Computing and Artificial Intelligence 2025, 3(2), 2923.
https://doi.org/10.59400/cai2923

Article

Multifidelity Bayesian optimization for hyperparameter tuning of deep reinforcement learning algorithms

Eduardo C. Garrido-Merchán[1,*], Martin Molina[2], Gonzalo Martínez[2]

[1] Institute of Research in Technology (IIT), Universidad Pontificia Comillas, 28015 Madrid, Spain
[2] Department of Artificial Intelligence, Universidad Politécnica de Madrid, 28040 Madrid, Spain
*** Corresponding author:** Eduardo C. Garrido-Merchán, ecgarrido@comillas.edu

CITATION

Garrido-Merchán EC, Molina M, Martínez G. Multifidelity Bayesian optimization for hyperparameter tuning of deep reinforcement learning algorithms. Computing and Artificial Intelligence. 2025; 3(2): 2923. https://doi.org/10.59400/cai2923

ARTICLE INFO

Received: 10 March 2025
Accepted: 29 April 2025
Available online: 21 May 2025

Abstract: This research focuses on comparing standard Bayesian optimization and multifidelity Bayesian optimization in the hyperparameter search to improve the performance of reinforcement learning algorithms in environments such as OpenAI LunarLander and CartPole. The primary goal is to determine whether multifidelity Bayesian optimization provides significant improvements in solution quality compared to standard Bayesian optimization. To address this question, several Python implementations were developed, evaluating the solution quality using the mean of the total rewards obtained as the objective function. Various experiments were conducted for each environment and version using different seeds, ensuring that the results were not merely due to the inherent randomness of reinforcement learning algorithms. The results demonstrate that multifidelity Bayesian optimization outperforms standard Bayesian optimization in several key aspects. In the LunarLander environment, multifidelity optimization achieved better convergence and more stable performance, yielding a higher average reward compared to the standard version. In the CartPole environment, although both methods quickly reached the maximum reward, multifidelity did so with greater consistency and in less time. These findings highlight the ability of multifidelity optimization to optimize hyperparameters more efficiently, using fewer resources and less time while achieving superior performance.

Keywords: deep reinforcement learning; bayesian optimization; meta learning

1. Introduction

The field of reinforcement learning (RL) has experienced significant growth in recent years due to its ability to address a wide range of decision-making problems in complex and dynamic environments [1] like for example in the field of robotics where it is revolutionizing the field [2]. However, one of the key challenges in RL is the tuning of hyperparameters, which are parameters that control the behavior and performance of learning algorithms and that is an area that has been recently addressed by the new automatic reinforcement learning field [3]. Efficient optimization of these hyperparameters is crucial to achieving optimal RL algorithm performance within a reasonable time frame [4]. However, it is a really challenging field as every evaluation of a set of values of the hyperparameters usually requires lots of computational and time resources in real problems in fields such as robotics or finance [5].

In this context, Bayesian optimization class of methods has emerged as a promising technique for hyperparameter tuning in RL [6]. In particular, Bayesian optimization leverages the power of probabilistic models like Gaussian processes or Bayesian neural networks [7] to explore the search space in an efficient way and not by brute force as vanilla methods such as random search or grid search do. In contrast,

Bayesian optimization methods adapt the search as new evaluations of the configuration space, in this case consisting of the reinforcement learning hyperparameter values, are observed [8]. However, in computationally expensive settings, such as those involving complex simulations or high-dimensional RL models, standard Bayesian optimization can become prohibitively costly in terms of computational time. It is critical to consider here that deep reinforcement learning algorithms contain in their Bayesian space not only the hyperparameters of the reinforcement learning algorithms such as the gamma discount rate. It also happens the same with all the hyperparameters of deep neural networks like, for example, the number of layers of the network, the amount of neurons in each of the layers selected or the learning rate [9]. Consequently, in such a vast hyperparameter space, with very expensive evaluations of any of the hyperparameter values, it is a very challenging task to create algorithms that are to efficiently obtain satisfactory suggestions. However, it is necessary to adapt the deep reinforcement learning algorithms and models to any possible scenario.

To address this issue, multifidelity Bayesian optimization has been proposed as an extension that utilizes information from different levels of evaluation fidelity to accelerate the optimization process convergence [10]. This technique is particularly useful in scenarios where low-cost evaluations can be obtained, providing approximate information about the target function [11]. Recall that a fidelity is basically an approximation to the true underlying objective function. For example, high fidelity is an accurate approximation to the objective function whose correlation with it is high but its cost is also high. In contrast, a low fidelity is a low-quality approximation to the objective function whose correlation is low but, however, the cost of it is cheap. Intuitively, if we evaluate a low-fidelity configuration and its result is very bad, then it is not worth evaluating that configuration with a high fidelity, and we can just discard the neighborhood of hyperparameter values surrounding that bad-quality configuration. This is the intuition underlying the methodology of this paper, whose purpose is to make the hyperparameter tuning of deep reinforcement learning algorithms process cheaper. For the interested reader in the technical details of the Bayesian optimization class of methods, a full tutorial of the most efficient information-theoretic Bayesian optimization algorithms, which theoretically are the most complete ones, is useful to understand the history of this approach and why multifidelity approaches built on top of these algorithms are so useful in the scenario described in this introduction [12,13].

