Redeeming Intrinsic Rewards via Constrained Optimization

Eric Chen[*], **Zhang-Wei Hong** [*†], **Joni Pajarinen**[‡] **& Pulkit Agrawal**[†§]
Improbable AI Lab, Massachusetts Institute of Technology
MIT-IBM Watson AI Lab[†] Aalto University[‡]
NSF AI Institute for AI and Fundamental Interactions (IAIFI)[§]

Abstract

State-of-the-art reinforcement learning (RL) algorithms typically use random sampling (e.g., ϵ-greedy) for exploration, but this method fails in hard exploration tasks like Montezuma's Revenge. To address the challenge of exploration, prior works incentivize the agent to visit novel states using an exploration bonus (also called an intrinsic reward or curiosity). Such methods can lead to excellent results on hard exploration tasks but can suffer from intrinsic reward bias and underperform when compared to an agent trained using only task rewards. This performance decrease occurs when an agent seeks out intrinsic rewards and performs unnecessary exploration even when sufficient task reward is available. This inconsistency in performance across tasks prevents the widespread use of intrinsic rewards with RL algorithms. We propose a principled constrained policy optimization procedure that automatically tunes the importance of the intrinsic reward: it suppresses the intrinsic reward when exploration is unnecessary and increases it when exploration is required. This results in superior exploration that does not require manual tuning to balance the intrinsic reward against the task reward. Consistent performance gains across sixty-one ATARI games validate our claim. The code is available at https://github.com/Improbable-AI/eipo.

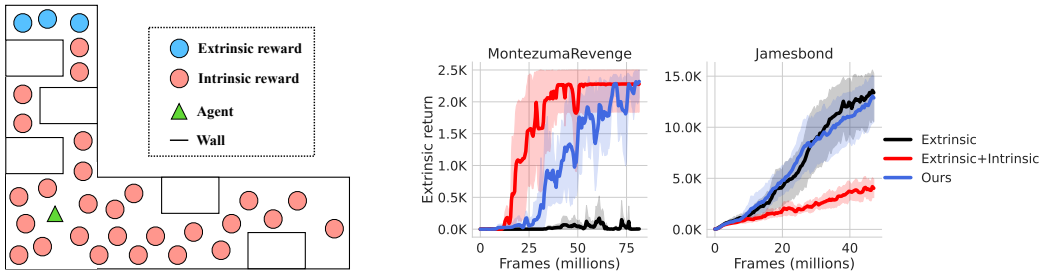
1 Introduction

The goal of reinforcement learning (RL) [1] is to find a mapping from states to actions (i.e., a policy) that maximizes reward. At every learning iteration, an agent is faced with a question: has the maximum possible reward been achieved? In many practical problems, the maximum achievable reward is unknown. Even when the maximum achievable reward is known, if the current policy is sub-optimal then the agent is faced with another question: would spending time improving its current strategy lead to higher rewards (*exploitation*), or should it attempt a different strategy in the hope of discovering potentially higher reward (*exploration*)? Pre-mature *exploitation* is akin to getting stuck in a *local-optima* and precludes the agent from exploring. Too much exploration on the other hand can be distracting, and prevent the agent from perfecting a good strategy. Resolving the *exploration-exploitation* dilemma [1] is therefore essential for data/time efficient policy learning.

In simple decision making problems where actions do not affect the state (e.g. bandits or contextual bandits [2]), provably optimal algorithms for balancing exploration against exploitation are known [3, 2]. However, in the general settings where RL is used, such algorithms are unknown. In the absence of methods that work well in both theory and practice, state-of-the-art RL algorithms rely on heuristic exploration strategies such as adding noise to actions or random sampling of sub-optimal actions (e.g., ϵ-greedy). However, such strategies fail in sparse reward scenarios where infrequent rewards hinder policy improvement. One such task is the notorious ATARI game, *Montezuma's Revenge* [4].

[*] denotes equal contribution. Correspondence to pulkitag@mit.edu.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).



(a) (b)

Figure 1: **(a)** At the start of training all locations are novel for the agent (green triangle), and therefore the pink circles representing *intrinsic rewards* are evenly distributed across the map. The blue circles represent sources of *extrinsic reward* or task-reward. Here *intrinsic* rewards can distract the agent, as the sum of *extrinsic* and *intrinsic* rewards can be increased by moving along the bottom corridor. **(b)** This type of distraction is a possible reason why an intrinsic reward method does not consistently outperform a method trained using only extrinsic rewards across ATARI games. Intrinsic rewards help in some games where the task or extrinsic reward is sparse (e.g., *Montezuma's revenge*), but hurt in other games such as *James Bond*. Our proposed method, EIPO, intelligently uses intrinsic rewards when needed and consistently matches the best-performing algorithm amongst extrinsic and extrinsic+intrinsic methods.

Sparse reward problems can be solved by supplementing the task reward (or extrinsic reward r_E) with a dense exploration bonus (or intrinsic reward r_I) generated by the agent itself [4–8]. Intrinsic rewards encourage the agent to visit novel states, which increases the chance of encountering states with task reward. Many prior works [4, 8, 9] show that jointly optimizing for intrinsic and extrinsic reward (i.e., $r_E + \lambda r_I$, where $\lambda \geq 0$ is a hyperparameter) instead of only optimizing for extrinsic reward (i.e., $\lambda = 0$) improves performance on sparse reward tasks such as *Montezuma's revenge* [4, 9, 10].

However, a recent study found that using intrinsic rewards does not consistently outperform simple exploration strategies such as ϵ-greedy across ATARI games [11]. This is because the mixed objective ($r_E + \lambda r_I$) is *biased* for $|\lambda| > 0$, and optimizing it does not necessarily yield the optimal policy with respect to the extrinsic reward alone [12]. Fig. 1a illustrates this problem using a toy example. Here the green triangle is the agent and the blue/pink circles denote the location of extrinsic and intrinsic rewards, respectively. At the start of training, all states are novel and provide a source of intrinsic reward (i.e., pink circles). This makes accumulating intrinsic rewards easy, which the agent may exploit to optimize its objective of maximizing the sum of intrinsic and extrinsic rewards. However, such optimization can result in a local maxima: the agent might move rightwards along the bottom corridor, essentially distracting the agent from the blue task rewards at the top. In this example, since it is not hard to find the task reward, better performance is obtained if only the extrinsic reward ($\lambda = 0$) is maximized. The trouble, however, is that in most environments one doesn't know a priori how to optimally trade off intrinsic and extrinsic rewards (i.e., choose λ).

A common practice is to conduct an extensive hyperparameter search to find the best λ, as different values of λ are best suited for different tasks (see Fig. 4). Furthermore, as the agent progresses on a task, the best exploration-exploitation trade-off can vary, and a constant λ may not be optimal throughout training. In initial stages of training exploration might be preferred. Once the agent is able to obtain some task reward, it might prefer exploiting these rewards instead of exploring further. The exact dynamics of the exploration-exploitation trade-off is task-dependent, and per-task tuning is tedious, undesirable, and often computationally infeasible. Consequently, prior works use a fixed λ during training, which our experiments reveal is sub-optimal.

We present an optimization strategy that alleviates the need to manually tune the relative importance of extrinsic and intrinsic rewards as training progresses. Our method leverages the *bias* of intrinsic rewards when it is useful for exploration and mitigates this bias when it does not help accumulate higher extrinsic rewards. This is achieved using an *extrinsic optimality constraint* that forces the extrinsic rewards earned after optimizing the mixed objective to be equal to the extrinsic rewards accumulated by the *optimal* policy that maximizes extrinsic rewards only. Enforcing the *extrinsic optimality constraint* in general settings is intractable because the optimal extrinsic reward is unknown. We devise a practical algorithm called **Extrinsic-Intrinsic Policy Optimization (EIPO)**, which uses an approximation to solve this constrained optimization problem (Section 3).

While in principle we can apply EIPO to any intrinsic reward method, we evaluate performance using state-of-the-art *random network distillation (RND)* [9]. Fig. 1b presents *teaser* results on two ATARI games: (i) *Montezuma's Revenge* - where joint optimization with RND (red) substantially

outperforms a PPO policy [13] optimized with only extrinsic rewards (black); (ii) *James Bond* - where PPO substantially outperforms RND. These results reinforce the notion that bias introduced by intrinsic rewards helps in some games, but hurts in others. Our algorithm EIPO (blue) matches the best algorithm in both games, showing that it can leverage intrinsic rewards as needed. Results across 61 ATARI games reinforce this finding. Additionally, in some games EIPO outperforms multiple strong baselines with and without intrinsic rewards, indicating that our method can not only mitigate the potential performance decreases caused by intrinsic reward bias, but can also improve performance beyond the current state-of-the-art.

2 Preliminaries

We consider a discrete-time Markov Decision Process (MDP) consisting of a state space $\mathcal{S}$, an action space $\mathcal{A}$, and an extrinsic reward function $\mathcal{R}_E : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. We distinguish the extrinsic and intrinsic reward components by E and I, respectively. The extrinsic reward function $\mathcal{R}_E$ refers to the actual task objective (e.g., game score). The agent starts from an initial state s_0 sampled from the initial state distribution $\rho_0 : \mathcal{S} \to \mathbb{R}$. At each timestep t, the agent perceives a state s_t from the environment, takes action a_t sampled from the policy π, receives extrinsic reward $r_t^E = \mathcal{R}_E(s_t, a_t)$, and moves to the next state s_{t+1} according to the transition function $\mathcal{T}(s_{t+1}|s_t, a_t)$. The agent's goal is to use interactions with the environment to find the optimal policy π such that the extrinsic objective value $J_E(\pi)$ is maximized:

$$\max_{\pi} J_E(\pi), \text{ where } J_E(\pi) = \mathbb{E}_{s_0,a_0,\cdots\sim\pi}\Big[\sum_{t=0}^{\infty}\gamma^t r_t^E\Big] \text{ (Extrinsic objective)}, \tag{1}$$
$$s_0 \sim \rho_0, a_t \sim \pi(a|s_t), s_{t+1} \sim \mathcal{T}(s_{t+1}|s_t, a_t)\ \forall t > 0$$

where γ denotes a discount factor. For brevity, we abbreviate $\mathbb{E}_{s_0,a_0,\cdots\sim\pi}\Big[.\Big]$ as $\mathbb{E}_{\pi}\Big[.\Big]$ unless specified.

