ORSO: Accelerating Reward Design via Online Reward Selection and Policy Optimization

Chen Bo Calvin Zhang
cbczhang@mit.edu
Massachusetts Institute of Technology, ETH Zurich

Zhang-Wei Hong
zwhong@mit.edu
Massachusetts Institute of Technology

Aldo Pacchiano
pacchian@bu.edu
Boston University, Broad Institute of MIT and Harvard

Pulkit Agrawal
pulkitag@mit.edu
Massachusetts Institute of Technology

Abstract

Reinforcement learning (RL) algorithms require carefully designed shaped reward functions to learn effective policies, especially in environments with sparse task rewards. However, manually designing a suitably shaped reward function is challenging and often requires extensive domain knowledge and trial and error. Current methods for automating reward design can be prohibitively time-consuming. In this paper, we cast the reward design process as an online model selection problem and propose ORSO (**O**nline **R**eward **S**election and Policy **O**ptimization), a novel algorithm to efficiently design shaped reward functions. Because existing online model selection algorithms are provably efficient, ORSO can identify effective reward functions efficiently. We provide regret guarantees for ORSO and demonstrate its effectiveness on several continuous control benchmarks. Compared to prior methods, ORSO is more sample-efficient, consistently finds high-quality dense reward functions, and achieves similar performance to hand-engineered rewards created by domain experts.

1 Introduction

Reinforcement learning (RL) (Sutton and Barto, 2018) is a powerful framework for learning a policy for an agent (e.g., a robot) that maximizes some reward function through interaction with an environment. The reward function specifies the task's goal or objective to be achieved. Many interesting real-world tasks are sparse-reward in nature, meaning the learning agent only receives a reward signal upon task completion. This sparsity of rewards makes it difficult for RL agents to learn a good policy, as there is a lack of learning signals to guide the agent toward the desired behavior during training. To address this issue, a common approach is to introduce dense, shaped rewards that provide informative feedback at each step, guiding the agent toward desired behaviors (Margolis and Agrawal, 2023). However, designing a reward function that accurately captures the desired behavior and provides sufficient learning signals remains a significant challenge (Singh et al., 2009; Ng et al., 1999). These reward-shaping techniques essentially modify the RL training objective. For instance, a task reward function for a door-opening robot might only consider whether the door is open or not, providing little learning signal. To guide the robot towards opening the door more easily, a shaped reward function could combine the final door-open reward with an intermediate reward for minimizing the distance between the robot's hand and the door handle.

Manually designing shaped reward functions (Margolis and Agrawal, 2023; Liu et al., 2024) requires significant domain expertise and can be cumbersome for complex tasks. Alternative approaches attempted to automate the reward design process using coding language models (Ma et al., 2023). Both manual reward design and prior automated methods essentially perform an evolutionary search, iteratively training policies with selected reward functions and observing their performance. This process is time-consuming, as it requires training policies to completion for each reward function candidate.

In this paper, we formulate the reward design process as an online learning problem, which allows us to leverage techniques from the online model selection literature to efficiently search over reward function candidates. We propose ORSO (**O**nline **R**eward **S**election and Policy **O**ptimization). ORSO operates in two phases: (1) generating a set of candidate reward functions using a stochastic generator, and (2) efficiently identifying the best reward function from this set via online reward selection and policy optimization.

Specifically, we leverage the coding abilities of large language models like GPT-4 (Achiam et al., 2023) to generate Python code for reward functions based on minimal environment descriptions and formulate the problem of reward selection as a model selection problem (Pacchiano et al., 2020; 2023; Agarwal et al., 2017; Foster et al., 2019; Lee et al., 2021), where each model represents a candidate reward function (and its associated policy).

We provide regret guarantees for ORSO under a specific model selection algorithm and demonstrate its effectiveness on several continuous control tasks using the Isaac Gym simulator (Makoviychuk et al., 2021). Compared to prior methods, ORSO consistently finds high-quality reward functions that lead to policies achieving similar performance to hand-engineered rewards created by domain experts. Moreover, ORSO is more sample-efficient, requiring substantially fewer environment interactions and less computation time than evolutionary methods.

The paper is structured as follows. Section 2 states the reward design problem and provides some notation and necessary background. Section 3 introduces the ORSO algorithm, detailing the reward design and online reward selection phases. Section 4 presents theoretical guarantees on the convergence of ORSO and Section 5 describes the experimental setup and results.

2 Preliminaries

In this section, we introduce the necessary notation for our analysis and formally define the reward design problem.

Reinforcement Learning (RL) In RL, the objective is to learn a policy for an agent (e.g., a robot) that maximizes the expected cumulative reward during the interaction with the environment. The interaction between the agent and the environment is formulated as a Markov decision process (MDP) (Puterman, 2014), $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \tilde{r}, \gamma, \rho_0)$, where the $\mathcal{S}$ and $\mathcal{A}$ denote state and action spaces, respectively, $P : \mathcal{S} \times \mathcal{A} \to \Delta_{\mathcal{S}}$[1] is the state transition dynamics, $\tilde{r} : \mathcal{S} \times \mathcal{A} \to \Delta_{\mathbb{R}}$ denotes the reward function, $\gamma \in [0, 1)$ is the discount factor, and $\rho_0 \in \Delta_{\mathcal{S}}$ is the initial state distribution. At each timestep $t \in \mathbb{N}$ of interaction, the agent selects an action $a_t \sim \pi(\,\cdot \mid s_t)$ based on its policy π, receives a (possibly) stochastic reward $\tilde{r}_t \sim \tilde{r}(s_t, a_t)$, and transitions to the next state s_{t+1} according to the transition dynamics $P(\,\cdot \mid s_t, a_t)$. RL algorithms aim to find a policy $\pi^\star$ that maximizes the discounted cumulative reward, i.e.,

$$\pi^\star \in \arg\max_{\pi} \mathcal{J}_{\tilde{r}}(\pi) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \tilde{r}_t \;\middle|\; \begin{array}{l} s_0 \sim \rho_0,\ a_t \sim \pi(\,\cdot \mid s_t), \\ \tilde{r}_t \sim \tilde{r}(s_t, a_t),\ s_{t+1} \sim P(\,\cdot \mid s_t, a_t) \end{array}\right]. \tag{1}$$

Reward Design (RD) The reward function $\tilde{r}$ encodes the task objective, but it is often sparse, complicating the RL optimization process. It is common to design a dense reward function to facilitate learning. We formalize the reward design (RD) problem as follows.

Definition 2.1 (Reward Design)**.** *Let $\mathfrak{A}$ be a reinforcement learning algorithm that takes an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \tilde{r}, \gamma, \rho_0)$ as input and returns a policy $\pi^{\mathcal{M}} = \mathfrak{A}(\mathcal{M})$. Given $\mathcal{M}$ and $\mathfrak{A}$, the reward*

[1]$\Delta_{\mathcal{S}}$ denotes the set of probability distributions over $\mathcal{S}$.

design problem aims to find a reward function $r : \mathcal{S} \times \mathcal{A} \to \Delta_{\mathbb{R}}$ such that the policy $\pi^{\mathcal{M}_r} = \mathfrak{A}(\mathcal{M}_r)$, where $\mathcal{M}_r = (\mathcal{S}, \mathcal{A}, P, r, \gamma, \rho_0)$, achieves a higher expected return under the original reward $\tilde{r}$, i.e.,

$$\mathcal{J}_{\tilde{r}}(\pi^{\mathcal{M}_r}) \geq \mathcal{J}_{\tilde{r}}(\pi^{\mathcal{M}}). \tag{2}$$

The space of possible reward functions $\mathcal{R} = \{r \mid r : \mathcal{S} \times \mathcal{A} \to \Delta_{\mathbb{R}}\}$ is typically infinite, making exhaustive search infeasible. However, we can usually have access to a set of candidate reward functions, $\mathcal{R}^K = \left\{r^1, \ldots, r^K\right\}$. How do we select the optimal one efficiently? Naively training using each r^i and selecting the best-performing one can be computationally prohibitive. Consequently, we aim to develop more efficient techniques for reward selection that can explore and exploit the candidate reward functions without training on each reward function until convergence. In this paper, we cast the problem into a model selection problem.

Model Selection We borrow the notation from Pacchiano et al. (2023). We consider a general sequential decision-making process consisting of a meta-learner interacting with an environment over $T \in \mathbb{N}$ rounds. In each round $t = 1, 2, \ldots, T$, the agent selects a policy π_t and after executing policy π_t, the environment returns a reward $r_t \in \mathbb{R}$. The objective of the learner is to choose policies $\pi_1, \ldots, \pi_T$ to maximize the expected cumulative sum of rewards $\mathbb{E}\left[\sum_{t=1}^{T} r_t\right]$. We denote by $v_\pi = \mathbb{E}[r \mid \pi]$ the expected reward, given that the learner plays policy π, i.e., the value of policy π. The instantaneous regret of π is $\mathrm{reg}(\pi) = v^\star - v^\pi$, where $v^\star$ is the value of the optimal policy. The cumulative regret after T rounds of interaction is $\mathrm{Reg}(T) = \sum_{t=1}^{T} \mathrm{reg}(\pi_t)$. The total reward accumulated by the algorithm over T rounds is denoted by $u_T = \sum_{t=1}^{T} v^{\pi_t}$.

In model selection, the meta-learner interacts with an environment over T rounds, selecting from K base learners (reward function candidates). In each round t, the meta-learner picks a base learner $i_t \in [K]$ and follows its policy, updating the learner's state with new data. Unlike multi-armed bandits (MAB), where mean rewards are stationary, the mean rewards here are non-stationary due to the stateful nature of base learners (the base learners are learning as they see more data), making the design of effective model selection algorithms challenging.