The primary objective of this manuscript is to evaluate the effectiveness of multifidelity Bayesian optimization in hyperparameter tuning for deep reinforcement learning algorithms, compared to just applying the standard Bayesian optimization algorithm, which is the default tool used for automating the deep reinforcement learning hyperparameter tuning process. RL has gained prominence for its ability to solve complex decision-making problems, yet the computational cost of hyperparameter tuning remains a significant bottleneck, particularly in deep reinforcement learning settings [1]. While standard Bayesian optimization offers an efficient search strategy [8], its application in computationally intensive environments can be limited. Multifidelity Bayesian optimization addresses this limitation by incorporating evaluations of varying fidelity levels, potentially reducing

computational demands while maintaining optimization quality [10]. This study aims to provide empirical insights into the trade-offs between these approaches, contributing to the growing body of research on efficient hyperparameter optimization in RL.

Having introduced the contextualization and motivation behind this manuscript, we can now state that this paper is structured as follows. Section 2 provides a concise review of RL fundamentals, introduces the DRL algorithm employed in the experiments, and discusses Bayesian optimization principles. It also surveys popular tools for hyperparameter optimization and highlights examples of Bayesian optimization applied to RL problems, grounding the work in existing literature. Section 3 details the experimental setup, including the RL environments, optimization algorithms, tuned hyperparameters, and other relevant design considerations. Section 4 presents and analyzes the experimental results, comparing the performance of different optimization techniques and discussing key observations. Finally, Section 5 concludes the study with a comprehensive analysis of the project's objectives, summarizing the main experimental findings and proposing directions for future research in this domain.

2. Fundamentals of the methodologies of reinforcement learning and Bayesian optimization

Reinforcement Learning (RL) [14] is a framework that operates within a Markov Decision Process (MDP) model, where an agent learns an optimal policy $\pi(a|s)$ to maximize the expected cumulative discounted reward $R\_t = \Sigma\ \gamma\text{^}t\ r\_t$, with $\gamma \in [0,1)$ as the discount factor, through interactions with an environment defined by state space S, action space A, transition dynamics $P(s'|s,a)$, and some notion of reward function $r(s,a)$ that is configurable with the user [1]. The critical advantage of the reinforcement setup is that the practitioner can configure the variables involved in the observation space, what the agent observes, in the action space, what the agent does, and the reward function, what the agent is going to learn in a simulator of the scenario where it is going to be deployed. The agent iteratively refines π by balancing exploration (sampling new actions) and exploitation (leveraging known rewards), typically via value-based methods like Q-learning, which updates $Q(s,a) = r + \gamma\ \max\_a'\ Q(s',a')$ using temporal difference learning, or policy gradient methods that directly optimize π by ascending $\nabla\_\theta\ J(\theta) = E[\nabla\_\theta \log \pi\_\theta(a|s)\ \hat{A}\_t]$, where $\hat{A}\_t$ is the advantage estimate. RL algorithms scale poorly to high-dimensional or continuous spaces, prompting the advent of deep reinforcement learning (DRL), which integrates deep neural networks (DNNs) as function approximators for Q-values or policies, enabling generalization across complex environments like Atari games or robotic control [4]. Among DRL methods, Proximal Policy Optimization (PPO) stands out for its stability and efficiency, employing a clipped surrogate objective $L\_CLIP(\theta) = E[\min(r\_t(\theta)\hat{A}\_t, \text{clip}(r\_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}\_t)]$, where $r\_t(\theta) = \pi\_\theta(a\_t|s\_t)/\pi\_\theta\_old(a\_t|s\_t)$ is the policy ratio, $\hat{A}\_t = R\_t - V(s\_t)$ approximates advantage via a value function V, and ε (typically 0.2) enforces a trust region, preventing destructive updates [15]. PPO's key strengths lie in its sample efficiency—reusing trajectories across multiple updates—and its robustness to hyperparameter settings, making it a preferred choice for tasks requiring

monotonic policy improvement, such as continuous control. However, PPO's performance hinges on tuning hyperparameters like learning rate, clipping threshold ε, and network architecture, which motivates advanced optimization strategies and this paper.