Intrinsic reward based exploration strategies[4, 8, 9] attempt to encourage exploration by providing "intrinsic rewards" (or "exploration bonuses") that incentivize the agent to visit unseen states. Using the intrinsic reward function $\mathcal{R}_I : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, the optimization objective becomes:

$$\max_{\pi\in\Pi} J_{E+I}(\pi), \text{ where } J_{E+I}(\pi) = \mathbb{E}_{\pi}\big[\sum_{t=0}^{\infty}\gamma^t(r_t^E + \lambda r_t^I)\big] \text{ (Mixed objective)}, \tag{2}$$

where λ denotes the intrinsic reward scaling coefficient. We abbreviate the intrinsic reward at timestep t as $r_t^I = \mathcal{R}_I(s_t, a_t)$. State-of-the-art intrinsic reward based exploration strategies [9, 14] often optimize the objective in Eq. 2 using Proximal Policy Optimization (PPO) [13].

3 Mitigating the Bias of Intrinsic Rewards

Simply maximizing the sum of intrinsic and extrinsic rewards does not guarantee a policy that also maximizes extrinsic rewards: $\arg\max_{\pi_{E+I}\in\Pi} J_{E+I}(\pi_{E+I}) \neq \arg\max_{\pi_E\in\Pi} J_E(\pi_E)$. At convergence the optimal policy $\pi^*_{E+I} = \arg\max_{\pi_{E+I}} J_{E+I}(\pi_{E+I})$ could be suboptimal w.r.t. J_E, which measures the agent's task performance. Because the agent's performance is measured using extrinsic reward only, we propose enforcing an *extrinsic optimality constraint* that ensures the optimal "mixed" policy $\pi^*_{E+I} = \arg\max_{\pi_{E+I}} J_{E+I}(\pi_{E+I})$ leads to as much extrinsic reward as the optimal "extrinsic" policy $\pi^*_E = \arg\max_{\pi_E\in\Pi} J_E(\pi_E)$. The resulting optimization objective is:

$$\max_{\pi_{E+I}\in\Pi} J_{E+I}(\pi_{E+I}) \tag{3}$$
$$\text{subject to } J_E(\pi_{E+I}) - \max_{\pi_E} J_E(\pi_E) = 0 \qquad \text{(Extrinsic optimality constraint)}.$$

Solving this optimization problem can be viewed as proposing a policy π_{E+I} that maximizes J_{E+I}, and then checking if the proposed π_{E+I} is feasible given the extrinsic optimality constraint.

The constrained objective is difficult to optimize because evaluating the extrinsic optimality constraint requires $J_E(\pi^*_E)$, which is unknown. To solve this optimization problem, we transform it into an *unconstrained min-max* optimization problem using Lagrangian duality (Section 3.1). We then describe an iterative algorithm for solving the min-max optimization by alternating between minimization and maximization in Section 3.2, and we present implementation details in Section 3.3.

3.1 The Dual Objective: Unconstrained Min-Max Optimization Problem

The Lagrangian dual problem for the primal constrained optimization problem in Eq. 3 is:

$$\min_{\alpha \in \mathbb{R}^+} \Big[\max_{\pi_{E+I} \in \Pi} J_{E+I}(\pi_{E+I}) + \alpha \big(J_E(\pi_{E+I}) - \max_{\pi_E \in \Pi} J_E(\pi_E) \big) \Big], \tag{4}$$

where $\alpha \in \mathbb{R}^+$ is the Lagrangian multiplier. We rewrite Eq. 4 by merging $J_{E+I}(\pi)$ and $J_E(\pi)$:

$$J^{\alpha}_{E+I}(\pi_{E+I}) \coloneqq J_{E+I}(\pi_{E+I}) + \alpha J_E(\pi_{E+I}) = \mathbb{E}_{\pi_{E+I}} \Big[\sum_{t=0}^{\infty} \gamma^t \big[(1+\alpha) r^E(s_t, a_t) + r^I(s_t, a_t) \big] \Big].$$

The re-written objective provides an intuitive interpretation of α: larger values correspond to increasing the impetus on extrinsic rewards (i.e., exploitation). Substituting $J^{\alpha}_{E+I}(\pi_{E+I})$ into Eq. 4 and rearranging terms yields the following min-max problem:

$$\min_{\alpha \in \mathbb{R}^+} \Big[\max_{\pi_{E+I} \in \Pi} \min_{\pi_E \in \Pi} J^{\alpha}_{E+I}(\pi_{E+I}) - \alpha J_E(\pi_E) \Big]. \tag{5}$$

3.2 Algorithm for Optimizing the Dual Objective

We now describe an algorithm for solving π_E, π_{E+I}, and α in each of the sub-problems in Eq. 5.

Extrinsic policy π_E (min-stage). π_E is optimized via the minimization sub-problem, which can be re-written as a maximization problem:

$$\min_{\pi_E \in \Pi} J^{\alpha}_{E+I}(\pi_{E+I}) - \alpha J_E(\pi_E) \quad \rightarrow \quad \max_{\pi_E \in \Pi} \alpha J_E(\pi_E) - J^{\alpha}_{E+I}(\pi_{E+I}) \tag{6}$$

The main challenge is that evaluating the objectives $J^{\alpha}_{E+I}(\pi_{E+I})$ and $J_E(\pi_E)$ requires sampling trajectories from both policies π_{E+I} and π_E. If one were to use an on-policy optimization method such as PPO, this would require sampling trajectories from two separate policies at each iteration during training, which would be data inefficient. Instead, if we assume that the two policies are similar, then we can leverage results from prior work to use the trajectories from one policy (π_{E+I}) to *approximate* the return of the other policy (π_E) [15, 16, 13].

First, using the performance difference lemma from [15], the objective $\alpha J_E(\pi_E) - J^{\alpha}_{E+I}(\pi_{E+I})$ can be re-written (see Appendix A.1.3 for detailed derivation):

$$\alpha J_E(\pi_E) - J^{\alpha}_{E+I}(\pi_{E+I}) = \mathbb{E}_{\pi_E} \Big[\sum_{t=0}^{\infty} \gamma^t U^{\pi_{E+I}}_{\min}(s_t, a_t) \Big]. \tag{7}$$

$$\text{where } U^{\pi_{E+I}}_{\min}(s_t, a_t) \coloneqq \alpha r^E_t + \gamma V^{\pi_{E+I}}_{E+I}(s_{t+1}) - V^{\pi_{E+I}}_{E+I}(s_t),$$

$$V^{\pi_{E+I}}_{E+I}(s_t) \coloneqq \mathbb{E}_{\pi_{E+I}} \Big[\sum_{t=0}^{\infty} \gamma^t (r^E_t + r^I_t) | s_0 = s_t \Big]$$

Next, under the *similarity assumption*, a lower bound to the objective in Eq. 7 can be obtained [13]:

$$\begin{aligned} \mathbb{E}_{\pi_E} \Big[\sum_{t=0}^{\infty} \gamma^t U^{\pi_{E+I}}_{\min}(s_t, a_t) \Big] \geq \mathbb{E}_{\pi_{E+I}} \Big[\sum_{t=0}^{\infty} \gamma^t \min \Big\{ & \frac{\pi_E(a_t|s_t)}{\pi_{E+I}(a_t|s_t)} U^{\pi_{E+I}}_{\min}(s_t, a_t), \\ & \text{clip} \left(\frac{\pi_E(a_t|s_t)}{\pi_{E+I}(a_t|s_t)}, 1-\epsilon, 1+\epsilon \right) U^{\pi_{E+I}}_{\min}(s_t, a_t) \Big\} \Big] \end{aligned} \tag{8}$$

where $\epsilon \in [0, 1]$ denotes a threshold. Intuitively, this clipped objective (Eq. 8) penalizes the policy π_E that behaves differently from π_{E+I} because overly large or small $\frac{\pi_E(a_t|s_t)}{\pi_{E+I}(a_t|s_t)}$ terms are clipped. More details are provided in Appendix A.2.

Mixed policy π_{E+I} (max-stage). The sub-problem of solving for π_{E+I} is posed as

$$\max_{\pi_{E+I} \in \Pi} J^{\alpha}_{E+I}(\pi_{E+I}) - \alpha J_E(\pi_E). \tag{9}$$

We again rely on the approximation from [13] to derive a lower bound surrogate objective for $J^{\alpha}_{E+I}(\pi_{E+I}) - \alpha J_E(\pi_E)$ as follows (see Appendix A.1.2 for details):

$$J^{\alpha}_{E+I}(\pi_{E+I}) - \alpha J_E(\pi_E) \geq \mathbb{E}_{\pi_E}\Big[\sum_{t=0}^{\infty}\gamma^t \min\Big\{\frac{\pi_{E+I}(a_t|s_t)}{\pi_E(a_t|s_t)}U^{\pi_E}_{\max}(s_t,a_t), \\ \text{clip}\left(\frac{\pi_{E+I}(a_t|s_t)}{\pi_E(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right)U^{\pi_E}_{\max}(s_t,a_t)\Big\}\Big], \tag{10}$$

where $U^{\pi_E}_{\max}$ and $V^{\pi_E}_{E}$ are defined as follows:

$$U^{\pi_E}_{\max}(s_t,a_t) \coloneqq (1+\alpha)r^E_t + r^I_t + \gamma\alpha V^{\pi_E}_E(s_{t+1}) - \alpha V^{\pi_E}_E(s_t)$$
$$V^{\pi_E}_E(s_t) \coloneqq \mathbb{E}_{\pi_E}\Big[\sum_{t=0}^{\infty}\gamma^t r^E_t | s_0 = s_t\Big].$$

Lagrangian multiplier α. We solve for α by using gradient descent on the surrogate objective derived above. Let $g(\alpha) := \max_{\pi_{E+I}\in\Pi}\min_{\pi_E\in\Pi} J^{\alpha}_{E+I}(\pi_{E+I}) - \alpha J_E(\pi_E)$. Therefore, $\nabla g(\alpha) = J_E(\pi_{E+I}) - J_E(\pi_E)$. We approximate $\nabla g(\alpha)$ using the lower bound surrogate objective:

$$J_E(\pi_{E+I}) - J_E(\pi_E) \geq L(\pi_E,\pi_{E+I}) = \mathbb{E}_{\pi_E}\Big[\sum_{t=0}^{\infty}\gamma^t \min\Big\{\frac{\pi_{E+I}(a_t|s_t)}{\pi_E(a_t|s_t)}A^{\pi_E}(s_t,a_t), \\ \text{clip}\left(\frac{\pi_{E+I}(a_t|s_t)}{\pi_E(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right)A^{\pi_E}(s_t,a_t)\Big\}\Big], \tag{11}$$

where $A^{\pi_E}(s_t,a_t) = r^E_t + \gamma V^{\pi_E}_E(s_{t+1}) - V^{\pi_E}_E(s_t)$ is the advantage of taking action a_t at state s_t, and then following π_E for subsequent steps. We update α using a step size β (Appendix A.1.4):

$$\alpha \leftarrow \alpha - \beta L(\pi_E,\pi_{E+I}). \tag{12}$$

Unlike prior works that use a fix trade off between extrinsic and intrinsic rewards, optimizing α during training allows our method to automatically and dynamically tune the trade off.