The policy associated with base learner i at round t is denoted by π_t^i, so that $\pi_t = \pi_t^{i_t}$. We denote the number base learner i has been played up to round t as $n_t^i = \sum_{\ell=1}^{t} \mathbb{1}\left\{i_t = i\right\}$ and the total cumulative reward for learner i as $u_t^i = \sum_{\ell=1}^{t} \mathbb{1}\left\{i_t = i\right\} v^{\pi_t^i}$. We denote the internal clock for each base learner with a subscript (k) such that $\pi_{(k)}^i$ is the policy of learner i when chosen for the k-th time, i.e., $\pi_t^i = \pi_{(n_t^i)}^i$.

3 Orso: <u>O</u>nline <u>R</u>eward <u>S</u>election and Policy <u>O</u>ptimization

In this section, we introduce Orso (Online Reward Selection and Optimization), a novel approach to efficiently and effectively design reward functions for reinforcement learning. Our method operates in two phases: (1) reward generation and (2) online reward selection and policy optimization. Figure 1 illustrates the main components of Orso., which will be further described in this section.

Reward Generation In the first phase of Orso, we generate a set of candidate reward functions $\mathcal{R}^K$ for the online selection phase. Given an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \tilde{r}, \rho_0)$ and a stochastic generator G, we sample K reward function candidates, $\mathcal{R}^K = \left\{r^1, \ldots, r^K \;\middle|\; \forall i \in [K],\ r^i : \mathcal{S} \times \mathcal{A} \to \Delta_{\mathbb{R}},\ r^i \sim G\right\}$, from G during the reward design phase. The generator G can be any distribution over the reward function space $\mathcal{R}$. For instance, if the set of possible reward functions is given by a linear combination of two reward components c_1, c_2, which are functions of the current state and action, such that $r(s, a) = w_1 c_1(s, a) + w_2 c_2(s, a)$, then the generator G can be represented by the means and variances of two normal distributions, one for each weight w_1, w_2.

Online Reward Selection and Policy Optimization Our algorithm for online reward selection and policy optimization is described in Algorithm 1. On a high level, the algorithm proceeds as

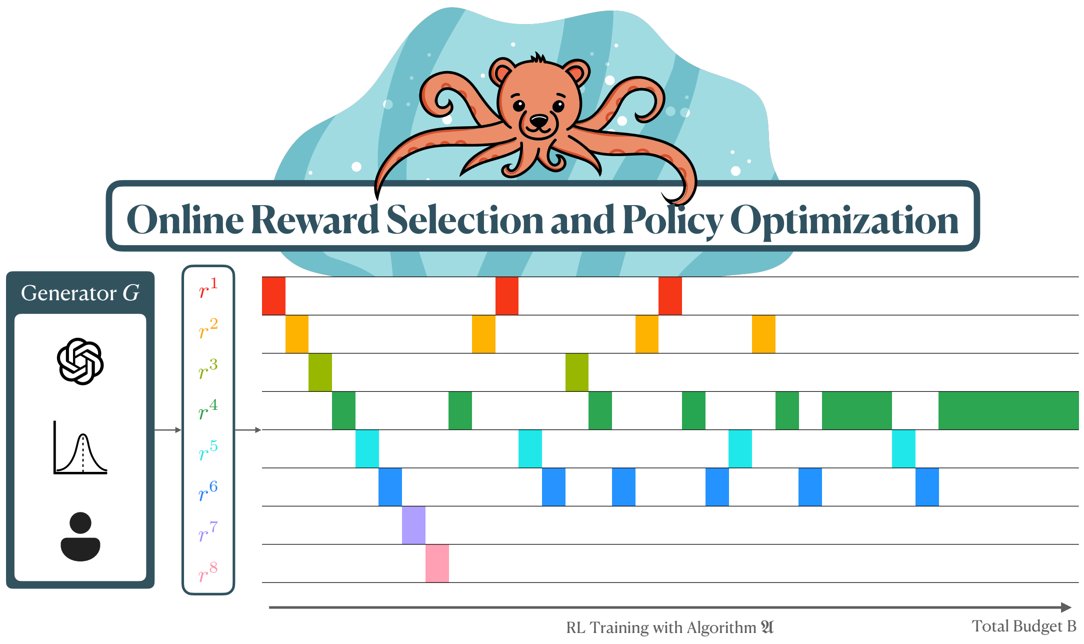

Online Reward Selection and Policy Optimization
Generator G
r^1
r^2
r^3
r^4
r^5
r^6
r^7
r^8
RL Training with Algorithm $\mathfrak{A}$
Total Budget B

Figure 1: Overview of ORSO. ORSO samples a set of reward functions $\mathcal{R}^K$ from a stochastic generator G (e.g., a human reward engineer, a large language model, a distribution over reward component coefficients). During the online reward selection and policy optimization phase, ORSO actively selects which reward function to use at every iteration of RL training using a model selection algorithm. The algorithm returns the best reward function according to the ground truth task performance metric.

follows. Given an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \tilde{r}, \rho_0)$, an RL algorithm $\mathfrak{A}$ and a reward generator G, we sample set of K reward functions $\mathcal{R}^K \sim G$ and initialize K distinct policies $\pi^1, \ldots, \pi^K$. The base learners are represented by tuples $\left\{(\pi^i, r^i)\right\}_{i=1}^K$ of policies and reward functions. During each step t of the reward selection process, the algorithm selects a base learner $i_t \in [K]$ according to a selection strategy. We then construct the MDP corresponding to the selected reward function, $\mathcal{M}_{i_t} = (\mathcal{S}, \mathcal{A}, P, r^{i_t}, \rho_0)$, and perform N iterations of training with algorithm $\mathfrak{A}$, updating the policy corresponding to reward function i_t to obtain π^{i_t}. Policy π^{i_t} is simultaneously evaluated under the true reward function $\tilde{r}$ and the necessary variables for the model selection algorithm are then updated (e.g., reward estimates, arm visitation counts, and confidence intervals). The algorithm returns the reward function r and the corresponding policy π that performs the best under the true reward function $\tilde{r}$.

3.1 Implementation

In our experiments, we seek to remove the necessity of designing reward components manually, therefore we employ a large language model to write Python code for the reward functions. Sampling reward functions from the language model could result in invalid reward functions, such as those with infinite or non-numerical values. To address this, we employ rejection sampling to discard samples that do not meet specific criteria. Section 5 provides further details on the implementation of the sampling process and the rejection criteria.

Algorithm 1 ORSO: Online Reward Selection and Policy Optimization

```
Require: MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \tilde{r}, \rho_0)$, algorithm $\mathfrak{A}$, generator $G$
 1: Sample $K$ reward functions $\mathcal{R}^K = \{r^1, \ldots, r^K\} \sim G$
 2: Initialize $K$ policies $\Pi = \{\pi^1, \ldots, \pi^K\}$
 3: for $t = 1, 2, \ldots, T$ do
 4:     Select an arm $i_t \in [K]$ according to a selection strategy
 5:     Construct MDP $\mathcal{M}_{i_t} = (\mathcal{S}, \mathcal{A}, P, r^{i_t}, \rho_0)$
 6:     Update $\pi^{i_t} \leftarrow \mathfrak{A}(\mathcal{M}_{i_t}, N, \pi^{i_t})$ and evaluate $r_t \leftarrow \texttt{Eval}(\pi^{i_t})$
 7:     Update necessary variables
 8: end for
 9: // Best policy and reward function under the true reward function
10: return $\pi, r = \arg\max_{i \in [K]} (\texttt{Eval}(\pi^i))$
```

4 Theoretical Guarantees

In this section, we provide regret guarantees for ORSO with D^{3}RB as the selection algorithm. The main idea underlying our regret guarantees is that the internal state of all suboptimal reward functions is only updated up to a point where the regret equals that of the best algorithm. We first introduce some useful definitions for our analysis.

Definition 4.1 (Regret scale and coefficients (Definition 2.1 from Pacchiano et al. (2023)))**.** *The regret of base learner i after being played k times is $\frac{\sum_{\ell=1}^{k} \text{reg}(\pi^i_{(\ell)})}{\sqrt{k}}$. For a positive constant $d_{\min}$, the regret coefficient of base learner i after being played for k rounds is*

$$d^i_{(k)} = \max\left\{ d_{\min}, \frac{\sum_{\ell=1}^{k} \text{reg}(\pi^i_{(\ell)})}{\sqrt{k}} \right\}. \tag{3}$$

We assume there exists a base learner that monotonically dominates every other learner.

Assumption 4.2. *There is a base learner $i_\star$ such that at all time steps, its expected sum of rewards dominates any other learner, i.e., $u^{i_\star}_{(t)} \geq u^k_{(t)}$, for all $i \in [K]$ and such that its average expected rewards are increasing, i.e., $\frac{u^{i_\star}_{(t)}}{t} \leq \frac{u^{i_\star}_{(t+1)}}{t+1}$, for all $t \in \mathbb{N}$. This is equivalent to saying that $d^{i_\star}_{(t)} \geq d^{i_\star}_{(t+1)}$ for all $t \in \mathbb{N}$.*

Assumption 4.2 implies that learner $i_\star$'s cumulative expected reward sequences are always at least as large as the sum of rewards for any other learner and that its average performance is increasing monotonically. Following the notation of Pacchiano et al. (2023), we refer to the event that the confidence intervals for the reward estimator are valid as $\mathcal{E}$. Then we can refine Lemma 9.3 from Pacchiano et al. (2023) in the case where Assumption 4.2 holds.

Lemma 4.3. *Under event $\mathcal{E}$ and Assumption 4.2, with probability $1-\delta$, the regret of all base learners i is bounded in all rounds T as*

$$\sum_{t=1}^{n^i_T} \text{reg}(\pi^i_{(t)}) \leq 6 d^{i_\star}_T \sqrt{n^{i_\star}_T + 1} + 5c\sqrt{(n^{i_\star}_T + 1)\ln\frac{K \ln T}{\delta}}, \tag{4}$$

where $d^{i_\star}_T = d^{i_\star}_{(n^{i_\star}_T)}$.