Bayesian Optimization (BO) addresses hyperparameter tuning by modeling the unknown objective function f(x)—here, some kind of function related to RL agent performance—as a probabilistic surrogate, typically a Gaussian Process (GP) with mean m(x) and covariance k(x,x'), updated via observed samples D{x_i, y} [8]. BO iteratively selects the next hyperparameter configuration x* by maximizing an acquisition function, such as Expected Improvement (EI), defined as EI(x) = E[max(f(x)−f(x_best), 0)], balancing exploration of uncertain regions and exploitation of promising ones. The posterior mean and variance of the GP guide this process, with kernel choice (e.g., Matérn or RBF) and length-scale priors influencing surrogate accuracy. While effective for low-dimensional, continuous spaces, standard BO struggles with the computational expense of DRL, where each evaluation involves training a model over thousands of timesteps, necessitating efficient alternatives. Hyperparameter optimization tools like SMAC (Sequential Model-based Algorithm Configuration) enhance BO by replacing GPs with random forests, better suited for mixed discrete-continuous spaces common in RL (e.g., categorical choices like optimizer type alongside continuous learning rates) [16]. SMAC iteratively builds a surrogate model, optimizes an acquisition function (e.g., EI), and evaluates configurations, leveraging local search and intensification to refine promising regions. SMAC3, an evolution of SMAC, introduces multi-fidelity optimization, exploiting low-cost approximations (e.g., training on fewer timesteps or smaller networks) to estimate f(x) at varying fidelity levels, then refining with high-fidelity evaluations as needed. This approach accelerates convergence by reducing the number of expensive evaluations, critical for DRL applications where computational budgets are constrained. In RL contexts, BO and its variants have been applied to tune PPO and other algorithms, optimizing metrics like cumulative reward or convergence speed, demonstrating significant performance gains over grid or random search in domains like robotics and game playing [8,15].

3. Experimental setup

This section outlines the experimental setup, detailing the design, specific problems addressed, hyperparameters targeted for optimization, implementation tools and strategies, and the default configuration of the Proximal Policy Optimization (PPO) algorithm. It describes the experimental structure and methodology, the selected problems with their objectives and challenges, the systematic approach to hyperparameter optimization for model performance enhancement.

Within reinforcement learning, the term "environment" denotes the simulated or physical domain where the learning agent operates, supplying state information, action outcomes, and rewards that reflect goal achievement, guiding the agent to maximize cumulative rewards over time. Two environments were selected for this study: CartPole, chosen for its simplicity and ability to demonstrate core RL concepts, and

LunarLander, selected for its increased complexity relative to CartPole, enabling assessment of RL algorithm performance and robustness in more challenging settings.

The CartPole environment, a classic reinforcement learning problem, simulates an inverted pendulum (pole) attached to a mobile cart, with the objective of balancing the pole upright by adjusting the cart's position to prevent it from falling. Its simplicity yet inherent challenge makes it a widely used benchmark for RL algorithm experimentation. The agent operates with two discrete actions—moving the cart left or right—and observes a state comprising the cart's position, cart velocity, pole angle, and pole angular velocity. The LunarLander environment simulates a spacecraft landing on the lunar surface, aiming to safely touch down in a designated zone while avoiding crashes and minimizing fuel consumption. Its increased complexity, with multiple actions and diverse states and rewards, makes it a compelling testbed for RL algorithms. The agent controls the spacecraft's engine orientation and thrust, selecting from four discrete actions: do nothing, fire left, fire right, or fire downward. It observes a state comprising the spacecraft's position, velocity, orientation angle, angular velocity, and surface contact data, enabling informed decision-making. Rewards are positive for successful landings, collision avoidance, and fuel efficiency, and negative for crashes, off-target landings, or excessive fuel use, providing feedback to guide the agent toward an effective control policy.

The hyperparameters targeted for optimization are the learning rate, discount factor, and GAE lambda, critical for agent performance in CartPole and LunarLander due to their direct influence on learning dynamics. The learning rate governs the step size of neural network weight updates during training, where overly high values may cause unstable convergence, and overly low values may lead to slow or suboptimal learning, making its optimization vital for effective training in CartPole. The discount factor weighs future versus immediate rewards, with higher values favoring long-term outcomes—key in CartPole's delayed-reward structure—while lower values risk short-sighted decisions, necessitating careful tuning to balance temporal considerations. Lambda, part of Generalized Advantage Estimation (GAE) in policy-based methods like Actor-Critic, reduces variance in advantage estimates, enhancing stability in PPO training. Optimization leverages SMAC (Sequential Model-based Algorithm Configuration) via a custom optimizer.py script utilizing the Hyperparameter Optimization Facade (HPOFacade) class, designed to boost PPO agent performance in CartPole and LunarLander simulations.

The experimental procedure comprises six steps: (1) preparing the CartPole and LunarLander test environments; (2) defining hyperparameter search spaces for each optimization algorithm; (3) executing HPO-SMAC and MF-SMAC for hyperparameter optimization in each environment; (4) training RL agents using the best hyperparameter configurations identified; (5) evaluating the trained agents' performance in the test environments; and (6) conducting a comparative analysis of results to assess the efficacy of the optimization algorithms.