3.3 Implementation

Min-max alternation schedule. We use an iterative optimization scheme that alternates between solving for π_E by minimizing the objective in Eq. 6, and solving for π_{E+I} by maximizing the objective in Eq. 9 (`max_stage`). We found that switching the optimization every iteration hinders performance which is hypothesize is a result of insufficient training of each policy. Instead, we switch between the two stages when the objective value does not improve anymore. Let $J[i]$ be the objective value $J^{\alpha}_{E+I}(\pi_{E+I}) - \alpha J_E(\pi_E)$ at iteration i. The switching rule is:

$$\begin{cases} J[i] - J[i-1] \leq 0 \implies \texttt{max\_stage} \leftarrow \texttt{False}, & \text{if } \texttt{max\_stage = True} \\ J[i] - J[i-1] \geq 0 \implies \texttt{max\_stage} \leftarrow \texttt{True}, & \text{if } \texttt{max\_stage = False}, \end{cases}$$

where `max_stage` is a binary variable indicating whether the current stage is the max-stage. α is only updated when the max-stage is done using Eq. 12.

Parameter sharing. The policies π_{E+I} and π_E are parametrized by two separate multi-layer perceptrons (MLP) with a shared CNN backbone. The value functions $V^{\pi_{E+I}}_{E+I}$ and $V^{\pi_E}_E$ are represented in the same fashion, and share a CNN backbone with the policy networks. When working with image inputs (e.g., ATARI), sharing the convolutional neural network (CNN) backbone between π_E and π_{E+I} helps save memory, which is important when using GPUs (in our case, an NVIDIA RTX 3090Ti). If both policies share a backbone however, maximizing the objective (Eq. 9) with respect to π_{E+I} might interfere with $J_E(\pi_E)$, and impede the performance of π_E. Similarly, minimizing the objective (Eq. 6) with respect to π_E might modify J^{α}_{E+I} and degrade the performance of π_{E+I}.

Interference can be avoided by introducing auxiliary objectives that prevent decreases in $J_E(\pi_E)$ and J^{α}_{E+I} when optimizing for π_{E+I}, π_E, respectively. For example, when updating π_{E+I} in the max-stage, $\max_{\pi_{E+I}} J^{\alpha}_{E+I}(\pi_{E+I}) - \alpha J_E(\pi_E)$, the auxiliary objective $\max_{\pi_E} J_E(\pi_E)$ can prevent updates in the shared CNN backbone from decreasing $J(\pi_E)$. We incorporate the auxiliary objective without additional hyperparameters (Appendix A.2.3). The overall objective is in Appendix A.2.4.

Algorithm 1 Extrinsic-Intrinsic Policy Optimization (EIPO)

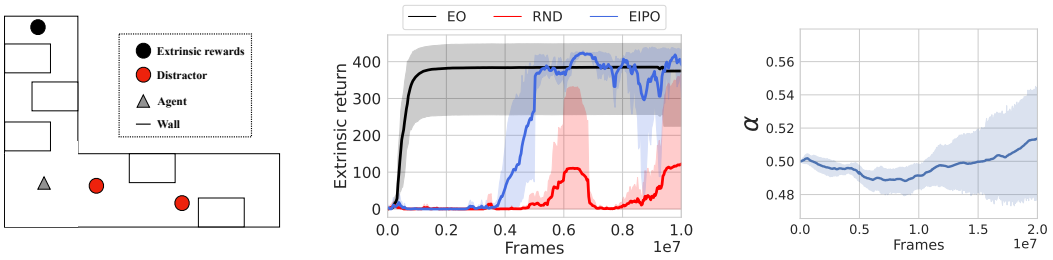
Initialize policies π_{E+I} and π_E, `max_stage[0]` $\leftarrow$ `False`, and $J[0] \leftarrow 0$
for $i = 1 \cdots$ **do** ▷ i denotes iteration index
  if `max_stage[i - 1]` **then** ▷ Max-stage: rollout by π_E and update π_{E+I}
    Collect trajectories τ_E using π_E and compute $U^{\pi_E}_{\text{max}}(s_t, a_t)\ \forall (s_t, a_t) \in \tau_E$
    Update π_{E+I} by Eq. 10 and π_E by auxiliary objective (Section 3.3)
    $J[i] \leftarrow J^{\alpha}_{E+I}(\pi_{E+I}) - \alpha J_E(\pi_E)$
    `max_stage[i]` $\leftarrow J[i] - J[i-1] \leq 0$
  else ▷ Min-stage: rollout by π_{E+I} and update π_E
    Collect trajectories τ_{E+I} using π_{E+I} and compute $U^{\pi_{E+I}}_{\text{min}}(s_t, a_t)\ \forall (s_t, a_t) \in \tau_{E+I}$
    Update π_E by Eq. 8 and π_{E+I} by auxiliary objective (Section 3.3)
    $J[i] \leftarrow J^{\alpha}_{E+I}(\pi_{E+I}) - \alpha J_E(\pi_E)$
    `max_stage[i]` $\leftarrow J[i] - J[i-1] \geq 0$
  end if
  if `max_stage[i - 1] = True` and `max_stage[i] = False` **then**
    Update α (Eq. 32) ▷ Update when the max-stage is done
  end if
end for

Extrinsic-Intrinsic Policy Optimization (EIPO) - the policy optimization algorithm we introduce to solve Eq. 5, is outlined in Algorithm 1. Pseudo-code can be found in Algorithm 2, and full implementation details including hyperparameters can be found in Appendix A.2.

4 Experiments

While EIPO is agnostic to the choice of intrinsic rewards, we mainly experiment with RND [9] because it is the state-of-the-art intrinsic reward method. EIPO implemented with RND is termed *EIPO-RND* and compared against several baselines below. All policies are learned using PPO [13].

- **EO (Extrinsic only)**: The policy is trained using extrinsic rewards only: $\pi^* = \arg\max_{\pi \in \Pi} J_E(\pi)$.
- **RND (Random Network Distillation) [9]**: The policy is trained using the sum of extrinsic rewards (r^E_t) and RND intrinsic rewards (λr^I_t): $\pi^* = \arg\max_{\pi \in \Pi} J_{E+I}(\pi)$. For ATARI experiments, we chose a single value of λ that works best across games. Other methods below also use this λ.
- **EN (Ext-norm-RND)**: We found that a variant of RND where the extrinsic rewards are normalized using running mean and standard deviation (Appendix A.2.2) outperforms the original RND implementation, especially in games where RND performs worse than EO. Our method EIPO and other baselines below are therefore implemented using *Ext-norm-RND*.
- **DY (Decay-RND)**: Instead of having a fixed trade-off between exploration and exploitation throughout training, dynamically adjusting exploration vs. exploitation can lead to better performance. Without theoretical results describing how to make such adjustments, a commonly used heuristic is to gradually transition from exploration to exploitation. One example is ϵ-greedy exploration [17], where ϵ is decayed over the course of training. Similarly, we propose a variant of *Ext-norm-RND* where intrinsic rewards are progressively scaled down to eliminate exploration bias over time. Intrinsic rewards r^I_t are scaled by $\lambda(i)$, where i denotes the iteration number. The objective function turns into $J_{E+I} = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t (r^E_t + \lambda(i) r^I_t)\right]$, where $\lambda(i)$ is defined as $\lambda(i) = \text{clip}(\frac{i}{I}(\lambda_{\text{max}} - \lambda_{\text{min}}), \lambda_{\text{min}}, \lambda_{\text{max}})$. Here, λ_{max} and λ_{min} denote the predefined maximum and minimum $\lambda(i)$, and I is the iteration after which decay is fixed. We follow the linear decay schedule used to decay ϵ-greedy exploration in DQN [17].
- **DC (Decoupled-RND) [18]**: The primary assumption in EIPO is that π_{E+I} and π_E are similar (Section 3.2). A recent work used this assumption in a different way to balance intrinsic and extrinsic rewards [18]: they regularize the mixed policy to stay close to the extrinsic policy by minimizing the Kullback–Leibler (KL) divergence $\text{D}_{\text{KL}}(\pi_E || \pi_{E+I})$. However, they do not impose the *extrinsic-optimality constraint*, which is a key component in EIPO. Therefore, comparing against this method called *Decoupled-RND* (*DC*) [18] will help separate the gains in performance



(a) (b) (c)

Figure 2: **(a)** 2D navigation task with a similar distribution of extrinsic and intrinsic rewards as Figure 1a. The gray triangle is the agent. The **black** dot is the goal providing extrinsic reward. Red dots are randomly placed along the bottom corridor at the start of every episode. Due to the novelty of their locations, these dots serve as a source of intrinsic rewards. **(b)** RND is distracted by intrinsic rewards and fails to match EO optimized only with extrinsic rewards. Without intrinsic rewards, EO is inferior to our method EIPO in discovering a good path to the black goal. This result indicates that EIPO is not distracted by intrinsic rewards and uses them as necessary to improve performance. **(c)** α controls the importance of extrinsic optimality constraint. It decreases until the extrinsic return starts to rise between 0.5 and 1.0 million frames. Afterwards, α increases, showing that our method emphasizes intrinsic rewards at the start of training, and capitalizes on extrinsic rewards once found.

that come from making the *similarity assumption* versus the importance of *extrinsic-optimality constraint*. We adapted *DC* to make use of *Ext-norm-RND* intrinsic rewards:

$$\pi^*_{E+I} = \arg\max_{\pi_{E+I}} \mathbb{E}_{\pi_{E+I}}\Big[\sum_{t=0}^{\infty} \gamma^t (r^E_t + r^I_t) - \mathrm{D_{KL}}(\pi_E || \pi_{E+I})\Big], \quad \pi^*_E = \arg\max_{\pi_E} \mathbb{E}_{\pi_I}\Big[\sum_{t=0}^{\infty} \gamma^t r^E_t\Big].$$