We provide the proof for Lemma 4.3 in Section 9. Lemma 4.3 implies that when Assumption 4.2 holds, the regrets are perfectly balanced. This is in stark contrast with the regret guarantees of Pacchiano et al. (2023) that prove the D^{3}RB algorithm's overall regret to scale as $\left(\bar{d}^{(i_\star)}_T\right)^2 \sqrt{T}$ where $\bar{d}^{i_\star}_t = \max_{\ell \leq t} d^{i_\star}_\ell$. Instead, our results above depend not on the monotonic regret coefficients $\bar{d}^{i_\star}_t$ but on the true regret coefficients $d_t^{i_\star}$. Even if learner $i_\star$ has a slow start (and therefore a large $\bar{d}_T^{i_\star}$), as long as monotonicity holds and the $i_\star$-th learner recovers in the later stages of learning, our results show that D^{3}RB will achieve a regret guarantee comparable with running learner $i_\star$ in isolation.

5 Practical Implementation and Experimental Results

In this section, we present a practical implementation of Orso and its experimental results on several continuous control tasks. We evaluate the performance of Orso along two axes. Firstly, we show the ability of Orso to design effective reward functions, meaning reward functions that lead to policies that perform well under the true reward function. More importantly, we show that Orso is more efficient than the conventional method of training policies on each reward function until convergence.

5.1 Experimental Setup

Environments and RL Algorithm We evaluate Orso on a set of continuous control tasks using the Isaac Gym simulator (Makoviychuk et al., 2021). We train our policies using proximal policy optimization (PPO) (Schulman et al., 2017) and build our implementation on CleanRL (Huang et al., 2022). We use the hyperparameters provided by the environment developers.

5.1.1 Baselines

The rewards generated by Orso are compared against three baselines. We analyze the performance of policies trained using each reward function detailed below. We also evaluate the reward function design efficiency of Orso compared to Eureka (Ma et al., 2023).

Dense/Shaped (DS). Firstly, we consider the human-designed reward functions for each task. These are highly engineered reward functions provided by the developers of the environments. We note that these are constructed such that training PPO with the given hyperparameters yields a performant policy with respect to the ground truth reward function.

Sparse/Unshaped (SU). This is the ground truth reward function r for each MDP. These reward functions can be sparse (for manipulation) or unshaped (for locomotion). We use the same reward definitions as Eureka, which we report in Section 10.

Eureka. Similarly to our implementation, Eureka uses a large language model to generate Python code for the reward functions of several continuous control tasks. Eureka uses an evolutionary scheme to evaluate and improve its reward functions. During each iteration, Eureka samples a set of reward functions from an LLM, trains an RL algorithm on each reward function, and uses the best-performing reward function as a context for the LLM to perform the evolutionary step. In our experiments, we sample 4 reward functions and perform 5 evolutionary steps in Eureka, for a total of 20 candidate reward functions.

5.1.2 Reward Generation

We use GPT-4 (Achiam et al., 2023) as a reward generator to avoid having to manually design reward function components. The language model is prompted to generate reward function code in Python based on some minimal environment code describing the observation space and useful class variables.

While the LLM produces seemingly good code, this does not guarantee that the sampled code is bug-free and runnable. We employ a simple rejection sampling technique to construct sets of only valid reward functions with high probability, such that reward functions that are not compilable or that produce $\pm\infty$ or `NaN` values are discarded. The details of the rejection sampling mechanism can be found in Section 11. In our experiments, we find that sampling $K = 16$ valid reward functions from G suffices to produce at least one effective reward function.

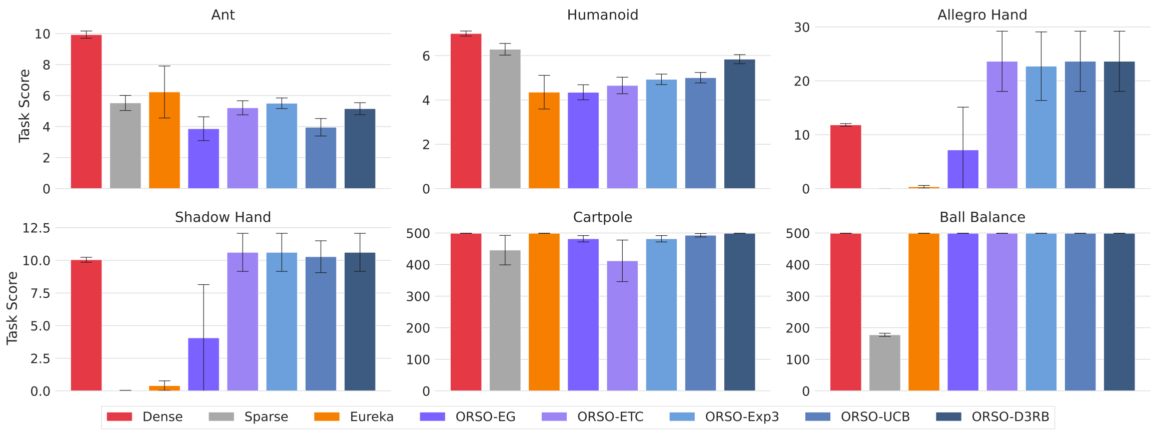


Figure 2: Mean performance of selected arms on each task with 95% confidence intervals.

5.1.3 Online Reward Selection and Policy Optimization

We evaluate multiple reward selection algorithms from the multi-armed bandit and model selection literature: explore-then-commit (ETC), ε-greedy (EG), upper confidence bound (UCB) (Auer, 2002), exponential-weight algorithm for exploration and exploitation (EXP3) (Auer et al., 2002), and doubling data-driven regret balancing (D3RB) (Pacchiano et al., 2023). We provide the pseudocode and the hyperparameters used for each selection algorithm in Section 13.

Since our reward selection algorithm operates in an online fashion, it is not necessary to train on each reward function until convergence as suboptimal arms are discarded early on or played less often. We can therefore reduce the total number of iterations required to find the optimal arm. For this reason, we train for a total of $\ln(K) \times$ `n_iters` iterations with Orso, where `n_iters` is the number of iterations used for training PPO with the DS reward function.

5.2 Main Results

We evaluate Orso's ability to efficiently choose the optimal reward function from $\mathcal{R}^K$, comparing various online selection strategies' performances. Each algorithm is applied to the same $\mathcal{R}^K \sim G$ set for fairness. We repeat this process with different random seeds, generating diverse sets of reward functions from G. This allows us to evaluate the LLM's reward function generation effectiveness. To evaluate the selected reward functions' quality, we train a PPO agent using each best reward function for `n_iters` iterations. Additionally, we compare our results with Eureka, training and evaluating its top reward functions across multiple seeds. PPO is also run on DS and SU reward functions for further comparison.

Orso Reward Functions Achieve High True Reward In Figure 2, we report the mean evaluation performance for the reward function selected by each method with 95% confidence intervals. We can observe that the performance of Orso is consistently similar to or better than that of the DS reward function and improves upon the true SU reward function.

We were unable to replicate the results of Eureka reported by Ma et al. (2023) in manipulation tasks. To investigate this discrepancy, we trained a policy for the Shadow Hand using the reward functions specified by Ma et al. (2023), achieving a maximum task performance of approximately 0.80 across 3 seeds. We found that the performance reported by Ma et al. (2023) was due to the inclusion of a manually added bonus term in the DS reward function, which Orso generated automatically.

Orso Efficiently Selects an Effective Reward Functions We measure the efficiency of our algorithm by the number of PPO iterations necessary to find a good reward function. This is equivalent

Table 1: Average number of PPO iterations for each algorithm as a multiple of `n_iters`

Environment	Eureka	Orso (ours)
Ant	13.20 ± 2.50	2.77
Humanoid	9.80 ± 4.48	2.77
Allegro Hand	6.67 ± 2.82	2.77
Shadow Hand	5.25 ± 4.67	2.77
Cartpole	15.80 ± 1.40	2.77
Ball Balance	16.00 ± 1.66	2.77

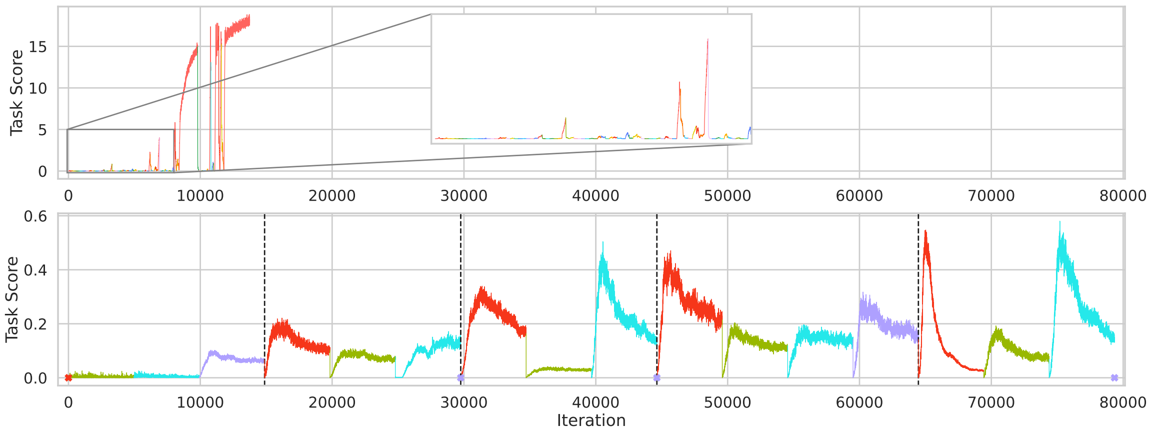


Figure 3: Task score of Orso (top) and Eureka (bottom) on Allegro Hand during training. The different colors represent different reward functions used during training. The vertical dashed lines indicate the evolution steps in Eureka. The $\times$'s symbolize reward functions that throw an error when used for training.

to comparing sample efficiency and wall-clock/compute efficiency for a given task and hardware configuration. Table 1 reports the number of iterations necessary to select the reward function as a multiple of `n_iters` with 95% confidence intervals. Orso requires only $\ln(K) \approx 2.77$ `n_iters` iterations when $K = 16$. On the other hand, in the worst case, Eureka with 4 rewards per evolution requires 20 `n_iters` iterations. However, this is not the case in practice as Eureka does not reject invalid rewards before starting training. Therefore, we see that the effective multiple of `n_iters` is lower.