4. Experimental results

This section presents the experimental results evaluating the performance of a Proximal Policy Optimization (PPO) agent in CartPole and LunarLander

environments. The reason why we have chosen these settings as benchmarks for the experiments of this paper is because they are very common in the deep reinforcement learning literature and also cheap and general settings to compare these algorithms and also algorithms used to optimize the hyperparameter values [17–19]. In this paper, we are comparing a baseline approach with fixed hyperparameters and constant computational resources against a multifidelity approach that varies training budgets across configurations, as it is common practice in multi-fidelity Bayesian optimization literature [20,21]. Results are visualized through plots of cumulative rewards during training and periodic evaluations, conducted at intervals defined by eval_freq (TOTAL_TIMESTEPS/100), ensuring assessments occur regularly, with outcomes reported at these discrete points, explaining the absence of decimal values in convergence data. The choice of cumulative rewards as a proxy for the quality of deep reinforcement learning algorithms is also found in literature and the reason why we have chosen such a quality measure [22]. Plots display episodes on the *X*-axis and cumulative rewards on the *Y*-axis, facilitating comparison of both approaches' performance. Results for both versions across the two environments are detailed, with hyperparameter configurations rounded to five decimals and comprehensive performance comparisons provided.

We now describe the details of the experiments using basic Bayesian optimization for a Proximal Policy Optimization (PPO) agent in the CartPole environment, a simple setting with two discrete actions and a small state space, enabling rapid convergence to the optimal reward of +500 (range: −500 to +500) within minutes. Due to its simplicity and low training demands, various hyperparameter configurations can achieve this maximum. In the baseline experiment, employing an early stopping criterion of 180 s and 10,000 training timesteps, the optimized hyperparameters—discount factor: 0.95599, GAE lambda: 0.96237, learning rate: 0.00033—yielded a post-evaluation reward of 500. The performance plot (**Figure 1**) illustrates the agent's training and evaluation, confirming swift attainment of the maximum reward, underscoring CartPole's suitability for achieving strong results across multiple hyperparameter settings with basic Bayesian optimization.

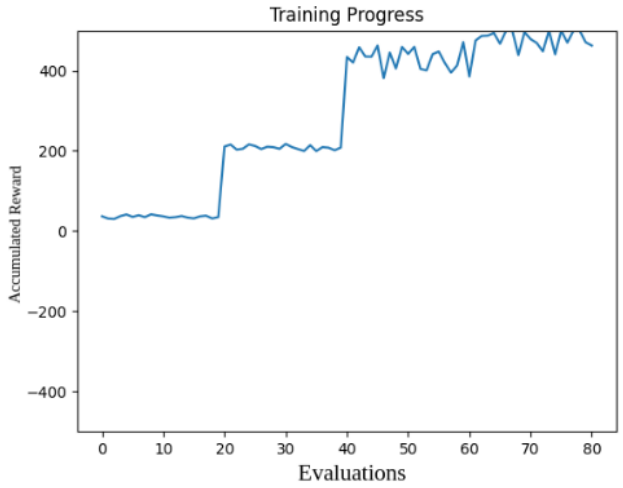

Figure 1. Agent performance in CartPole with the basic Bayesian optimization version.

The following experiment modifies the GenericSolver file to halt training once the reward consistently exceeds a predefined threshold of 350, with a maximum duration of 180 s and 10,000 timesteps, though rapid convergence often reduced the actual steps needed. The code was adjusted to minimize training time rather than reward, yielding hyperparameters—discount factor: 0.94213, GAE lambda: 0.93638, learning rate: 0.00026. The plot (**Figure 2**) depicts the agent's training and evaluation performance, showing early stopping at 6000 steps (evaluation 60 of 100) due to meeting the threshold, rather than completing all 10,000 steps. Results indicate that applying a reward threshold and early stopping accelerates convergence in CartPole, prioritizing efficiency over maximizing reward, achieving the target in less time. This approach proves valuable when training duration is a priority, demonstrating effective hyperparameter configurations that balance reward attainment with reduced computational effort.

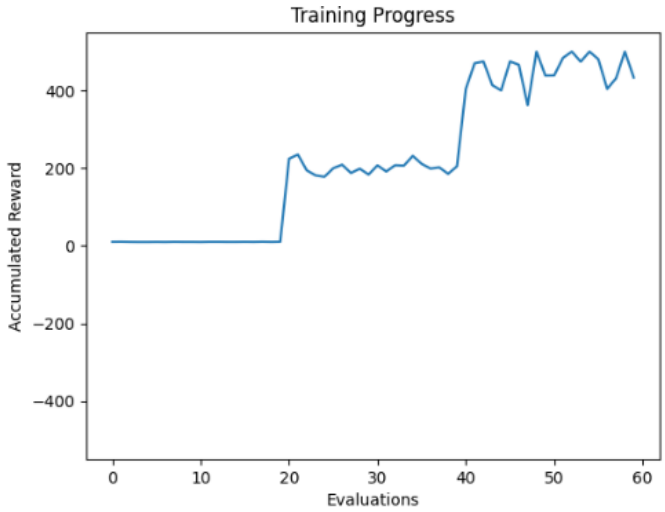

Figure 2. Convergency with the basic Bayesian optimization method.