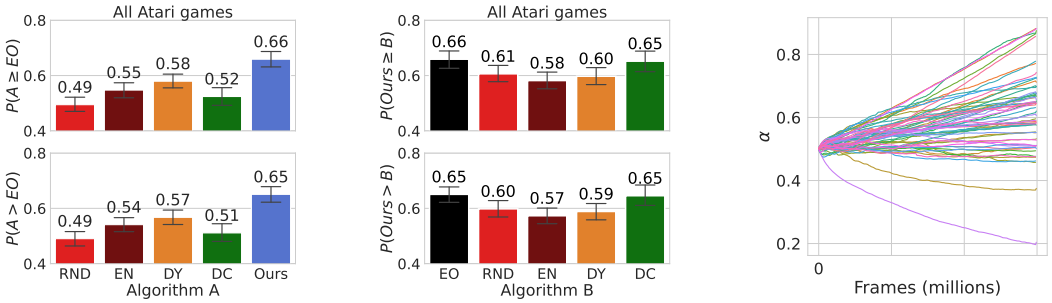
4.1 Illustrative example

We exemplify the problem of intrinsic reward bias using a simple 2D navigation environment (Fig. 2a) implemented using the Pycolab game engine [19]. The gray sprite denotes the agent, which observes a 5×5 pixel window view of its surroundings. If it reaches the (**black**) location at the top of the map, an extrinsic reward of +1 is provided. The red circles are randomly placed in the bottom corridor at the start of each episode. Because these circles randomly change location, they induce RND intrinsic rewards throughout training. Because intrinsic rewards along the bottom corridor are easier to obtain than the extrinsic reward, optimizing the mixed objective J_{E+I} can yield a policy that results in the agent exploring the bottom corridor (i.e., exploiting the intrinsic reward) without ever discovering the extrinsic reward at the top. Fig. 2b plots the evolution of the average extrinsic returns during training for EO, RND, and *EIPO-RND* across 5 random seeds. We find that *EIPO-RND* outperforms both RND and EO. RND gets distracted by the intrinsic rewards (red blocks) and is worse than the agent optimizing only the extrinsic reward (EO). EO performs slightly worse than *EIPO-RND*, possibly because in some runs the EO agent fails to reach the goal without the guidance of intrinsic rewards.

To understand why *EIPO-RND* performs better, we plot the evolution of the parameter α that trades-off exploration against exploitation during training (Fig. 2c). Lower values of α denote that the agent is prioritizing intrinsic rewards (exploration) over extrinsic rewards (exploitation; see Section 3.1). This plot shows that for the first $\sim 0.5M$ steps, the value of α decreases (Fig. 2c) indicating that the agent is prioritizing *exploration*. Once the agent finds the extrinsic reward (between $0.5M - 1M$ steps), the value of α stabilizes which indicates that further prioritization of *exploration* is unnecessary. After $\sim 1M$ steps the value of α increases as the agent prioritizes *exploitation*, and extrinsic return increases (Fig. 2b). These plots show that EIPO is able to dynamically trade-off exploration against exploitation during training. The dynamics of α during training also supports the intuition that EIPO transitions from exploration to exploitation over the course of training.

4.2 EIPO Redeems Intrinsic Rewards

To investigate if the benefits of EIPO carry over to more realistic settings, we conducted experiments on ATARI games [20], the de-facto benchmark for exploration methods [4, 11]. Our goal is not to maximize performance on ATARI, but to use ATARI as a proxy to anticipate performance of different RL algorithms on a new and potentially real-world task. Because ATARI games vary in the task objective and the difficulty of exploration, if an algorithm A *consistently* outperforms other algorithms (say B) on ATARI, it provides evidence that even for a new task whose exploration difficulty and task



(a) (b) (c)

Figure 3: **(a)** *EIPO-RND* (ours) has a higher probability of improvement $P(\textit{EIPO-RND} > \text{EO})$ over EO than all other baselines. It suggests *EIPO-RND* is more likely to attain a higher score than EO, compared to other methods. **(b)** Probability of improvement $P(\textit{EIPO-RND} > B) > 0.5 \; \forall B$ indicates that *EIPO-RND* performs strictly better than the baselines in the majority of trials. **(c)** Each colored curve denotes the evolution of α in one game using EIPO. The variance in α trajectories across games reveals that different exploration-exploitation dynamics is apt for different games.

objective is unknown, the same algorithm may fare better. If we can find such an algorithm, it eases the job of RL practitioners by mitigating the need for the current practice of cycling through different RL algorithms to find the one that works the best for the task at hand.

We used "probability of improvement" metric $P(A > B)$ with a 95%-confidence interval (see Appendix A.4.2; [21]) to judge if algorithm A consistently outperforms B across all ATARI games. If $P(A > B) > 0.5$, it indicates that algorithm A outperforms B on a majority of games (i.e., *consistent* improvement). Higher values of $P(A > B)$ means greater consistency in performance improvement of A over B. For the sake of completeness, we also report $P(A \geq B)$ which is a weaker condition of improvement measuring if algorithm A is at-par or better than algorithm B. These statistics are calculated using the median of extrinsic returns over the last hundred episodes for at least 5 random seeds for each method and game. More details are provided in Appendix A.4.

Comparing Exploration with Intrinsic Rewards v/s Only Optimizing Extrinsic Rewards Results in Fig. 3a show that $P(\text{RND} > \text{EO}) = 0.49$ with confidence interval $\{0.46, 0.52\}$, where EO denotes PPO optimized using extrinsic rewards only. These results indicate inconsistent performance gains of RND over EO, an observation also made by prior works [11]. *Ext-norm-RND* (*EN*) fares slightly better against EO suggesting that normalizing extrinsic rewards helps the joint optimization of intrinsic and extrinsic rewards. We find that $P(\textit{EIPO-RND} > EO) = 0.65$ with confidence interval $\{0.62, 0.67\}$, indicating that EIPO is able to successfully leverage intrinsic rewards for exploration and is likely to outperform EO on new tasks. Like EIPO, *Decoupled-RND* assumes that π_E and π_{E+I} are similar, but *Decoupled-RND* is not any better than *Ext-norm-RND* when compared against EO. This suggests that the similarity assumption on its own does not improve performance and that *extrinsic-optimality-constraint* is key to performance improvement.

EIPO *Strictly* Outperforms Baseline Methods Results in Fig. 3b show that $P(\textit{EIPO-RND} > B) > 0.5$ in a statistically rigorous manner for all baseline algorithms (B) across ATARI games. Experiments on the Open AI Gym MuJoCo benchmark follow the same trend: EIPO is either at-par or outperforms the best algorithm amongst EO and RND (see Appendix 10). These results show that EIPO is also applicable to tasks with continuous state and action space. The results in ATARI and MuJoCo benchmarks taken together provide strong evidence that given a new task, EIPO has highest chance of achieving the best performance.

4.3 Does EIPO Outperform RND Tuned with Hyper-parameter Search?

In results until now, the RND method used the single best intrinsic reward scaling coefficient λ across games. Our claim is that EIPO can automatically tune the weighting between intrinsic and extrinsic rewards. If this is indeed true, then EIPO should either match or outperform the best λ determined on a per-game basis through an exhaustive search over different values of λ. Since such hyper-parameter search is time consuming, for such an evaluation we selected a mix of representative games: a few where PPO optimized with only extrinsic rewards (EO) outperforms RND and other games where RND outperforms EO (e.g., *Venture* and *Montezeuma's Revenge*).

For each game we searched over multiple values of λ based on the relative difference in magnitudes between intrinsic (r^I) and extrinsic (r^E) rewards (see Appendix A.4.4 for details). The results

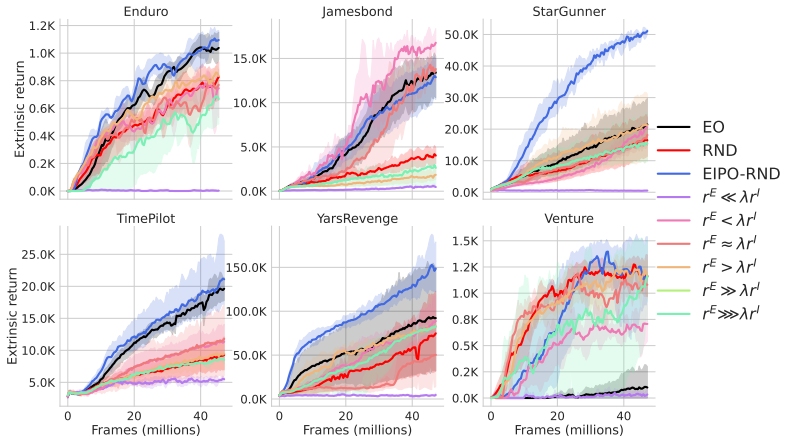


Figure 4: No choice of the intrinsic reward scaling coefficients λ consistently yield performance improvements over RND and EO, whereas our method *EIPO-RND* consistently outperforms RND and EO without any tuning across environments. It indicates that our method is less susceptible to λ.

summarized in Fig. 4 show that no choice of λ consistently improves RND performance. For instance, $r^E \approx \lambda r^I$ substantially improves RND in *Jamesbond*, yet deteriorates RND in *YarsRevenge*. Moreover, we found that with RND, the relationship between choice of λ and final performance is not intuitive. For example, *Jamesbond* is regarded as an easy exploration game [4]. Yet, the agent that uses high intrinsic reward (i.e., $r^E < \lambda r^I$) outperforms the agent with small intrinsic reward ($r^E \gg \lambda r^I$). EIPO overcomes these challenges and is able to automatically adjust the relative importance of intrinsic rewards. It outperforms RND despite game specific tuning of λ.