To better visualize how Orso selects the best reward function and discards suboptimal ones efficiently, we plot the task score for Allegro Hand during training of both Orso (with D^{3}RB) and Eureka in Figure 3. We can observe that Orso initially switches reward function frequently, but can quickly discard suboptimal arms and select the best reward function. On the other hand, Eureka trains on each reward function for `n_iter` iterations, regardless of its performance so far. We report additional curves for all tasks and algorithms in Section 12.

6 Related Work

Reward Design for RL Designing effective reward functions for reinforcement learning has been a long-standing challenge. Previous approaches include manually specifying reward components and tuning their coefficients (Ng et al., 1999; Margolis and Agrawal, 2023), learning reward models from demonstrations (Ziebart et al., 2008; Ho and Ermon, 2016; Abbeel and Ng, 2004) or preferences (Zhang and Ramponi, 2023; Christiano et al., 2017), and using evolutionary strategies to optimize reward functions (Ma et al., 2023). While promising, these methods often require significant domain expertise, large amounts of data, or computationally expensive processes.

LLMs for Reward Functions Recent work has explored using large language models (LLMs) to generate reward functions based on natural language descriptions (Ma et al., 2023; Yu et al., 2023; Xie et al., 2023). Our approach builds upon this line of work by leveraging LLMs as stochastic reward generators but introduces a more efficient online reward selection and policy optimization process.

In a related but distinct setting, there have been efforts to directly utilize language and vision models as reward models (Rocamonde et al., 2023; Klissarov et al., 2023; Kwon et al., 2023). Rocamonde et al. (2023) employs CLIP similarity as a state-only reward model. Motif (Klissarov et al., 2023) first constructs a pair-wise preferences dataset using a large language model (LLM), learns a preference-based intrinsic reward model with the Bradley-Terry model, and then uses this reward model to train a reinforcement learning agent.

Online Model Selection The problem of model selection in sequential decision-making has been of interest in recent years (Agarwal et al., 2017; Foster et al., 2019; Pacchiano et al., 2020; Lee et al., 2021). Our work is closely related to the Pacchiano et al. (2023), which tackles the problem of selecting the best model or hyperparameters in a sequential fashion. In our setting, the set of models comprises various reward functions and their corresponding policies.

7 Conclusion

This paper formalizes the reward design problem for RL and proposes ORSO, a novel algorithm that automates the design of dense reward functions. ORSO operates in two phases: (1) generating a set of candidate reward functions, and (2) efficiently identifying the best reward function from this set via online reward selection and policy optimization.

We provide a theoretical analysis of ORSO's regret when using the D^3RB algorithm for selection. Moreover, we evaluate our approach on a set of continuous control tasks, demonstrating ORSO's consistent ability to find high-quality reward functions. Notably, ORSO is substantially more sample and compute efficient than prior evolutionary approaches.

Our work opens up several exciting avenues for future research. Although ORSO can design effective dense reward functions efficiently, it still requires a task reward or success metric in the code, which can be non-trivial to design. Future work could focus on eliminating the need for specifying ground truth rewards by leveraging methods that directly translate language instructions into evaluators, such as using vision-language models. By automating reward design, we believe ORSO can accelerate progress toward agents that can easily acquire complex behaviors from high-level task specifications.

References

Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

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

Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.

Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. The nonstochastic multi-armed bandit problem. *SIAM journal on computing*, 32(1):48–77, 2002.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

Dylan J Foster, Akshay Krishnamurthy, and Haipeng Luo. Model selection for contextual bandits. *Advances in Neural Information Processing Systems*, 32, 2019.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and JoÃĢo GM AraÃšjo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

Martin Klissarov, Pierluca D'Oro, Shagun Sodhani, Roberta Raileanu, Pierre-Luc Bacon, Pascal Vincent, Amy Zhang, and Mikael Henaff. Motif: Intrinsic motivation from artificial intelligence feedback. *arXiv preprint arXiv:2310.00166*, 2023.

Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. Reward design with language models. *arXiv preprint arXiv:2303.00001*, 2023.

Jonathan Lee, Aldo Pacchiano, Vidya Muthukumar, Weihao Kong, and Emma Brunskill. Online model selection for reinforcement learning with function approximation. In *International Conference on Artificial Intelligence and Statistics*, pages 3340–3348. PMLR, 2021.

Minghuan Liu, Zixuan Chen, Xuxin Cheng, Yandong Ji, Ruihan Yang, and Xiaolong Wang. Visual whole-body control for legged loco-manipulation. *arXiv preprint arXiv:2403.16967*, 2024.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*, 2023.

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

Gabriel B Margolis and Pulkit Agrawal. Walk these ways: Tuning robot control for generalization with multiplicity of behavior. In *Conference on Robot Learning*, pages 22–31. PMLR, 2023.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287, 1999.

Aldo Pacchiano, My Phan, Yasin Abbasi Yadkori, Anup Rao, Julian Zimmert, Tor Lattimore, and Csaba Szepesvari. Model selection in contextual stochastic bandit problems. *Advances in Neural Information Processing Systems*, 33:10328–10337, 2020.

Aldo Pacchiano, Christoph Dann, and Claudio Gentile. Data-driven regret balancing for online model selection in bandits. *arXiv preprint arXiv:2306.02869*, 2023.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Juan Rocamonde, Victoriano Montesinos, Elvis Nava, Ethan Perez, and David Lindner. Vision-language models are zero-shot reward models for reinforcement learning. *arXiv preprint arXiv:2310.12921*, 2023.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Satinder Singh, Richard L Lewis, and Andrew G Barto. Where do rewards come from. In *Proceedings of the annual conference of the cognitive science society*, pages 2601–2606. Cognitive Science Society, 2009.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Tianbao Xie, Siheng Zhao, Chen Henry Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2reward: Automated dense reward function generation for reinforcement learning. *arXiv preprint arXiv:2309.11489*, 2023.

Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, et al. Language to rewards for robotic skill synthesis. *arXiv preprint arXiv:2306.08647*, 2023.

Chen Bo Calvin Zhang and Giorgia Ramponi. Hip-rl: Hallucinated inputs for preference-based reinforcement learning in continuous domains. In *ICML 2023 Workshop The Many Facets of Preference-Based Learning*, 2023.

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

8 Orso with Doubling Data-Driven Regret Balancing

Here, we present the complete ORSO algorithm with Doubling Data-Driven Regret Balancing (D^{3}RB) as the model selection algorithm. In our experiments, we use $d_{\min} = 1, \delta = 0.1$, and $c = 0.1$.

Algorithm 2 ORSO with D^{3}RB: Online Reward Selection and Policy Optimization with D^{3}RB

```
Require: MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \tilde{r}, \rho_0)$, algorithm $\mathfrak{A}$, generator $G$, minimum regret coefficients $d_{\min}$,
    failure probability $\delta$
 1: Sample $K$ reward functions $\mathcal{R}^K = \{r^1, \ldots, r^K\} \sim G$
 2: Initialize $K$ policies $\Pi = \{\pi^1, \ldots, \pi^K\}$
 3: Initialize balancing potentials $\phi_1^i = d_{\min}$ for all $i \in [K]$
 4: Initialize regret coefficients $\widehat{d}_0^i = d_{\min}$ for add $i \in [K]$
 5: Initialize counts $n_0^i = 0$ and total values $\widehat{u}_0^i = 0$ for all $i \in [K]$
 6: for $t = 1, 2, \ldots, T$ do
 7:     Select a base learner $i_t \in [K] \in \arg\min_{i \in [K]} \phi_t^i$
 8:     Construct MDP $\mathcal{M}_{i_t} = (\mathcal{S}, \mathcal{A}, P, r^{i_t}, \rho_0)$
 9:     Update $\pi^{i_t} \leftarrow \mathfrak{A}(\mathcal{M}_{i_t}, N, \pi^{i_t})$ and receive reward $r_t \leftarrow \texttt{Eval}(\pi^{i_t})$
10:     // Update necessary variables
11:     Set $n_t^i = n_{t-1}^i, \widehat{u}_t^i = \widehat{u}_{t-1}^i, \widehat{d}_t^i = \widehat{d}_{t-1}^i$, and $\phi_{t+1}^i = \phi_t^i$ for all $i \in [K] \setminus \{i_t\}$
12:     Update statistics for current learner $n_t^{i_t} = n_{t-1}^{i_t} + 1$ and $\widehat{u}_t^{i_t} = \widehat{u}_{t-1}^{i_t} + r_t$
13:     Perform misspecification test
```

$$\frac{\widehat{u}_t^{i_t}}{n_t^{i_t}} + \frac{\widehat{d}_{t-1}^{i_t}\sqrt{n_t^{i_t}}}{n_t^{i_t}} + c\sqrt{\frac{\ln \frac{K \ln n_t^{i_t}}{\delta}}{n_t^{i_t}}} < \max_{j \in [K]} \frac{\widehat{u}_t^j}{n_t^j} - c\sqrt{\frac{\ln \frac{K \ln n_t^j}{\delta}}{n_t^j}} \tag{5}$$

```
14:     if test is triggered then
15:         Double the regret coefficient $\widehat{d}_t^{i_1} \leftarrow 2\widehat{d}_{t-1}^{i_t}$
16:     else
17:         Keep the regret coefficient unchanged $\widehat{d}_t^{i_1} \leftarrow \widehat{d}_{t-1}^{i_t}$
18:     end if
19:     Update the balancing potential $\phi_{t+1}^{i_t} \leftarrow \widehat{d}_t^{i_t}\sqrt{n_t^{i_t}}$
20: end for
21: // Best policy and reward function under the true reward function
22: return $\pi, r = \arg\max_{i \in [K]}(\texttt{Eval}(\pi^i))$
```

9 Proof of Lemma 4.3

In this section, we present the complete proof of Lemma 4.3. We will start by showing that when Assumption 4.2 holds, then with probability at least $1-\delta$, the estimated regret coefficient of learner $i_\star$ will never double provided that $d_{\min} \geq c$, where c is the confidence multiplier in D^{3}RB.