We now examine multifidelity Bayesian optimization for a Proximal Policy Optimization (PPO) agent in CartPole, leveraging its simplicity and rapid convergence to explore advanced optimization strategies with limited costly evaluations. The experiment set a 180-s optimization time limit and adjusted training timesteps between 5000 and 10,000 based on budget constraints, yielding hyperparameters—discount factor: 0.95418, GAE lambda: 0.95975, learning rate: 0.00532. The performance plot (**Figure 3**) indicates that multifidelity optimization does not enhance hyperparameter search efficiency in CartPole compared to the basic approach. Given the environment's simplicity and the baseline's strong performance, optimal results are achieved quickly, diminishing the benefits of multifidelity techniques in this context.

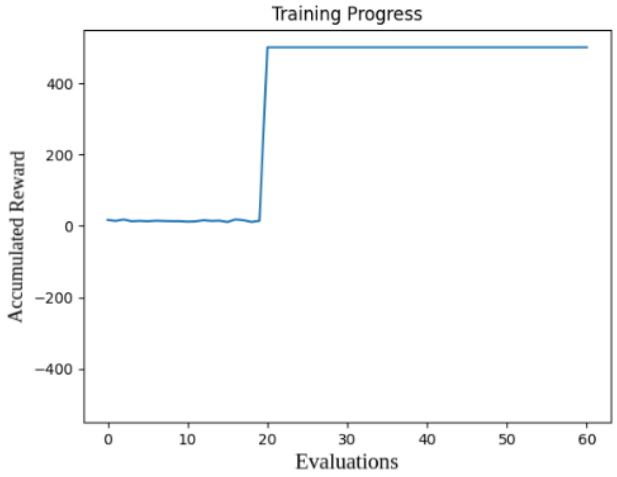

Figure 3. Agent performance in CartPole with the multi fidelity Bayesian optimization version.

However, for experiments with reduced timesteps (maximum 2500, minimum 2000) and an early stopping criterion of 180 s, multifidelity Bayesian optimization yielded the configuration—discount factor: 0.95418, GAE lambda: 0.95975, learning rate: 0.00532. We have found a 3% efficiency improvement over the non-multifidelity approach in CartPole, attributed to testing more configurations within the constrained timeframe. By conducting multiple evaluations with a limited training budget, multifidelity optimization identified hyperparameter settings that slightly enhanced agent performance, highlighting its advantage in resource-constrained scenarios despite the environment's simplicity.

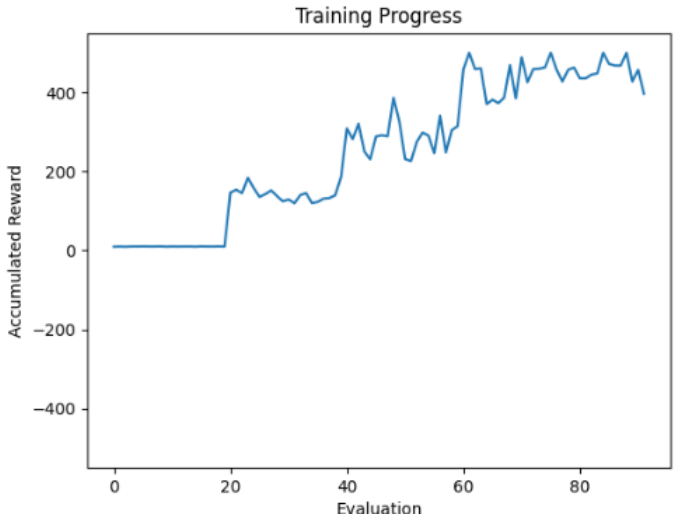

Figure 4. Convergency with the multi fidelity Bayesian optimization method.

The next experiment modified the GenericSolver to stop training once the reward consistently exceeded a threshold of 350, with a 180-s early stopping limit and training timesteps ranging from 5000 (minimum budget) to 10,000 (maximum budget), though rapid convergence often reduced steps needed. The code was adjusted to minimize training time rather than reward, yielding hyperparameters—discount factor: 0.95547, GAE lambda: 0.93638, learning rate: 0.00016. The plot (**Figure 4**) shows the agent's performance, converging at 8000 steps—20% fewer than the maximum—due to early

stopping. Results validate a high reward of 476, demonstrating that multifidelity Bayesian optimization with a reward threshold and time-focused optimization enhances efficiency in CartPole, achieving the target reward faster despite not maximizing it.

We now present the baseline experiment for basic Bayesian optimization in LunarLander, a significantly more complex environment than CartPole, requiring the agent to handle diverse states and actions, resulting in slower, computationally expensive learning. The optimization process was configured with a 3-h (10,800-s) duration and 1,000,000 timesteps to ensure sufficient training for evaluating configuration efficacy. The resulting hyperparameters—discount factor: 0.95547, GAE lambda: 0.93638, learning rate: 0.00016—achieved the best performance among eight tested configurations, yielding an average validation reward of 239. The plot (**Figure 5**) illustrates training and evaluation performance, confirming that, despite LunarLander's complexity, basic Bayesian optimization can identify effective hyperparameter settings, delivering robust results, albeit with substantially greater time and computational demands compared to CartPole.