Dynamically adjustment of Exploration-Exploitation trade off during training improves performance If the performance gain of EIPO solely resulted from automatically determining a constant trade off between intrinsic and extrinsic rewards throughout training on a per-game basis, an exhaustive hyper-parameter search of λ should have matched the performance of *EIPO-RND*. The fact that *EIPO-RND* outperforms RND with different choices of λ leads us to hypothesize that adjusting the importance of intrinsic rewards over the course of training is also important to performance. This dynamic adjustment is controlled the value of α which is optimized during training (Section 3.2). Fig. 3c shows the trajectory of α during training across games. In most games, α increases over time indicating that the agent transitions from exploration to exploitation as the training progresses. Such decay in exploration is critical to performance: the algorithm, *Decay-RND* (DY), that implements this intuition by uniformly *decaying* the importance of intrinsic rewards outperforms RND (Fig. 5 and Fig. 3a). However, in some games α decreases over time indicating that more exploration is required in the end. An example is the game *Carnival* where *EIPO-RND* jumps in performance at the end of training which is accompanied by decrease in α (see Appendix A.8). These observations suggest that simple strategies that follow the same schedule for adjusting the balance between intrinsic and extrinsic rewards across tasks will perform worse than EIPO. The result that $P(\textit{EIPO-RND} > \textit{DY}) = 0.59$ reported in Fig. 3b confirms our belief and establishes that dynamic and per-game tuning of exploration-exploitation tradeoff is necessary for good performance.

4.4 Does EIPO improve over RND only in Easy Exploration Tasks?

The observation that *EIPO-RND* outperforms RND on most games might result from performance improvement on easy exploration games, which constitute the majority in ATARI, at the expense of worse performance on hard exploration games. To mitigate this possibility, we evaluated performance on games where RND outperforms EO as a proxy for hard exploration. The PPO-normalized scores [21] and 95% confidence intervals of various methods are reported in Table 1. *EIPO-RND* achieves slightly higher mean score than *Ext-norm-RND*, but lies within the confidence interval. This result suggests that EIPO not only mitigates the bias introduced by intrinsic rewards in easy exploration tasks, but either matches or improves performance in hard exploration tasks.

Table 1: *EIPO-RND* is either at-part or outperforms RND in the games where RND is better than EO (i.e., hard-exploration tasks).

Algorithm	PPO-normalized score Mean (CI)	
RND	384.57	(85.57, 756.69)
Ext-norm RND	427.08	(86.53, 851.52)
Decay-RND	383.83	(84.19, 753.17)
Decoupled-RND	1.54	(1.09, 2.12)
EIPO-RND	**435.56**	(109.45, 874.88)

5 Discussion

EIPO can be a drop-in replacement for any RL algorithm. Due to difficultly in tuning the intrinsic rewards and inconsistent performance across tasks, until now intrinsic reward based exploration was not part of the standard pipeline of state-of-the-art RL algorithms. *EIPO-RND* mitigates these challenges and also outperforms noisy networks [22], an exploration method that does not use intrinsic rewards, but is the best performing exploration strategy across ATARI games when used with Q-learning [17] (see Appendix A.9). The consistent performance gains of *EIPO-RND* on ATARI and MuJoCo benchmarks make it a strong contender for the defacto RL algorithm going forward. Though we performed experiments with PPO, the EIPO objective (Eq. 3.1) is agnostic to the particular choice of RL algorithm. As such, it can be used as a drop-in replacement in any RL pipeline.

Limitations. Across the 61 ATARI games, we use the same initial value of α and demonstrated that per-task tuning of importance of intrinsic rewards can be avoided by optimizing for α. However, it is possible that the initial value of α and learning step-size β depend on the choice of intrinsic reward function. To see if this the case, using the same α and β, we tested EIPO with another intrinsic reward metric - ICM [8]. The preliminary results in Appendix A.10 show that performance gains of *EIPO-ICM* over the baselines are less consistent than *EIPO-RND*. While one possibility is that *RND* is a better intrinsic reward than ICM, the other possibility is the intimate dependence between EIPO hyperparameters and the choice of intrinsic reward function which future work should investigate. Finally, the performance benefits of EIPO are limited by the how good is the underlying intrinsic reward function. Finding better intrinsic rewards remains an exciting avenue of research, which we hope can be accelerated by EIPO because it removes the need for manual tuning.

Potential applications to reward shaping. Intrinsic rewards can be thought of a particular instantiation of reward shaping [23] – the practice of incorporating auxiliary reward terms to boost overall task performance which is key to success of RL on many applications (see examples in [24, 25]). Balancing auxiliary rewards against task reward is tedious and often necessitates extensive hyper-parameter search. Because EIPO makes no intrinsic reward specific assumptions, its success in balancing the RND auxiliary reward suggests it might also be applicable in other reward shaping scenarios.

Acknowledgments

We thank members of the Improbable AI Lab for the helpful discussion and feedback. We are grateful to MIT Supercloud and the Lincoln Laboratory Supercomputing Center for providing HPC resources. The research was supported in part by the MIT-IBM Watson AI Lab, an AWS MLRA research grant, Google cloud credits provided as part of Google-MIT support, DARPA Machine Common Sense Program, ARO MURI under Grant Number W911NF-21-1-0328, ONR MURI under Grant Number N00014-22-1-2740, and by the United States Air Force Artificial Intelligence Accelerator under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

Author Contributions

- **Eric Chen** developed heuristic-based methods for balancing intrinsic and extrinsic rewards which led to the intuition for formulating the extrinsic-intrinsic optimality constraint. He provided critical input during EIPO prototyping. Implemented baselines, ran experiments at scale, and helped with paper writing.
- **Zhang-Wei Hong** conceived of the extrinsic-intrinsic optimality constraint, derived and developed the first prototype of EIPO, ran small-scale experiments, played primary role in paper writing and advised Eric.
- **Joni Pajarinen** provided feedback on the technical correctness of EIPO formulation and writing.
- **Pulkit Agrawal** conceived and oversaw the project. He was involved in the technical formulation, research discussions, paper writing, overall advising and positioning of the work.

References

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.

[2] Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. 2020.

[3] Dylan Foster and Alexander Rakhlin. Beyond ucb: Optimal and efficient contextual bandits with regression oracles. In *International Conference on Machine Learning*, pages 3199–3210. PMLR, 2020.

[4] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *NIPS*, 2016.

[5] Jurgen Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In *From animals to animats: Proceedings of the first international conference on simulation of adaptive behavior*, 1991.

[6] Nuttapong Chentanez, Andrew Barto, and Satinder Singh. Intrinsically motivated reinforcement learning. *Advances in neural information processing systems*, 17, 2004.

[7] Pierre-Yves Oudeyer and Frederic Kaplan. What is intrinsic motivation? a typology of computational approaches. *Frontiers in neurorobotics*, 2009.

[8] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the 34th International Conference on Machine Learning*, pages 2778–2787, 2017.

[9] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=H1lJJnR5Ym.

[10] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. First return, then explore. *Nature*, 590(7847):580–586, 2021.

[11] Adrien Ali Taïga, William Fedus, Marlos C Machado, Aaron Courville, and Marc G Bellemare. Benchmarking bonus-based exploration methods on the arcade learning environment. *arXiv preprint arXiv:1908.02388*, 2019.

[12] William F Whitney, Michael Bloesch, Jost Tobias Springenberg, Abbas Abdolmaleki, Kyunghyun Cho, and Martin Riedmiller. Decoupled exploration and exploitation policies for sample-efficient reinforcement learning. *arXiv preprint arXiv:2101.09458*, 2021.

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

[14] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018.

[15] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *In Proc. 19th International Conference on Machine Learning*. Citeseer, 2002.

[16] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1889–1897, 2015.

[17] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

[18] Lukas Schäfer, Filippos Christianos, Josiah Hanna, and Stefano V Albrecht. Decoupling exploration and exploitation in reinforcement learning. *arXiv preprint arXiv:2107.08966*, 2021.

[19] Thomas Stepleton. The pycolab game engine, 2017. *URL https://github. com/deepmind/pycolab*, 2017.

[20] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[21] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 34, 2021.

[22] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

[23] 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.

[24] Gabriel B Margolis, Ge Yang, Kartik Paigwar, Tao Chen, and Pulkit Agrawal. Rapid locomotion via reinforcement learning. *Robotics: Science and Systems*, 2022.

[25] Tao Chen, Jie Xu, and Pulkit Agrawal. A system for general in-hand object re-orientation. *Conference on Robot Learning*, 2021.

[26] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *International Conference of Representation Learning*, 2015.

[27] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033. IEEE, 2012.

[28] Cetin Meriçli, Tekin Meriçli, and H Levent Akin. A reward function generation method using genetic algorithms: a robot soccer case study. In *AAMAS*, pages 1513–1514, 2010.

[29] Jonathan Sorg, Richard L Lewis, and Satinder Singh. Reward design via online gradient ascent. *Advances in Neural Information Processing Systems*, 23, 2010.

[30] Xiaoxiao Guo, Satinder Singh, Richard Lewis, and Honglak Lee. Deep learning for reward design to improve monte carlo tree search in atari games. *arXiv preprint arXiv:1604.07095*, 2016.

[31] Zeyu Zheng, Junhyuk Oh, and Satinder Singh. On learning intrinsic rewards for policy gradient methods. *Advances in Neural Information Processing Systems*, 31, 2018.

[32] Zeyu Zheng, Junhyuk Oh, Matteo Hessel, Zhongwen Xu, Manuel Kroiss, Hado Van Hasselt, David Silver, and Satinder Singh. What can learned intrinsic rewards capture? In *International Conference on Machine Learning*, pages 11436–11446. PMLR, 2020.

[33] Yujing Hu, Weixun Wang, Hangtian Jia, Yixiang Wang, Yingfeng Chen, Jianye Hao, Feng Wu, and Changjie Fan. Learning to utilize shaping rewards: A new approach of reward shaping. *Advances in Neural Information Processing Systems*, 33:15931–15941, 2020.

[34] Xi Chen, Ali Ghadirzadeh, Mårten Björkman, and Patric Jensfelt. Meta-learning for multi-objective reinforcement learning. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 977–983. IEEE, 2019.

[35] Christian Shelton. Balancing multiple sources of reward in reinforcement learning. *Advances in Neural Information Processing Systems*, 13, 2000.

[36] Stefan Ultes, Paweł Budzianowski, Inigo Casanueva, Nikola Mrkšić, Lina Rojas-Barahona, Pei-Hao Su, Tsung-Hsien Wen, Milica Gašić, and Steve Young. Reward-balancing for statistical spoken dialogue systems using multi-objective reinforcement learning. *arXiv preprint arXiv:1707.06299*, 2017.

[37] Runzhe Yang, Xingyuan Sun, and Karthik Narasimhan. A generalized algorithm for multi-objective reinforcement learning and policy adaptation. *Advances in Neural Information Processing Systems*, 32, 2019.

