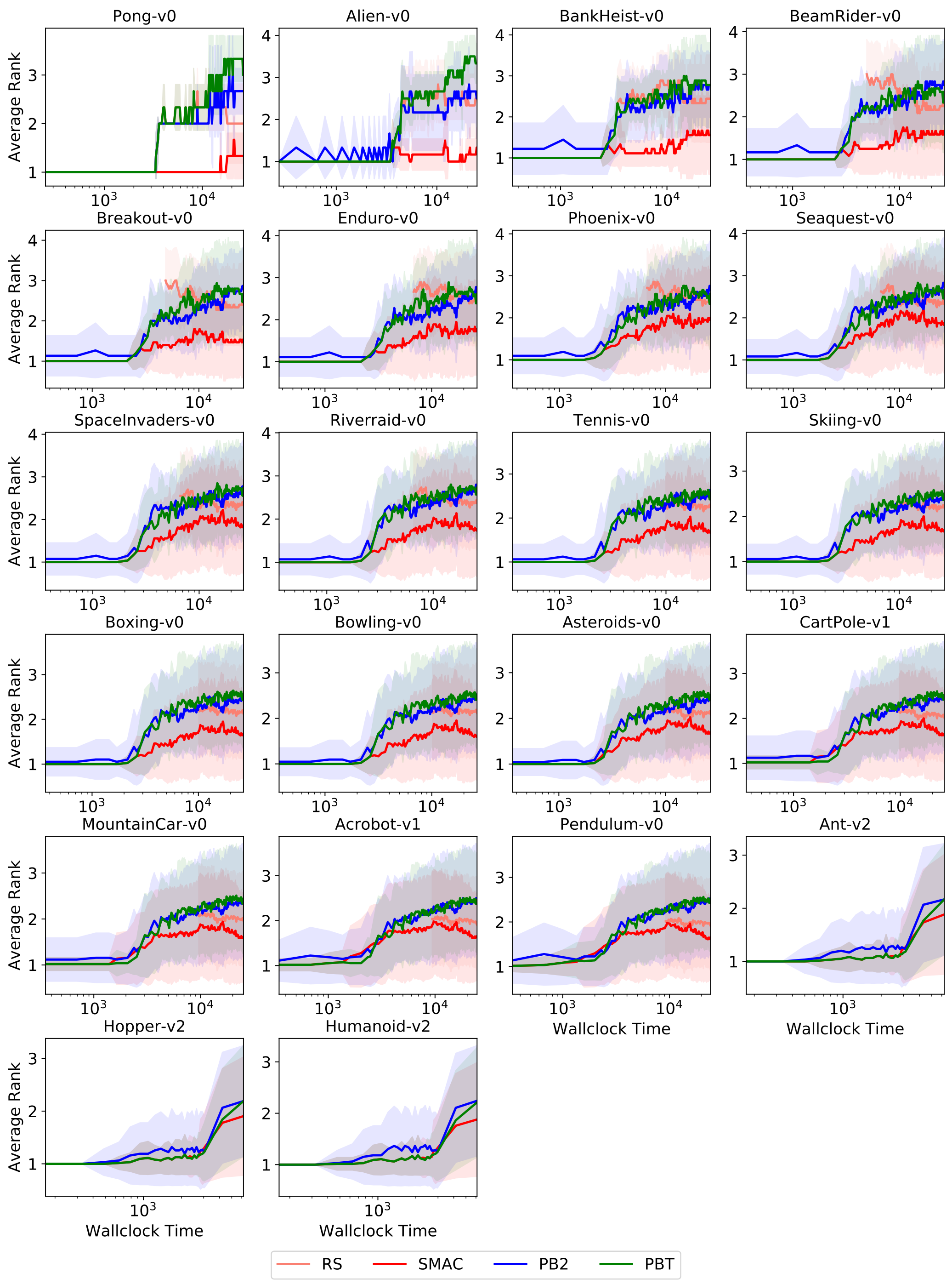


Figure 13: Average Rank for A2C Search Space per Environment

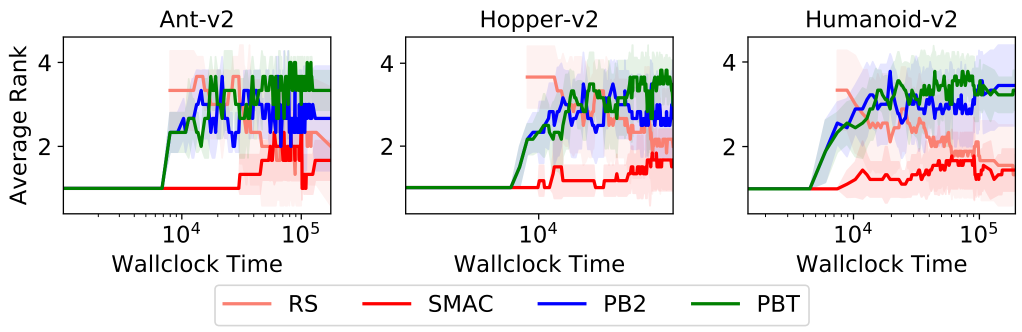


Figure 14: Average Rank for TD3 Search Space per Environment

E License

We provide Auto-RL Bench 1.0 under an MIT License. OpenAI Gym [53] and Stable-Baselines3 [47] are also offered under an MIT License.

F Limitations and Future Work

AutoRL-Bench 1.0 provides data that allows for the evaluation of black-box HPO, grey-box HPO, as well as online HPO methods. However, we only focus on model-free RL algorithms as our search spaces. This limitation can be lifted by extending the benchmark through increasing the number of search spaces. Further, by its tabular nature, AutoRL-Bench 1.0 covers exactly the evaluated configuration spaces but does not allow to reason about algorithmic behaviour outside the covered space. Still, AutoRL-Bench 1.0 lays the foundation for principled study of AutoRL and in particular HPO for RL. In future work, similar to trends in benchmarking for NAS [see, e.g., 33], we plan at to use surrogates models to cover larger configuration spaces while keeping the positive aspects of a tabular benchmark.

G Societal Impact

The provided benchmark serves the goal of environmental sustainability by, in the long run, reducing the energy consumption of training RL agents through the help of better AutoRL. Further, the benchmark is designed to facilitate research on AutoRL. As AutoRL aims at unlocking the full potential of RL by making RL algorithms off-the-shelf solution approaches to a variety of problem domains, there are various potential positive and negative impacts on society. By essentially democratizing RL this could enable users to employ RL even for nefarious use-cases. Still, the democratization of RL will ensure that not only a select few with vast amounts of compute are able to employ RL.