Lemma 9.1 (Non-doubling regret coefficient)**.** *When $\mathcal{E}$ holds, and algorithm D^{3}RB is in use*

$$\widehat{d}_t^{i_\star} = d_{\min} \quad \textit{and} \quad n_T^i \leq n_T^{i_\star} + 1 \quad \textit{for all } i \in [K] \tag{6}$$

for all $t \in \mathbb{N}$.

Proof. In order to show this result it is sufficient to show that when $\mathcal{E}$ holds, algorithm $i_\star$ does not undergo any doubling event. Doubling of the regret coefficients only happens when the misspecification test triggers for algorithm $i_\star$.

We will show this by induction. Let us assume $\widehat{d}_{t-1}^{i_\star} = d_{\min}$ and that $i_t = i_\star$. When $\mathcal{E}$ holds, the left-hand side (LHS) of D^{3}RB's misspecification test satisfies

$$\begin{aligned}
\frac{\widehat{u}_t^{i_t}}{n_t^{i_t}} + \frac{\widehat{d}_{t-1}^{i_t}\sqrt{n_t^{i_t}}}{n_t^{i_t}} + c\sqrt{\frac{\ln\frac{K\ln n_t^{i_t}}{\delta}}{n_t^{i_t}}} &= \frac{\widehat{u}_t^{i_\star}}{n_t^{i_\star}} + \frac{\widehat{d}_{t-1}^{i_\star}\sqrt{n_t^{i_\star}}}{n_t^{i_\star}} + c\sqrt{\frac{\ln\frac{K\ln n_t^{i_\star}}{\delta}}{n_t^{i_\star}}} \\
&\geq \frac{u_t^{i_\star}}{n_t^{i_\star}} + \frac{\widehat{d}_{t-1}^{i_\star}\sqrt{n_t^{i_\star}}}{n_t^{i_\star}} \\
&\overset{(i)}{=} \frac{u_t^{i_\star}}{n_t^{i_\star}} + \frac{d_{\min}\sqrt{n_t^{i_\star}}}{n_t^{i_\star}}
\end{aligned} \tag{7}$$

where (i) holds because by the induction hypothesis $\widehat{d}_{t-1}^{i_\star} = d_{\min}$. We will now show that $n_t^{i_\star} \geq n_t^j$ for all $j \in [K]$. Since by the inductive hypothesis $\widehat{d}_\ell^{i_\star} = d_{\min}$ for all $\ell \leq t-1$, the potential $\phi_\ell^{i_\star} = d_{\min}\sqrt{n_{\ell-1}^{i_\star}}$ for all $\ell \leq t$.

For $i \in [K]$ let $t(i)$ be the last time – before time t – algorithm i was played. For $i \neq i_\star$ we have $t(i) < t$. Since i was selected at time $t(i)$, by definition of the potentials,

$$\widehat{d}_{t(i)-1}^{i_\star}\sqrt{n_{t(i)-1}^{i_\star}} = d_{\min}\sqrt{n_{t(i)-1}^{i_\star}} \geq \widehat{d}_{t(i)-1}^{i}\sqrt{n_{t(i)-1}^{i}} \geq d_{\min}\sqrt{n_{t(i)-1}^{i}}$$

so that $n_{t(i)-1}^{i_\star} \geq n_{t(i)-1}^i$. Since both $n_t^{i_\star} = n_{t(i)-1}^{i_\star} + 1$ and $n_t^i = n_{t(i)-1}^i + 1$ we conclude that $n_t^{i_\star} \geq n_t^i$.

We now turn our attention to the right-hand side (RHS) of D^{3}RB's misspecification test. When $\mathcal{E}$ holds, the RHS of D^{3}RB's misspecification test satisfies,

$$\begin{aligned}
\max_{j\in[K]} \frac{\widehat{u}_t^j}{n_t^j} - c\sqrt{\frac{\ln\frac{K\ln n_t^j}{\delta}}{n_t^j}} &\leq \max_{j\in[K]} \frac{u_t^j}{n_t^j} \\
&\overset{(i)}{\leq} \max_{j\in[K]} \frac{u_{(n_t^j)}^{i_\star}}{n_t^j} \\
&\overset{(ii)}{\leq} \frac{u_t^{i_\star}}{n_t^{i_\star}}
\end{aligned} \tag{8}$$

where inequalities (i) and (ii) hold because of Assumption 4.2. Combining inequalities 7 and 8 we conclude the misspecification test of algorithm D^{3}RB will not trigger. This finalizes the proof. □

We are now ready to prove the regret bound on the base learners given in Lemma 4.3

Lemma 4.3. *Under event $\mathcal{E}$ and Assumption 4.2, with probability $1-\delta$, the regret of all base learners i is bounded in all rounds T as*

$$\sum_{t=1}^{n_T^i} \operatorname{reg}(\pi_{(t)}^i) \leq 6 d_T^{i_\star} \sqrt{n_T^{i_\star}+1} + 5c\sqrt{(n_T^{i_\star}+1)\ln\frac{K\ln T}{\delta}}, \tag{4}$$

where $d_T^{i_\star} = d^{i_\star}_{(n_T^{i_\star})}$.

Proof. Consider a fixed base learner i and time horizon T, and let $t \leq T$ be the last round where i was played but the misspecification test did not trigger. If no such round exists, then set $t = 0$. By Corollary 9.1 in Pacchiano et al. (2023), i can be played at most $1 + \log_2 \frac{\bar{d}_T^i}{d_{\min}}$ times between t and T and thus

$$\sum_{k=1}^{n_T^i} \operatorname{reg}(\pi_{(k)}^i) \leq \sum_{k=1}^{n_t^i} \operatorname{reg}(\pi_{(k)}^i) + 1 + \log_2 \frac{\bar{d}_T^i}{d_{\min}}.$$

If $t = 0$, then the desired statement holds. Thus, it remains to bound the first term in the RHS above when $t > 0$. Since $i = i_t$ and the test did not trigger we have, for any base learner j with $n_t^j > 0$,

$$\begin{aligned}
\sum_{k=1}^{n_t^i} \operatorname{reg}(\pi_{(k)}^i) &= n_t^i v^\star - u_t^i && \text{(definition of regret)}\\
&= n_t^i v^\star - \frac{n_t^i}{n_t^j} u_t^j + \frac{n_t^i}{n_t^j} u_t^j - u_t^i\\
&= \frac{n_t^i}{n_t^j}\left(n_t^j v^\star - u_t^j\right) + \frac{n_t^i}{n_t^j} u_t^j - u_t^i\\
&= \frac{n_t^i}{n_t^j}\left(\sum_{k=1}^{n_t^j} \operatorname{reg}(\pi_{(k)}^j)\right) + \frac{n_t^i}{n_t^j} u_t^j - u_t^i && \text{(definition of regret)}\\
&\leq \frac{n_t^i}{n_t^j}\left(d_t^j \sqrt{n_t^j}\right) + \frac{n_t^i}{n_t^j} u_t^j - u_t^i && \text{(definition of regret rate)}\\
&\leq \sqrt{\frac{n_t^i}{n_t^j}}\, d_t^j \sqrt{n_t^i} + \frac{n_t^i}{n_t^j} u_t^j - u_t^i.
\end{aligned}$$

We now focus on $j = i_\star$ and use the balancing condition in Lemma 9.2 in Pacchiano et al. (2023) to bound the first factor $\sqrt{n_t^i / n_t^{i_\star}}$. This condition gives that $\phi_{t+1}^i \leq 3\phi_{t+1}^{i_\star}$. Since both $n_t^{i_\star} > 0$ and $n_t^i > 0$, we have $\phi_{t+1}^i = \widehat{d}_t^i \sqrt{n_t^i}$ and $\phi_{t+1}^{i_\star} = \widehat{d}_t^{i_\star} \sqrt{n_t^{i_\star}}$. Thus, we get

$$\sqrt{\frac{n_t^i}{n_t^{i_\star}}} = \sqrt{\frac{n_t^i}{n_t^{i_\star}}} \cdot \frac{\widehat{d}_t^i}{\widehat{d}_t^{i_\star}} \cdot \frac{\widehat{d}_t^{i_\star}}{\widehat{d}_t^i} = \frac{\phi_{t+1}^i}{\phi_{t+1}^{i_\star}} \cdot \frac{\widehat{d}_t^{i_\star}}{\widehat{d}_t^i} \leq 3 \frac{\widehat{d}_t^{i_\star}}{\widehat{d}_t^i} \leq 3, \tag{9}$$

where the last inequality holds because of Lemma 9.1 and because $\widehat{d}_t^i \geq d_{\min}$.