[38] Axel Abels, Diederik Roijers, Tom Lenaerts, Ann Nowé, and Denis Steckelmacher. Dynamic weights in multi-objective deep reinforcement learning. In *International Conference on Machine Learning*, pages 11–20. PMLR, 2019.

A Appendix

A.1 Full derivation

We present the complete derivation of the objective function in each sub-problem defined in Section 3.2. We start by clarifying notation:

A.1.1 Notations

- $J_E(\pi) = \mathbb{E}_{s_0,a_0,\cdots\sim\pi}\Big[\sum_{t=0}^{\infty}\gamma^t r_t^E\Big], s_0 \sim \rho_0, a_t \sim \pi(a|s_t), s_{t+1} \sim \mathcal{T}(s_{t+1}|s_t,a_t)\ \forall t > 0$
- $J_{E+I}(\pi) = \mathbb{E}_{\pi}\Big[\sum_{t=0}^{\infty}\gamma^t(r_t^E + r_t^I)\Big]$
- $V_E^{\pi}(s_t) := \mathbb{E}_{\pi}\Big[\sum_{t=0}^{\infty}\gamma^t r_t^E|s_0 = s_t\Big]$
- $V_{E+I}^{\pi}(s_t) := \mathbb{E}_{\pi}\Big[\sum_{t=0}^{\infty}\gamma^t(r_t^E + r_t^I)|s_0 = s_t\Big]$
- $V_{E+I}^{\pi,\alpha}(s_t) := \mathbb{E}_{\pi}\Big[\sum_{t=0}^{\infty}\gamma^t((1+\alpha)r_t^E + r_t^I)|s_0 = s_t\Big]$

A.1.2 Max-stage objective $U_{\max}^{\pi_E}$

We show that the objective (LHS) can be lower bounded by the RHS shown below

$$J_{E+I}^{\alpha}(\pi_{E+I}) - \alpha J_E(\pi_E) \geq \mathbb{E}_{\pi_E}\Big[\sum_{t=0}^{\infty}\gamma^t \min\Big\{\frac{\pi_{E+I}(a_t|s_t)}{\pi_E(a_t|s_t)}U_{\max}^{\pi_E}(s_t,a_t), \text{clip}\left(\frac{\pi_{E+I}(a_t|s_t)}{\pi_E(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right)U_{\max}^{\pi_E}(s_t,a_t)\Big\}. \tag{13}$$

We can then expand the LHS as the follows:

$$\begin{aligned}
&J_{E+I}^{\alpha}(\pi_{E+I}) - \alpha J_E(\pi_E)\\
&= -\alpha J_E(\pi_E) + J_{E+I}^{\alpha}(\pi_{E+I})\\
&= -\alpha\mathbb{E}_{s_0\sim\rho_0}\Big[\sum_{t=0}^{\infty}\gamma^t V_{\pi_E}^E(s_t)\Big] + \mathbb{E}_{\pi_{E+I}}\Big[\sum_{t=0}^{\infty}\gamma^t((1+\alpha)r_t^E + r_t^I)\Big]\\
&= \mathbb{E}_{\pi_{E+I}}\Big[-\alpha V_E^{\pi_E}(s_0) + \sum_{t=0}^{\infty}\gamma^t((1+\alpha)r_t^E + r_t^I)\Big]
\end{aligned}$$

For brevity, let $r_t = (1+\alpha)r_t^E + r_t^I$ and $\alpha V_E^{\pi_E}(s_t) = V_t$. Expanding the left-hand of Eq. 13:

$$\begin{aligned}
-V_0 + \sum_{t=0}^{\infty}\gamma^t r_t &= (r_0 + \cancel{\gamma V_1} - V_0) + \gamma(r_1 + \cancel{\gamma V_2} - \cancel{V_1}) + \gamma^2(r_2 + \cancel{\gamma V_3} - \cancel{V_2}) + \cdots\\
&= \sum_{t=0}^{\infty}\gamma^t(r_t + \gamma V_{t+1} - V_t)\\
&= \sum_{t=0}^{\infty}\gamma^t((1+\alpha)r_t^E + r_t^I + \gamma\alpha V_E^{\pi_E}(s_{t+1}) - \alpha V^{\pi_E}(s_t))\\
&= \sum_{t=0}^{\infty}\gamma^t U_{\max}^{\pi_E}(s_t,a_t)
\end{aligned}$$

To facilitate the following derivation, we rewrite the objective $J^{\alpha}_{E+I}(\pi_{E+I}) - \alpha J_E(\pi_E)$:

$$J^{\alpha}_{E+I}(\pi_{E+I}) - \alpha J_E(\pi_E) = \mathbb{E}_{\pi_{E+I}}\Big[\sum_{t=0}^{\infty} \gamma U^{\pi_E}_{\max}(s_t, a_t)\Big] \tag{14}$$

$$= \sum_{t=0}^{\infty} \sum_{s\in\mathcal{S}} \gamma P(s_t = s|\rho_0, \pi_{E+I}) \sum_{a\in\mathcal{A}} \pi_{E+I}(a|s) U^{\pi_E}_{\max}(s_t, a_t) \tag{15}$$

$$= \sum_{s\in\mathcal{S}} d^{\pi_{E+I},\gamma}_{\rho_0}(s) \sum_{a\in\mathcal{A}} \pi_{E+I}(a|s) U^{\pi_E}_{\max}(s_t, a_t) \tag{16}$$

$$= \mathbb{E}_{\pi_{E+I}}\Big[U^{\pi_E}_{\max}(s_t, a_t)\Big] \tag{17}$$

where $d^{\pi_{E+I},\gamma}_{\rho_0}$ is the discounted state visitation frequency of policy π_{E+I} with the initial state distribution ρ_0 and discount factor γ, defined as:

$$d^{\pi_{E+I},\gamma}_{\rho_0}(s) = \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\rho_0, \pi_{E+I})$$

Note that for brevity, we write $\mathbb{E}_{s\sim d^{\pi_{E+I},\gamma}_{\rho_0}, a\sim\pi}\Big[.\Big]$ as $\mathbb{E}_{\pi_{E+I}}\Big[.\Big]$ instead. To get rid of the dependency on samples from π_{E+I}, we use the local approximation [16, 15] shown below:

$$L^{\alpha}_{E+I}(\pi_{E+I}) = \alpha J_E(\pi_E) + \sum_{s\in\mathcal{S}} d^{\pi_E,\gamma}_{\rho_0}(s) \sum_{a\in\mathcal{A}} \pi_{E+I}(a|s) U^{\pi_E}_{\max}(s_t, a_t). \tag{18}$$

The discounted state visitation frequency of π_{E+I} is replaced with that of π_E. This local approximation is useful because if we can find a π_0 such that $L^{\alpha}_{E+I}(\pi_0) = J^{\alpha}_{E+I}(\pi_0)$, the local approximation matches the target in the first order: $\nabla_{\pi_{E+I}} L^{\alpha}_{E+I}(\pi_{E+I})|_{\pi_{E+I}=\pi_0} = \nabla_{\pi_{E+I}} J^{\alpha}_{E+I}(\pi_{E+I})|_{\pi_{E+I}=\pi_0}$. This implies that if $L^{\alpha}_{E+I}(\pi_{E+I})$ is improved, $J^{\alpha}_{E+I}(\pi_{E+I})|_{\pi_{E+I}=\pi_0}$ will be improved as well. Using the technique in Schulman et al. [16], this local approximation can lower bound $J^{\alpha}_{E+I}(\pi_{E+I}) - \alpha J_E(\pi_E)$ with an additional KL-divergence penalty as shown below:

$$J^{\alpha}_{E+I}(\pi_{E+I}) - \alpha J_E(\pi_E) \geq L^{\alpha}_{E+I}(\pi_{E+I}) - \alpha J_E(\pi_E) - C\mathrm{D}^{\max}_{\mathrm{KL}}(\pi_E||\pi_{E+I}) \tag{19}$$

$$= \sum_{s\in\mathcal{S}} d^{\pi_E,\gamma}_{\rho_0}(s) \sum_{a\in\mathcal{A}} \pi_{E+I}(a|s) U^{\pi_E}_{\max}(s_t, a_t) - C\mathrm{D}^{\max}_{\mathrm{KL}}(\pi_E||\pi_{E+I}) \tag{20}$$

$$= \sum_{s\in\mathcal{S}} d^{\pi_E,\gamma}_{\rho_0}(s) \sum_{a\in\mathcal{A}} \frac{\pi_{E+I}(a|s)}{\pi_E(a|s)} U^{\pi_E}_{\max}(s_t, a_t) - C\mathrm{D}^{\max}_{\mathrm{KL}}(\pi_E||\pi_{E+I}) \tag{21}$$

$$= \mathbb{E}_{\pi_E}\Big[\frac{\pi_{E+I}(a|s)}{\pi_E(a|s)} U^{\pi_E}_{\max}(s, a)\Big] - C\mathrm{D}^{\max}_{\mathrm{KL}}(\pi_E||\pi_{E+I}) \tag{22}$$

where C denotes a constant. As it is intractable to evaluate $\mathrm{D}^{\max}_{\mathrm{KL}}(\pi_E||\pi_{E+I})$, the clipping approach proposed in Schulman et al. [13] receives wider spread of use. Following [13], we convert Eq. 22 to a clipped objective shown as follows:

$$\mathbb{E}_{\pi_E}\Big[\sum_{t=0}^{\infty} \gamma^t \min\Big\{\frac{\pi_{E+I}(a_t|s_t)}{\pi_E(a_t|s_t)} U^{\pi_E}_{\max}(s_t, a_t), \mathrm{clip}\left(\frac{\pi_{E+I}(a_t|s_t)}{\pi_E(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right) U^{\pi_E}_{\max}(s_t, a_t)\Big\}\Big], \tag{23}$$

Note that to make use of the approximation proposed in [16, 15], we make the assumption that in the beginning of the max-stage, $\pi_E = \pi_{E+I}$. Under this assumption, π_E serves as π_0 (see above). This enables updating π_{E+I} using the local approximation. We leave relaxing this assumption as future work.