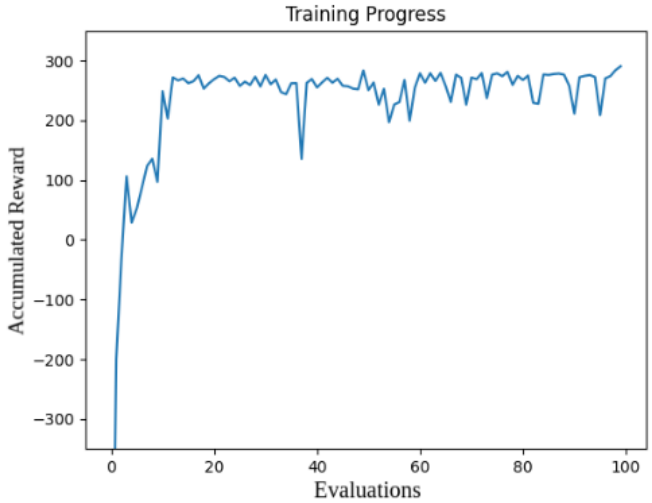

Figure 5. Agent performance in Lunar Lander with the basic Bayesian optimization version.

As we have previously examined the convergency with success in the Cartpole environment, we have omitted the experiment of performing multifidelity Bayesian optimization in the Lunar Lander environment. LunarLander's complexity surpasses CartPole's, with its agent navigating a wider array of states and actions, making learning slower and computationally demanding. This initial multifidelity Bayesian optimization experiment allocated 3 h (10,800 s) and 1,000,000 timesteps—higher than CartPole—to ensure sufficient training for configuration validation. The resulting hyperparameters—discount factor: 0.95547, GAE lambda: 0.93638, learning rate: 0.00016—achieved the best performance among 12 tested configurations, with a mean validation reward of 272, a 13% improvement over the 239 from the non-multifidelity approach. The plot (**Figure 6**) illustrates training and evaluation, highlighting multifidelity's ability to explore more configurations efficiently within the same timeframe, enhancing hyperparameter space search. These findings demonstrate that multifidelity Bayesian optimization yields more effective configurations in

LunarLander, improving average reward and training efficiency despite the environment's complexity.

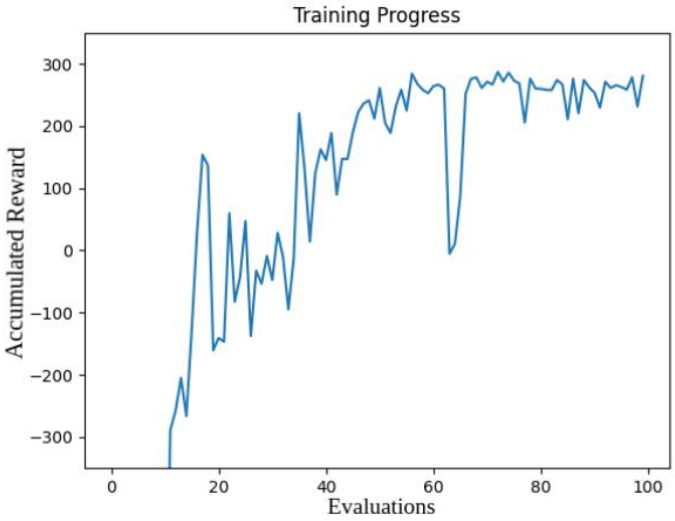

Figure 6. Agent performance in Lunar Lander with the multi fidelity Bayesian optimization version.

5. Conclusions, discussion and further work

This study compares basic and multifidelity Bayesian hyperparameter optimization in deep reinforcement learning scenarios with the assumption that the performance displayed by this method in supervised learning real-world problems is also going to be shown in the case of deep reinforcement learning common problems [23] such as in CartPole and LunarLander, revealing distinct performance and stability differences. We believe that the significance of testing the research hypothesis that multi-fidelity Bayesian optimization outperforms Bayesian optimization in the hyperparameter tuning of deep reinforcement learning algorithms setup is critical, as its use is going to incur a cheaper solution to this problem, which implies in the process being more green and computationally efficient. Regarding the results presented in the previous section, we can see that in the CartPole setup (**Table 1**), both approaches achieved a maximum reward of 500 at 10 timesteps in the baseline, but at 2500 timesteps, multifidelity maintained 500 versus the basic's 487 (2.67% improvement). For convergence, the basic version reached 403 at 6000 timesteps, while multifidelity hit 476 at 8000 timesteps (15% reward gain, though 30% slower), showcasing enhanced stability and efficiency.

Table 1. Comparison of results obtained with and without multifidelity in CartPole.

Metric	Basic Version	Multifidelity Version
Timesteps: 10,000	Accumulated Reward: 500	Accumulated Reward: 500
Timesteps: 2500	Accumulated Reward: 487	Accumulated Reward: 500
Timesteps When Reward Exceeds 350	Timesteps: 6000	Timesteps: 8000
Timesteps When Reward Exceeds 350	Accumulated Reward: 403	Accumulated Reward: 476

In LunarLander (**Table 2**), multifidelity outperformed the baseline at 1,000,000 timesteps (272 vs. 239), and at 250,000 timesteps (253 vs. 163). For convergence, multifidelity surpassed a 200-reward threshold in 40,000 timesteps (267) versus the basic's 58,000 timesteps (223), a 30% step reduction and 17% reward increase. Multifidelity consistently demonstrated superior robustness and reliability across experiments according to the results that we have obtained.