Plugging this back into the expression above and setting $j = i_\star$, we have

$$\sum_{k=1}^{n_t^i} \operatorname{reg}(\pi_{(k)}^i) \leq 3 d_t^{i_\star} \sqrt{n_t^i} + \frac{n_t^i}{n_t^{i_\star}} u_t^{i_\star} - u_t^i.$$

To bound the last two terms, we use the fact that the misspecification test did not trigger in round t. Therefore,

$$\begin{aligned}
u_t^i &\geq \widehat{u}_t^i - c\sqrt{n_t^i \ln \frac{K \ln n_t^i}{\delta}} && \text{(event } \mathcal{E}\text{)} \\
&= n_t^i \left(\frac{\widehat{u}_t^i}{n_t^i} + c\sqrt{\frac{\ln \frac{K \ln n_t^i}{\delta}}{n_t^i}} + \frac{\widehat{d}_t^i}{\sqrt{n_t^i}} \right) - 2c\sqrt{n_t^i \ln \frac{K \ln n_t^i}{\delta}} - \widehat{d}_t^i \sqrt{n_t^i} \\
&\geq \frac{n_t^i}{n_t^{i_\star}} \widehat{u}_t^{i_\star} - \sqrt{\frac{n_t^i}{n_t^{i_\star}}} c\sqrt{n_t^i \ln \frac{K \ln n_t^{i_\star}}{\delta}} - 2c\sqrt{n_t^i \ln \frac{K \ln n_t^i}{\delta}} - \widehat{d}_t^i \sqrt{n_t^i}. && \text{(test not triggered)}
\end{aligned}$$

Rearranging terms and plugging this expression in the bound above gives

$$\begin{aligned}
\sum_{k=1}^{n_t^i} \operatorname{reg}(\pi_{(k)}^i) &\leq 3d_t^{i_\star} \sqrt{n_t^i} + \sqrt{\frac{n_t^i}{n_t^{i_\star}}} c\sqrt{n_t^i \ln \frac{K \ln n_t^{i_\star}}{\delta}} + 2c\sqrt{n_t^i \ln \frac{K \ln n_t^i}{\delta}} + \widehat{d}_t^i \sqrt{n_t^i} \\
&\leq 3d_t^{i_\star} \sqrt{n_t^i} + 3c\sqrt{n_t^i \ln \frac{K \ln n_t^{i_\star}}{\delta}} + 2c\sqrt{n_t^i \ln \frac{K \ln n_t^i}{\delta}} + \widehat{d}_t^i \sqrt{n_t^i} && \text{(Equation (9))} \\
&\leq 3d_t^{i_\star} \sqrt{n_t^i} + 3c\sqrt{n_t^i \ln \frac{K \ln n_t^{i_\star}}{\delta}} + 2c\sqrt{n_t^i \ln \frac{K \ln n_t^i}{\delta}} + 3\widehat{d}_t^{i_\star} \sqrt{n_t^{i_\star}} && \text{(Equation (9))} \\
&\leq 3d_t^{i_\star} \sqrt{n_t^i} + 3\widehat{d}_t^{i_\star} \sqrt{n_t^{i_\star}} + 5c\sqrt{n_t^i \ln \frac{K \ln t}{\delta}} && (\max(n_t^i, n_t^{i_\star}) \leq t) \\
&\overset{(i)}{\leq} 3d_t^{i_\star} \sqrt{n_t^i} + 3d_{\min} \sqrt{n_t^{i_\star}} + 5c\sqrt{n_t^i \ln \frac{K \ln t}{\delta}} && \text{(Lemma 9.1)}
\end{aligned}$$

where inequality (i) follows from Lemma 9.1. Finally, Lemma 9.1 also implies $n_t^i \leq n_t^{i_\star} + 1$ and since $d_{\min} \leq d_t^{i_\star}$,

$$\sum_{k=1}^{n_t^i} \operatorname{reg}(\pi_{(k)}^i) \leq 6d_t^{i_\star} \sqrt{n_t^{i_\star} + 1} + 5c\sqrt{(n_t^{i_\star} + 1) \ln \frac{K \ln t}{\delta}}.$$

The statement follows by setting $t = T$. □

10 Task Score Functions

Table 2 reports the task score functions used as ground truth reward functions to evaluate the generated rewards. The functions are the same as the ones used in Ma et al. (2023).

Table 2: Task success metric definitions.

Environment	Success Metric
Ant	`current_distance - previous_distance`
Humanoid	`current_distance - previous_distance`
Allegro Hand	$\sum \mathbb{1}\{$`rotation_distance < 0.1`$\}$
Shadow Hand	$\sum \mathbb{1}\{$`rotation_distance < 0.1`$\}$
Cartpole	$\sum \mathbb{1}\{$`agent is alive`$\}$
Ball Balance	$\sum \mathbb{1}\{$`agent is alive`$\}$

11 Rejection Sampling Mechanism

In this section, we provide additional details on the rejection sampling scheme used by ORSO. Given criteria ϕ to be satisfied, rejection sampling repeats the following steps until we have sampled the desired number, K, of valid reward functions:

1. Sample a candidate reward function $r \sim G$
2. Chek if criteria ϕ is satisfied
   - If $\phi(r)$ is satisfied, add r to the set of candidate reward functions
   - If $\phi(r)$ is not satisfied, reject r

In our practical implementation, checking if criteria ϕ are satisfied consists of instantiating an environment with the generated reward function, running a random policy on it, and checking the values produced by the reward function. If the environment cannot be instantiated or if the values returned by the reward function are $\pm\infty$ or `NaN`, the reward function is rejected. It is worth noting that this only guarantees a higher probability of a valid reward function code as the policy used to evaluate the function is random and the optimization process used during the training of an RL algorithm could still induce undesirable values.

12 Additioanl Experimental Results

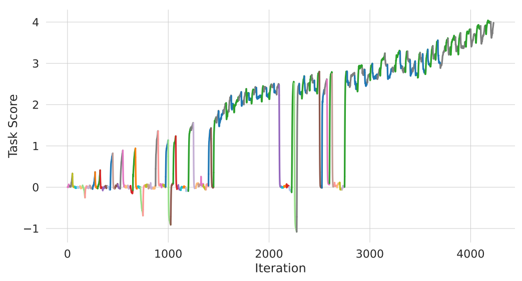
In Figures 4 to 9, we report additional curves ORSO and EUREKA.

13 MAB Algorithms and Hyperparameters

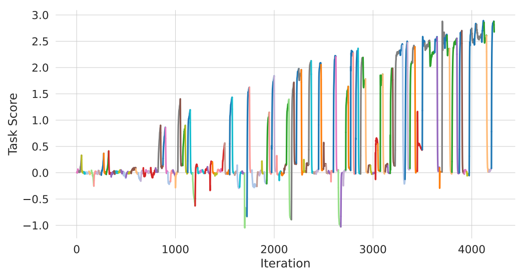
In this section, we present the pseudocode for all reward selection algorithms used in our experiments with their associated hyperparameters in Table 3. For locomotion tasks, we use $F = 25$ and for manipulation, we use $F = 100$ iterations.

Algorithm 3 Explore-then-Commit

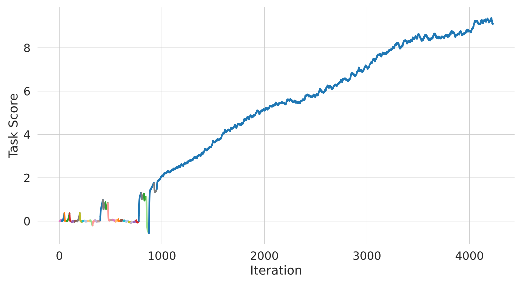
```
Require: Number of arms $K$, total time $T$, exploration phase length $T_0$
1: Initialize counts $n_0^i = 0$ and total values $\widehat{u}_0^i = 0$ for all $i \in [K]$
2: // Explore
3: for $t = 1, \dots, T_0$ do
4:     Select arm $i_t = (t \mod K) + 1$
5:     Play arm $i_t$ and observe reward $r_t$
6:     Set $n_t^i = n_{t-1}^i$, and $\widehat{u}_t^i = \widehat{u}_{t-1}^i$ for all $i \in [K] \setminus \{i_t\}$
7:     Update statistics for current learner $n_t^{i_t} = n_{t-1}^{i_t} + 1$ and $\widehat{u}_t^{i_t} = \widehat{u}_{t-1}^{i_t} + r_t$
8: end for
9: // Commit
10: $i_\star = \arg\max_i (u_t^i / n_t^i)$
11: for $t = T_0 + 1$ to $T$ do
12:     Play arm $i_\star$ and observe reward $r_t$
13:     Set $n_t^i = n_{t-1}^i$, and $\widehat{u}_t^i = \widehat{u}_{t-1}^i$ for all $i \in [K] \setminus \{i_t\}$
14:     Update statistics for current learner $n_t^{i_t} = n_{t-1}^{i_t} + 1$ and $\widehat{u}_t^{i_t} = \widehat{u}_{t-1}^{i_t} + r_t$
15: end for
```