A.1.3 Min-stage objective $U_{\min}^{\pi_{E+I}}$

We show that the objective (LHS) can be approximated by the RHS shown below

$$\begin{aligned}\alpha J_E(\pi_E) - J_{E+I}^{\alpha}(\pi_{E+I}) \geq \mathbb{E}_{\pi_{E+I}}\Big[\sum_{t=0}^{\infty}\gamma^t \min\Big\{&\frac{\pi_E(a_t|s_t)}{\pi_{E+I}(a_t|s_t)}U_{\min}^{\pi_{E+I}}(s_t,a_t),\\ &\text{clip}\left(\frac{\pi_E(a_t|s_t)}{\pi_{E+I}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right)U_{\min}^{\pi_{E+I}}(s_t,a_t)\Big\}.\end{aligned} \tag{24}$$

The derivation for the min-stage is quite similar to that of the max-stage. Thus we only outline the key elements:

$$\alpha J_E(\pi_E) - J_{E+I}^{\alpha}(\pi_{E+I}) = -J_{E+I}^{\alpha}(\pi_{E+I}) + \alpha J_E(\pi_E) \tag{25}$$

$$= -\mathbb{E}_{s_0}\Big[V_{E+I}^{\pi_{E+I},\alpha}(s_0)\Big] + \alpha\mathbb{E}_{\pi_E}\Big[\sum_{t=0}^{\infty}\gamma^t r_t^E\Big] \tag{26}$$

$$= \mathbb{E}_{\pi_E}\Big[-V_{E+I}^{\pi_{E+I},\alpha}(s_0) + \sum_{t=0}^{\infty}\gamma^t \alpha r_t^E\Big] \tag{27}$$

$$= \mathbb{E}_{\pi_E}\Big[\sum_{t=0}^{\infty}\gamma^t(\alpha r_t^E + \gamma V_{E+I}^{\pi_{E+I},\alpha}(s_{t+1}) - V_{E+I}^{\pi_{E+I},\alpha}(s_t))\Big] \tag{28}$$

$$= \mathbb{E}_{\pi_E}\Big[\sum_{t=0}^{\infty}\gamma^t \alpha r_t^E + \gamma V_{E+I}^{\pi_{E+I},\alpha}(s_{t+1}) - V_{E+I}^{\pi_{E+I},\alpha}(s_t)\Big] \tag{29}$$

$$\approx \mathbb{E}_{\pi_E}\Big[\sum_{t=0}^{\infty}\gamma^t \alpha r_t^E + \gamma V_{E+I}^{\pi_{E+I}}(s_{t+1}) - V_{E+I}^{\pi_{E+I}}(s_t)\Big]. \tag{30}$$

Since we empirically find that $V_{E+I}^{\pi_{E+I},\alpha}$ is hard to fit under a continually changing α, we replace $V_{E+I}^{\pi_{E+I},\alpha}$ with $V_{E+I}^{\pi_{E+I}}$ in Equation 29, which leads to Eq. 30. Following the same recipe used in deriving the objective of max-stage gives rise to the following objective:

$$\mathbb{E}_{\pi_{E+I}}\Big[\sum_{t=0}^{\infty}\gamma^t \min\Big\{\frac{\pi_E(a_t|s_t)}{\pi_{E+I}(a_t|s_t)}U_{\min}^{\pi_{E+I}}(s_t,a_t), \text{clip}\left(\frac{\pi_E(a_t|s_t)}{\pi_{E+I}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right)U_{\min}^{\pi_{E+I}}(s_t,a_t)\Big\}\Big] \tag{31}$$

A.1.4 α optimization

Let $g(\alpha) := \max_{\pi_{E+I}\in\Pi}\min_{\pi_E\in\Pi} J_{E+I}^{\alpha}(\pi_{E+I}) - \alpha J_E(\pi_E)$. As π_E and π_{E+I} are not yet optimal during the training process, we solve $\min_\alpha g(\alpha)$ using stochastic gradient descent as shown below:

$$\alpha \leftarrow \alpha - \beta\nabla_\alpha g(\alpha) \tag{32}$$

$$= \alpha - \beta\nabla_\alpha(J_{E+I}^{\alpha}(\pi_{E+I}) - \alpha J_E(\pi_E)) \tag{33}$$

$$= \alpha - \beta(J_E(\pi_{E+I}) - J_E(\pi_E)) \tag{34}$$

where β is the learning rate of α. $J_E(\pi_{E+I}) - J_E(\pi_E)$ can be lower bounded using the technique proposed in [13] as follows:

$$\begin{aligned}J_E(\pi_{E+I}) - J_E(\pi_E) \geq \mathbb{E}_{\pi_E}\Big[\sum_{t=0}^{\infty}\gamma^t \min\Big\{&\frac{\pi_{E+I}(a_t|s_t)}{\pi_E(a_t|s_t)}A^{\pi_E}(s_t,a_t),\\ &\text{clip}\left(\frac{\pi_{E+I}(a_t|s_t)}{\pi_E(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right)A^{\pi_E}(s_t,a_t)\Big\}\Big],\end{aligned} \tag{35}$$

where $A_E^{\pi_E}(s_t,a_t) := r_t^E + \gamma V_E^{\pi_E}(s_{t+1}) - V_E^{\pi_E}(s_t)$ denotes the extrinsic advantage of π_E.

A.2 Implementation details

A.2.1 Algorithm

Clipped objective We use proximal policy optimization (PPO) [13] to optimize the constrained objectives in Equation 6 and Equation 9. The policies π_E and π_{E+I} are obtained by solving the following optimization problems with clipped objectives:

- **Max-stage:**

$$\max_{\pi_{E+I}} \mathbb{E}_{\pi_E^{\text{old}}}\Big[\min\Big\{\frac{\pi_{E+I}(a|s)}{\pi_E^{\text{old}}(a|s)}U_{\max}^{\pi_E^{\text{old}}}(s,a), \text{clip}(\frac{\pi_{E+I}(a|s)}{\pi_E^{\text{old}}(a|s)}, 1-\epsilon, 1+\epsilon)U_{\max}^{\pi_E^{\text{old}}}(s,a)\Big\}\Big] \tag{36}$$

- **Min-stage:**

$$\max_{\pi_E} \mathbb{E}_{\pi_{E+I}^{\text{old}}}\Big[\min\Big\{\frac{\pi_E(a|s)}{\pi_{E+I}^{\text{old}}(a|s)}U_{\min}^{\pi_{E+I}^{\text{old}}}(s,a), \text{clip}(\frac{\pi_E(a|s)}{\pi_{E+I}^{\text{old}}(a|s)}, 1-\epsilon, 1+\epsilon)U_{\min}^{\pi_{E+I}^{\text{old}}}(s,a)\Big\}\Big] \tag{37}$$

where ϵ denotes the clipping threshold for PPO, and π_E^{old} and π_{E+I}^{old} denote the policies that collect trajectories. We will detail the choices of ϵ in the following paragraphs.

Rearranging the expression for GAE To leverage generalized advantage estimation (GAE) [26], we rearrange $U_{\max}^{\pi_E}$ and $U_{\min}^{\pi_{E+I}}$ to relate them to the advantage functions. The advantage function $A_E^{\pi_E}$ and $A_{E+I}^{\pi_{E+I}}$ are defined as:

$$A_E^{\pi_E}(s_t) = r_t^E + \gamma V_E^{\pi_E}(s_{t+1}) - V_E^{\pi_E}(s_t) \tag{38}$$

$$A_{E+I}^{\pi_{E+I}}(s_t) = r_t^E + r_t^I + \gamma V_{E+I}^{\pi_{E+I}}(s_{t+1}) - V_{E+I}^{\pi_{E+I}}(s_t). \tag{39}$$

As such, we can rewrite $U_{\max}^{\pi_E}$ and $U_{\min}^{\pi_{E+I}}$ as:

$$U_{\max}^{\pi_E}(s_t, a_t) = (1+\alpha)r_t^E + r_t^I + \gamma\alpha V_E^{\pi_E}(s_{t+1}) - \alpha V_E^{\pi_E}(s_t) \tag{40}$$

$$= r_t^E + r_t^I + \alpha A_E^{\pi_E}(s_t) \tag{41}$$

$$U_{\min}^{\pi_{E+I}}(s_t, a_t) = \alpha r_t^E + \gamma V_{E+I}^{\pi_{E+I}}(s_{t+1}) - V_{E+I}^{\pi_E+I}(s_t) \tag{42}$$

$$= (\alpha-1)r_t^E - r_t^I + A_E^{\pi_{E+I}}(s_t). \tag{43}$$

A.2.2 Extrinsic reward normalization

For each parallel worker, we maintain the running average of the extrinsic rewards $\bar{r}^E$. This value is updated in the following manner at each timestep t:

$$\bar{r}^E \leftarrow \gamma\bar{r}^E + r_t^E.$$

The extrinsic rewards are then rescaled by the standard deviation of $\bar{r}^E$ across workers as shown below:

$$r_t^E \leftarrow r_t^E / \text{Var}\Big[\bar{r}^E\Big].$$

A.2.3 Auxiliary objectives

The auxiliary objectives for each stage are listed below:

- **Max-stage:** We train the extrinsic policy π_E to maximize $J_E(\pi_E)$ using PPO as shown below:

$$\max_{\pi_E} \mathbb{E}_{\pi_E^{\text{old}}}\Big[\min\Big\{\frac{\pi_E(a|s)}{\pi_E^{\text{old}}(a|s)}A_E^{\pi_E^{\text{old}}}(s,a), \text{clip}(\frac{\pi_E(a|s)}{\pi_E^{\text{old}}(a|s)}, 1-\epsilon, 1+\epsilon)A_E^{\pi_E^{\text{old}}}(s,a)\Big\}\Big], \tag{44}$$

  where π_E^{old} denotes the extrinsic policy that collects trajectories at the current iteration.

- **Min-stage:** We train the mixed policy π_{E+I} to maximize $J_{E+I}(\pi_{E+I})$ using PPO as shown below:

$$\max_{\pi_{E+I}} \mathbb{E}_{\pi_{E+I}^{\text{old}}}\Big[\min\Big\{\frac{\pi_{E+I}(a|s)}{\pi_{E+I}^{\text{old}}(a|s)}A_{E+I}^{\pi_{E+I}^{\text{old}}}(s,a), \text{clip}(\frac{\pi_{E+I}(a|s)}{\pi_{E+I}^{\text{old}}(a|s)}, 1-\epsilon, 1+\epsilon)A_{E+I}^{\pi_{E+I}^{\text{old}}}(s,a)\Big\}\Big], \tag{45}$$

  where π_{E+I}^{old} denotes the mixed policy that collects trajectories at the current iteration.