Table 2. Comparison of results obtained with and without multifidelity in LunarLander.

Metric	Basic Version	Multifidelity Version
Timesteps: 1,000,000	Accumulated Reward: 239	Accumulated Reward: 272
Timesteps: 250,000	Accumulated Reward: 163	Accumulated Reward: 253
Timesteps When Reward Exceeds 200	Timesteps: 58,000	Timesteps: 40,000
Timesteps When Reward Exceeds 200	Accumulated Reward: 223	Accumulated Reward: 267

Overall, multifidelity optimization enhanced performance, stability, and convergence speed in both environments, proving more efficient and effective for agent training, particularly in complex settings like LunarLander. This improvement of the performance of the algorithm can be used to make the hyperparameter tuning of deep reinforcement learning algorithms process cheaper or to make more evaluations given the same budget to improve performance, according to the needs of the practitioner, which we believe is a critical advantage with respect to just applying standard Bayesian optimization techniques that do not take into account a different set of fidelities.

The paper adds empirical evidence with Lunar Lander and Cartpole regarding the usefulness of the multifidelity Bayesian optimization framework. However, we recognize that these are very easy standard reinforcement learning settings in comparison to the whole environments that can be tested. Consequently, more empirical evidence should be explored to reinforce the evidence of the hypothesis that we have shown in this work.

Future research could explore several avenues to enhance the multifidelity Bayesian optimization framework for RL hyperparameter tuning [24]. First, extending the approach to more complex, high-dimensional environments [25], such as robotic control tasks or multi-agent systems [26], explainable financial setups [27], could test its scalability and robustness [28] beyond CartPole and LunarLander, addressing computational bottlenecks noted in prior studies [1]. Second, integrating adaptive fidelity selection mechanisms, building on [10], could dynamically adjust evaluation budgets based on real-time convergence signals, potentially improving efficiency over static multifidelity setups. Third, incorporating transfer learning [29,30] to leverage optimized hyperparameters across related tasks, as suggested by [8], might reduce training times in LunarLander-like scenarios. Finally, comparing multifidelity optimization against emerging techniques, such as population-based training [31], or using advanced information Bayesian optimization [32], could clarify its relative advantages in diverse RL contexts, fostering broader adoption.

Author contributions: Conceptualization, ECGM and MM; methodology, MM and ECGM; software, GM; validation, MM; formal analysis, MM; investigation, ECGM;

writing—original draft preparation, ECGM, MM and GM; writing—review and editing, ECGM and MM; visualization, MM and GM; supervision, MM; project administration, ECGM and MM. All authors have read and agreed to the published version of the manuscript.

Institutional review board statement: Not applicable.

Informed consent statement: Not applicable.

Conflict of interest: The authors declare no conflict of interest.