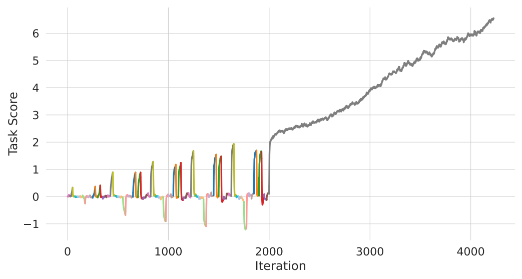
(a) Ant D^{3}RB



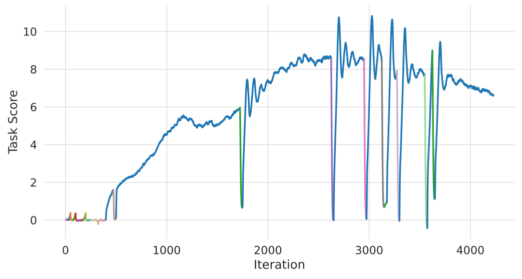
(b) Ant Exp3



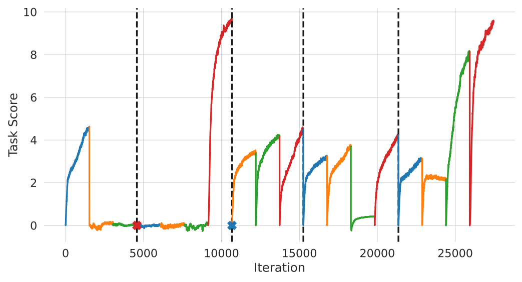
(c) Ant UCB



(d) Ant ETC



(e) Ant Epsilon-Greedy



(f) Ant Eureka

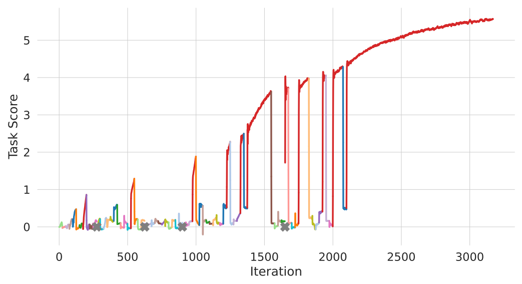
Figure 4: Ant

Algorithm 4 ε-Greedy

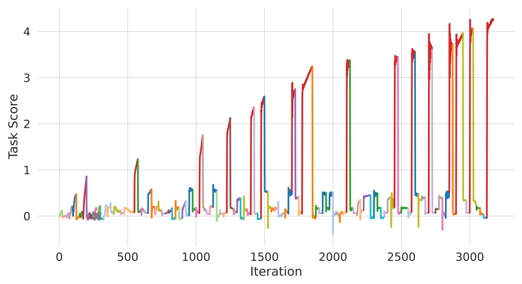
```
Require: Number of arms $K$, total time $T$, exploration probability $\varepsilon$
1: Initialize counts $n_0^i = 0$ and total values $\widehat{u}_0^i = 0$ for all $i \in [K]$
2: for $t = 1, \dots, T$ do
3:     With probability $\varepsilon$, select a random arm $i_t$, else $i_t = \arg\max_i(\widehat{u}_t^i / n_t^i)$
4:     Play arm $i_t$ and observe reward $r_t$
5:     Set $n_t^i = n_{t-1}^i$, and $\widehat{u}_t^i = \widehat{u}_{t-1}^i$ for all $i \in [K] \setminus \{i_t\}$
6:     Update statistics for current learner $n_t^{i_t} = n_{t-1}^{i_t} + 1$ and $\widehat{u}_t^{i_t} = \widehat{u}_{t-1}^{i_t} + r_t$
7: end for
```

Table 3: Hyperparameters for MAB Algorithms

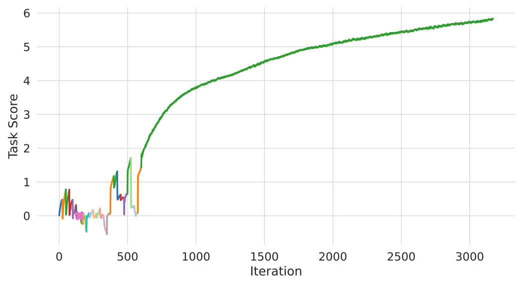
Algorithm	Parameter	Value
Explore-then-Commit	T_0	$5 \cdot K$
Epsilon-Greedy	ε	0.1
UCB	c	1.0
Exp3	η	0.1



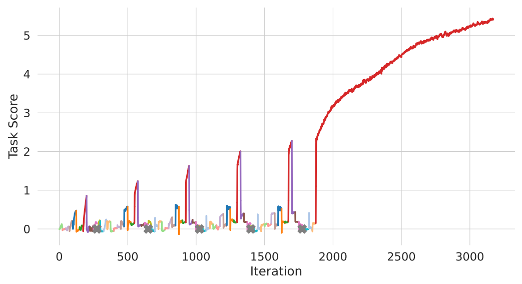
(a) Humanoid D^{3}RB



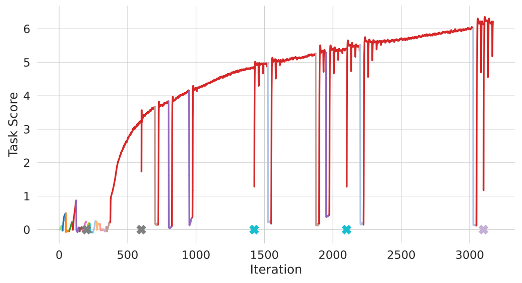
(b) Humanoid Exp3



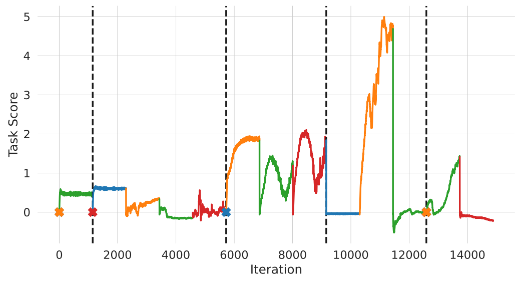
(c) Humanoid UCB



(d) Humanoid ETC



(e) Humanoid Epsilon-Greedy

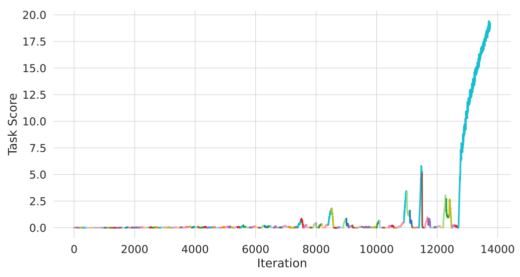


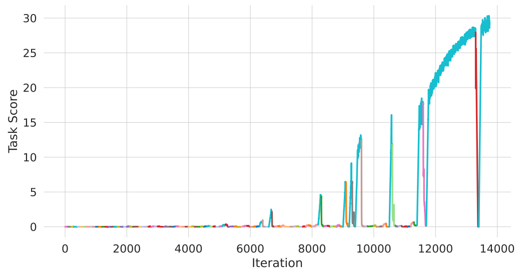
(f) Humanoid Eureka

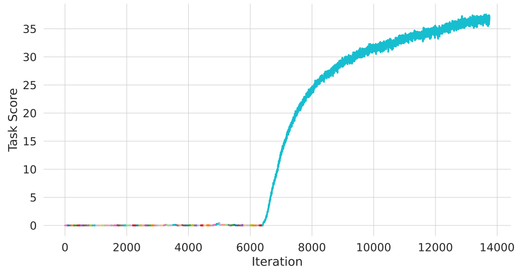
Figure 5: Humanoid

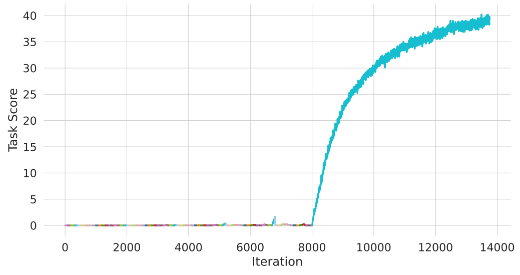
Algorithm 5 UCB (Upper Confidence Bound)

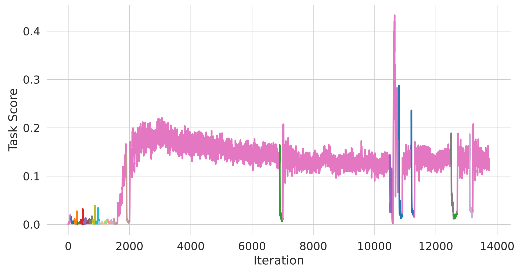
```
Require: Number of arms $K$, total time $T$, confidence multiplier $c$
 1: Initialize counts $n_0^i = 0$ and total values $\widehat{u}_0^i = 0$ for all $i \in [K]$
 2: for $t = 1, \dots, K$ do
 3:     Select arm $i_t = t$
 4:     Play arm $i_t$ and observe reward $r_t$
 5:     Update statistics for current learner $n_t^{i_t} = n_{t-1}^{i_t} + 1$ and $\widehat{u}_t^{i_t} = \widehat{u}_{t-1}^{i_t} + r_t$
 6: end for
 7: for $t = K + 1, \dots, T$ do
 8:     Select arm $i_t = \arg\max_i(\widehat{u}_t^i / n_t^i + c\sqrt{2 \ln t / n_t^i})$
 9:     Play arm $i_t$ and observe reward $r_t$
10:     Set $n_t^i = n_{t-1}^i$, and $\widehat{u}_t^i = \widehat{u}_{t-1}^i$ for all $i \in [K] \setminus \{i_t\}$
11:     Update statistics for current learner $n_t^{i_t} = n_{t-1}^{i_t} + 1$ and $\widehat{u}_t^{i_t} = \widehat{u}_{t-1}^{i_t} + r_t$
12: end for
```


(a) Allegro Hand D^{3}RB


(b) Allegro Hand Exp3


(c) Allegro Hand UCB


(d) Allegro Hand ETC


(e) Allegro Hand Epsilon-Greedy

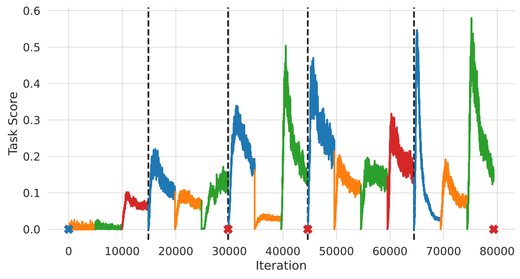
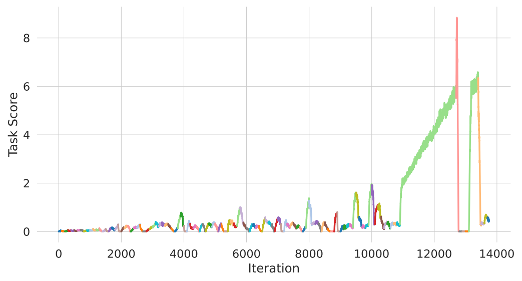