References

1. Sutton RS, Barto AG. Reinforcement learning: An introduction, 2nd ed. MIT Press; 2018.
2. Tang C, Abbatematteo B, Hu J, et al. Deep Reinforcement Learning for Robotics: A Survey of Real-World Successes. Annual Review of Control, Robotics, and Autonomous Systems; 2024. doi: 10.1146/annurev-control-030323-022510
3. Parker-Holder J, Rajan R, Song X, et al. Automated Reinforcement Learning (AutoRL): A Survey and Open Problems. Journal of Artificial Intelligence Research. 2022; 74: 517-568. doi: 10.1613/jair.1.13596
4. Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning. Nature. 2015; 518(7540): 529-533. doi: 10.1038/nature14236
5. Sahu SK, Mokhade A, Bokde ND. An Overview of Machine Learning, Deep Learning, and Reinforcement Learning-Based Techniques in Quantitative Finance: Recent Progress and Challenges. Applied Sciences. 2023; 13(3): 1956. doi: 10.3390/app13031956
6. Gong S, Wang M, Gu B, et al. Bayesian Optimization Enhanced Deep Reinforcement Learning for Trajectory Planning and Network Formation in Multi-UAV Networks. IEEE Transactions on Vehicular Technology. 2023; 72(8): 10933-10948. doi: 10.1109/tvt.2023.3262778
7. Yan Q, Wang H, Ma Y, et al. Uncertainty estimation in HDR imaging with Bayesian neural networks. Pattern Recognition. 2024; 156: 110802. doi: 10.1016/j.patcog.2024.110802
8. Snoek J, Larochelle H, Adams RP. Practical bayesian optimization of machine learning algorithms. Advances in neural information processing systems; 2012.
9. Li S, Su S, Lin X. Optimizing the hyper-parameters of deep reinforcement learning for building control. In: Building Simulation. Tsinghua University Press; 2025. doi: 10.1007/s12273-025-1233-y
10. Kandasamy K, Schneider J, Póczos B. High dimensional Bayesian optimisation with multifidelity models. In: Proceedings of the 34th International Conference on Machine Learning; 2017.
11. Lin Q, Hu J, Zhou Q, et al. A Multi-Fidelity Bayesian Optimization Approach for Constrained Multi-Objective Optimization Problems. Journal of Mechanical Design. 2024. doi: 10.1115/1.4064244
12. Garrido Merchán EC. Advanced methods for bayesian optimization in complex scenarios [PhD thesis]. Universidad Autónoma de Madrid; 2021.
13. Garrido-Merchán EC. Information-theoretic Bayesian Optimization: Survey and Tutorial. arXiv; 2025.
14. Matsuo Y, LeCun Y, Sahani M, et al. Deep learning, reinforcement learning, and world models. Neural Networks. 2022; 152: 267-275. doi: 10.1016/j.neunet.2022.03.037
15. Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms. arXiv; 2017.
16. Hutter F, Hoos HH, Leyton-Brown K. Sequential model-based optimization for general algorithm configuration. In: Learning and intelligent optimization, Proceedings of the 5th international conference. Springer Berlin Heidelberg; 2011.
17. Del Rio A, Jimenez D, Serrano J. Comparative Analysis of A3C and PPO Algorithms in Reinforcement Learning: A Survey on General Environments. IEEE Access. 2024; 12: 146795-146806. doi: 10.1109/access.2024.3472473
18. Veeramanikandan K, Ponnusamy R. Deep-Q Classifier for Predicting Balanced and Imbalanced Features in Cartpole and Lunarlander Dataset. In: Proceedings of the 2023 International Conference on Data Science, Agents & Artificial Intelligence (ICDSAAI); 2023. doi: 10.1109/icdsaai59313.2023.10452598
19. Xiong Y, Hu Z, Huang Y, et al. XRL-Bench: A Benchmark for Evaluating and Comparing Explainable Reinforcement Learning Techniques. In: Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining; 2024. doi: 10.1145/3637528.3671595

20. Irshad F, Karsch S, Döpp A. Multi-objective and multi-fidelity Bayesian optimization of laser-plasma acceleration. Physical Review Research. 2023; 5(1). doi: 10.1103/physrevresearch.5.013063
21. Winter JM, Abaidi R, Kaiser JWJ, et al. Multi-fidelity Bayesian optimization to solve the inverse Stefan problem. Computer Methods in Applied Mechanics and Engineering. 2023; 410: 115946. doi: 10.1016/j.cma.2023.115946
22. Dayal A, Cenkeramaddi LR, Jha A. Reward criteria impact on the performance of reinforcement learning agent for autonomous navigation. Applied Soft Computing. 2022; 126: 109241. doi: 10.1016/j.asoc.2022.109241
23. Folch JP, Lee RM, Shafei B, et al. Combining multi-fidelity modelling and asynchronous batch Bayesian Optimization. Computers & Chemical Engineering. 2023; 172: 108194. doi: 10.1016/j.compchemeng.2023.108194
24. Shu L, Jiang P, Wang Y. A multi-fidelity Bayesian optimization approach based on the expected further improvement. Structural and Multidisciplinary Optimization. 2020; 63(4): 1709-1719. doi: 10.1007/s00158-020-02772-4
25. Binois M, Wycoff N. A Survey on High-dimensional Gaussian Process Modeling with Application to Bayesian Optimization. ACM Transactions on Evolutionary Learning and Optimization. 2022; 2(2): 1-26. doi: 10.1145/3545611
26. Ning Z, Xie L. A survey on multi-agent reinforcement learning and its application. Journal of Automation and Intelligence. 2024; 3(2): 73-91. doi: 10.1016/j.jai.2024.02.003
27. de-la-Rica-Escudero A, Garrido-Merchán EC, Coronado-Vaca M. Explainable post hoc portfolio management financial policy of a Deep Reinforcement Learning agent. PLOS ONE. 2025; 20(1): e0315528. doi: 10.1371/journal.pone.0315528
28. Moos J, Hansel K, Abdulsamad H, et al. Robust Reinforcement Learning: A Review of Foundations and Recent Advances. Machine Learning and Knowledge Extraction. 2022; 4(1): 276-315. doi: 10.3390/make4010013
29. Ju H, Juan R, Gomez R, et al. Transferring policy of deep reinforcement learning from simulation to reality for robotics. Nature Machine Intelligence. 2022; 4(12): 1077-1087. doi: 10.1038/s42256-022-00573-6
30. Zhu Z, Lin K, Jain AK, et al. Transfer Learning in Deep Reinforcement Learning: A Survey. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2023; 45(11): 13344-13362. doi: 10.1109/tpami.2023.3292075
31. Jaderberg M, Dalibard V, Osindero S, et al. Population based training of neural networks. arXiv; 2017.
32. Fernández-Sánchez D, Garrido-Merchán EC, Hernández-Lobato D. Alpha Entropy Search for New Information-based Bayesian Optimization. arXiv; 2024.