(f) Allegro Hand Eureka

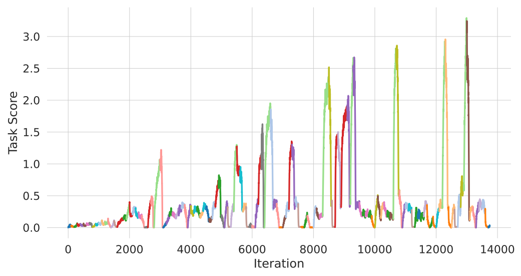
Figure 6: Allegro Hand

Algorithm 6 Exp3 (Exponential-weight algorithm for Exploration and Exploitation)

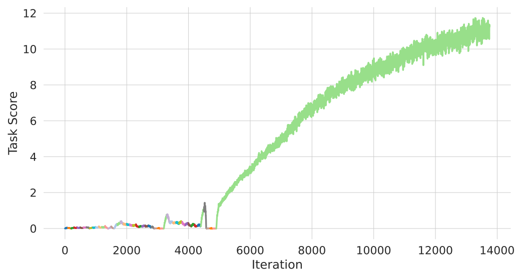
```
Require: Number of arms $K$, total time $T$, learning rate $\eta$
1: Initialize weights $w_0^i = 1$ and probabilities $p_0^i = 1/K$ for all $i \in [K]$
2: for $t = 1, \dots, T$ do
3:     Select arm $i_t$ according to distribution $P_t = [p_t^1, \dots, p_t^K]$
4:     Play arm $i_t$ and observe reward $r_t$
5:     Estimate reward $\hat{r}_t = r_t / p_t^{i_t}$
6:     Update weight $w_t^{i_t} = w_{t-1}^{i_t} \exp(\eta \hat{r}_t / K)$
7:     Update probabilities
           $p_t^i = (1-\eta)\frac{w_t^i}{\sum_{j=1}^K w_t^j} + \frac{\eta}{K} \qquad \text{for all } i \in [K]$
8: end for
```



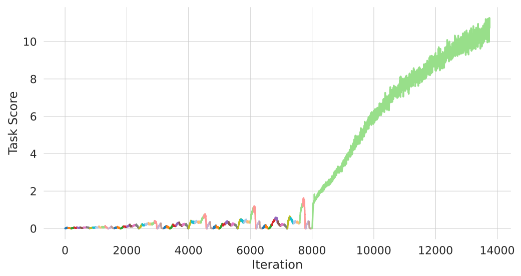
(a) Shadow Hand D^{3}RB



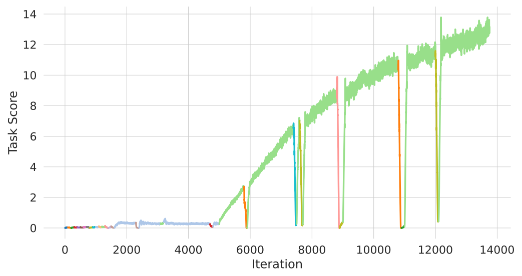
(b) Shadow Hand Exp3



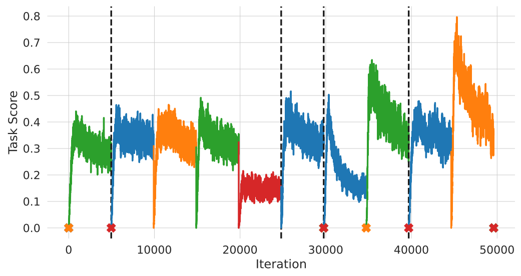
(c) Shadow Hand UCB



(d) Shadow Hand ETC

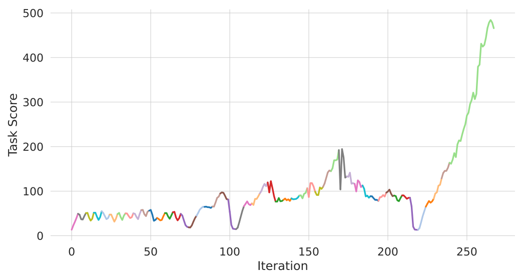


(e) Shadow Hand Epsilon-Greedy

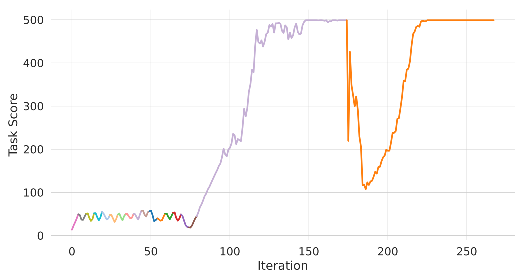


(f) Shadow Hand Eureka

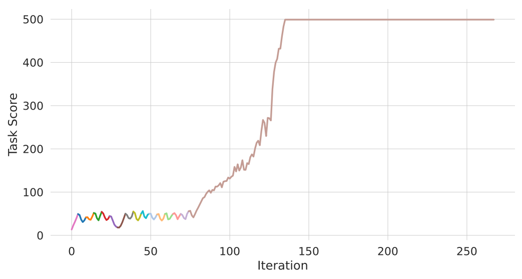
Figure 7: Shadow Hand



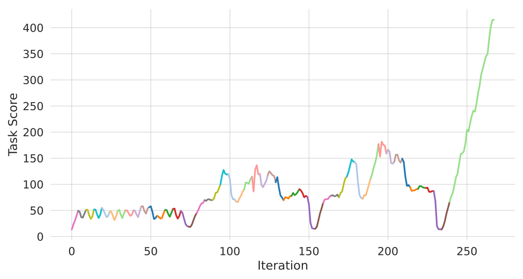
(a) Cartpole D^{3}RB



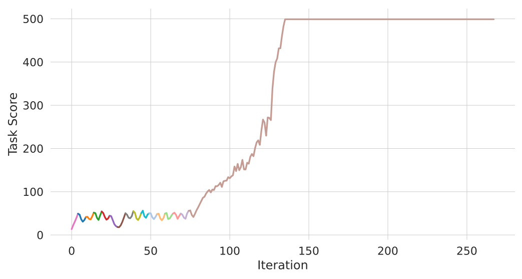
(b) Cartpole Exp3



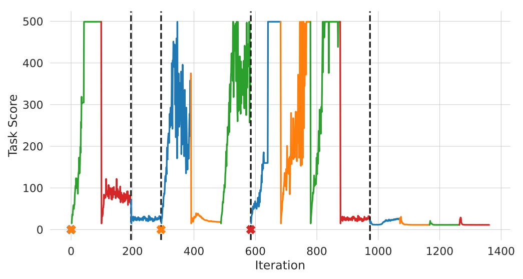
(c) Cartpole UCB



(d) Cartpole ETC

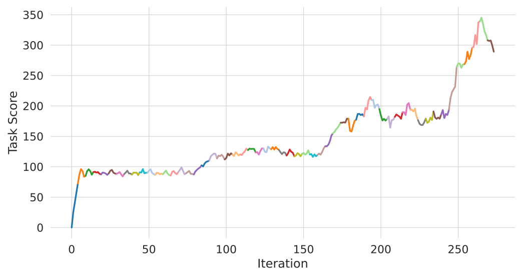


(e) Cartpole Epsilon-Greedy

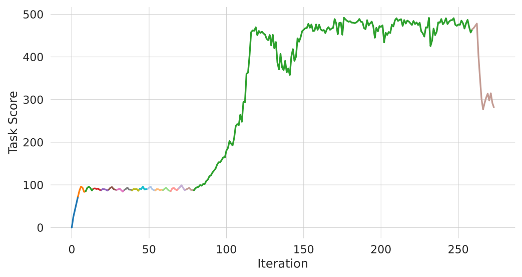


(f) Cartpole Eureka

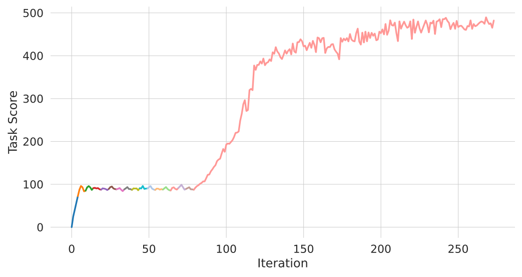
Figure 8: Performance of different methods on the Cartpole environment.



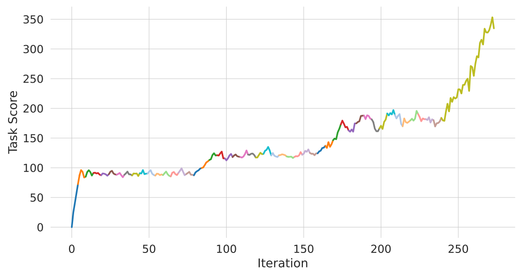
(a) Ball Balance D^{3}RB



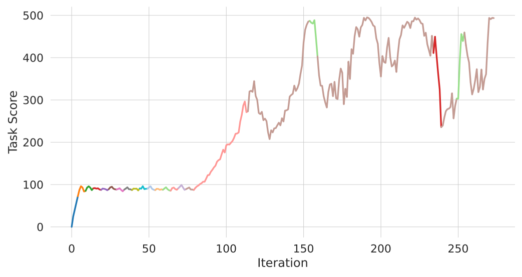
(b) Ball Balance Exp3



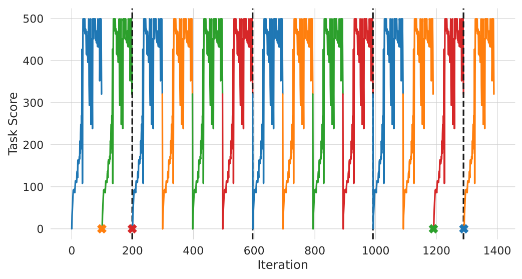
(c) Ball Balance UCB



(d) Ball Balance ETC



(e) Ball Balance Epsilon-Greedy



(f) Ball Balance Eureka

Figure 9: Performance of different methods on the Ball Balance environment.