A.2.4 Overall objective

In summary, we outline all the primary objectives Eqs. 36 and 37 and auxiliary objectives Eqs. 44 and 45 as follows:

$$\hat{J}_{E+I}(\pi_{E+I}) = \max_{\pi_{E+I}} \mathbb{E}_{\pi_E^{\text{old}}}\Big[\min\Big\{\frac{\pi_{E+I}(a|s)}{\pi_E^{\text{old}}(a|s)} U_{\max}^{\pi_E^{\text{old}}}(s,a), \text{clip}(\frac{\pi_{E+I}(a|s)}{\pi_E^{\text{old}}(a|s)}, 1-\epsilon, 1+\epsilon) U_{\max}^{\pi_E^{\text{old}}}(s,a)\Big\}\Big]$$

$$\hat{J}_{E}(\pi_{E}) = E\pi_{E+I}^{\text{old}} \min\Big\{\frac{\pi_{E}(a|s)}{\pi_{E+I}^{\text{old}}(a|s)} U_{\min}^{\pi_{E+I}^{\text{old}}}(s,a), \text{clip}(\frac{\pi_{E}(a|s)}{\pi_{E+I}^{\text{old}}(a|s)}, 1-\epsilon, 1+\epsilon) U_{\min}^{\pi_{E+I}^{\text{old}}}(s,a)\Big\}$$

$$\hat{J}_{E+I}^{\text{aux}}(\pi_{E+I}) = \mathbb{E}_{\pi_{E+I}^{\text{old}}}\Big[\min\Big\{\frac{\pi_{E+I}(a|s)}{\pi_{E+I}^{\text{old}}(a|s)} A_{E+I}^{\pi_{E+I}^{\text{old}}}(s,a), \text{clip}(\frac{\pi_{E+I}(a|s)}{\pi_{E+I}^{\text{old}}(a|s)}, 1-\epsilon, 1+\epsilon) A_{E+I}^{\pi_{E+I}^{\text{old}}}(s,a)\Big\}\Big]$$

$$\hat{J}_{E}^{\text{aux}}(\pi_{E}) = \mathbb{E}_{\pi_E^{\text{old}}}\Big[\min\Big\{\frac{\pi_{E}(a|s)}{\pi_E^{\text{old}}(a|s)} A_{E}^{\pi_E^{\text{old}}}(s,a), \text{clip}(\frac{\pi_{E}(a|s)}{\pi_E^{\text{old}}(a|s)}, 1-\epsilon, 1+\epsilon) A_{E}^{\pi_E^{\text{old}}}(s,a)\Big\}\Big].$$

The overall objectives optimized in both stages is summarized as follows:

$$\max_{\pi_{E+I}, \pi_E} \hat{J}_{E+I}(\pi_{E+I}) + \hat{J}_E^{\text{aux}}(\pi_E) \qquad \text{(Max-stage)} \qquad (46)$$

$$\max_{\pi_{E+I}, \pi_E} \hat{J}_{E}(\pi_{E}) + \hat{J}_{E+I}^{\text{aux}}(\pi_{E+I}) \qquad \text{(Min-stage)} \qquad (47)$$

Clipping the derivative of α The derivative of α, $\delta\alpha$ (see Section 3.3), is clipped to be within $(-\epsilon_\alpha, \epsilon_\alpha)$, where ϵ_α is a non-negative constant.

Codebase We implemented our method and each baseline on top of the `rlpyt`[2] codebase. We thank Adam Stooke and the `rlpyt` team for their excellent work producing this codebase.

Summary We outline the steps of our method in Algorithm 2.

Algorithm 2 Detailed Extrinsic-Intrinsic Policy Optimization (EIPO)

Initialize policies π_{E+I} and π_E and value functions $V_{E+I}^{\pi_{E+I}}$ and $V_E^{\pi_E}$
Set `max_stage[0]` $\leftarrow$ `False`, and $J[0] \leftarrow 0$
for $i = 1 \cdots$ **do** ▷ i denotes iteration index
  if `max_stage[i - 1]` **then** ▷ Max-stage: rollout by π_E and update π_{E+I}
    Collect trajectories τ_E using π_E
    Update π_{E+I} and π_E by Eq. 46
    Update $V_E^{\pi_E}$ (see [26])
    $J[i] \leftarrow J_{E+I}^{\alpha}(\pi_{E+I}) - \alpha J_E(\pi_E)$
    `max_stage[i]` $\leftarrow J[i] - J[i-1] \leq 0$
  else ▷ Min-stage: rollout by π_{E+I} and update π_E
    Collect trajectories τ_{E+I} using π_{E+I}
    Update π_{E+I} and π_E by Eq. 47
    Update $V_{E+I}^{\pi_{E+I}}$ (see [26])
    $J[i] \leftarrow J_{E+I}^{\alpha}(\pi_{E+I}) - \alpha J_E(\pi_E)$
    `max_stage[i]` $\leftarrow J[i] - J[i-1] \geq 0$
  end if
  if `max_stage[i - 1] = True` and `max_stage[i] = False` **then**
    Update α (Eq. 32) ▷ Update when the max-stage is done
  end if
end for

A.2.5 Models

Network architecture Let `Conv2D(ic, oc, k, s, p)` be a 2D convolutional neural network layer with `ic` input channels, `oc` output channels, kernel size `k`, stride size `s`, and padding `p`. Let

[2] https://github.com/astooke/rlpyt

Table 2: PPO Hyperparameters

Name	Value
Num. parallel workers	128
Num. minibatches of PPO	4
Trajectory length of each worker	128
Learning rate of policy/value function	0.0001
Discount γ	0.99
Value loss weight	1.0
Gradient norm bound	1.0
GAE λ	0.95
Num. PPO epochs	4
Clipping ratio	0.1
Entropy loss weight	0.001
Max episode steps	27000

Table 3: RND Hyperparameters

Name	Value
Drop probability	0.25
Intrinsic reward scaling λ	1.0
Learning rate	0.0001

`LSTM(n, m)` and `MLP(n, m)` be a long-short term memory layer and a multi-layer perceptron (MLP) with `n`-dimensional inputs and `m`-dimensional outputs, respectively.

For policies and value functions, the CNN backbone is implemented as two CNN layers, `Conv2D(1, 16, 8, 4, 0)` and `Conv2D(16, 32, 4, 2, 1)`, followed by an LSTM layer, `LSTM($CNN_OUTPUT_SIZE, 512)`. The policies π_{E+I} and π_{E+I}, and the value functions $V_{E+I}^{\pi_{E+I}}$ and $V_{E+I}^{\pi_{E+I}}$ have separate MLPs that take the LSTM outputs as inputs. Each policy MLP is `MLP(512,` $|\mathcal{A}|$`)`, and each value function MLP is `MLP(512, 1)`.

For the prediction networks and target networks in *RND*, we use a model architecture with three CNN layers followed by three MLP layers. The CNN layers are defined as follows: `Conv2D(1, 32, 8, 4, 0)`, `Conv2D(32, 64, 4, 2, 0)`, and `Conv2D(64, 64, 3, 1, 0)`, with `LeakyReLU` activations in between each layer. The MLP layers are defined as follows: `MLP(7*7*64, 512)`, `MLP(512, 512)`, and `MLP(512, 512)`, with `ReLU` activations in between each layer.

Hyperparameters The hyperparameters for PPO, RND, and EIPO are listed in Table 2, Table 3, and Table 4, respectively.

A.3 Environment details

Pycolab

- State space $\mathcal{S}$: $\mathbb{R}^{3\times84\times84}$, 5×5 cropped top-down view of the agent's surroundings, scaled to an 84×84 RGB image (see the code in the supplementary materials for details).
- Action space $\mathcal{A}$: $\{$`UP`, `DOWN`, `LEFT`, `RIGHT`, `NO ACTION`$\}$.
- Extrinsic reward function $\mathcal{R}_E$: See section 4.1.

ATARI

Table 4: EIPO Hyperparameters

Name	Value
Initial α	0.5
Step size β of α	0.005
Clipping range of $\delta\alpha$ $(-\epsilon_\alpha, \epsilon_\alpha)$	0.05

- State space $\mathcal{S}$: $\mathbb{R}^{1\times84\times84}$, 84×84 gray images.
- Action space $\mathcal{A}$: Depends on the environment.
- Extrinsic reward function $\mathcal{R}_E$: Depends on the environment.

A.4 Evaluation details

A.4.1 Training length

We determine the training lengths of each method based on the the number frame that EO and RND require for convergence (i.e., the performances stop increasing and the learning curve plateaus). For most ATARI games, we train each method for 3000 iterations where each iteration consists of $128 * 128$ frames and thus in total for 49152000 frames. Except for `Montezuma's Revenge`, we train 6000 iterations.

For Pycolab, we train each method for 1563 iterations where each iteration consists of $64 * 500$ frames and thus $50,000,000$ frames.

A.4.2 Probability of improvement

We validate whether EIPO prevents the possible performance degradation introduced by intrinsic rewards, and consistently either improves or matches the performance of PPO in 61 ATARI games. As our goal is to investigate if an algorithm generally performs better than PPO instead of the performance gain, we evaluate each algorithm using the "probability of improvement" metric suggested in [21]. We ran at least 5 random seeds for each method in each environment, collecting the median extrinsic returns within the last 100 episodes and calculating the probability of improvements $P(X \geq \text{PPO})$ [3] with 95%-confidence interval against PPO for each algorithm X. The confidence interval is estimated using the bootstrapping method. The probability of improvement is defined as:

$$P(X \geq Y) = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} S(x_i, y_j), \quad S(x_i, y_j) = \begin{cases} 1, x_i \geq y_j \\ 0, x_i < y_j, \end{cases}$$

where x_i and y_j denote the samples of median of extrinsic return trials of algorithms X and Y, respectively.

We also define strict probability of improvement to measure how an algorithm dominate others:

$$P(X > Y) = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} S(x_i, y_j), \quad S(x_i, y_j) = \begin{cases} 1, x_i > y_j \\ \frac{1}{2}, x_i = y_j \\ 0, x_i < y_j, \end{cases}$$

A.4.3 Normalized score

In addition, we report the PPO-normalized score [21] to validate whether EIPO preserves the performance gain granted by RND when applicable. Let p_X be the distribution of median extrinsic returns over the last 100 episodes of training for an algorithm X. Defining p_{PPO} as the distribution of mean extrinsic returns in the last 100 episodes of training for PPO, and p_{rand} as the average extrinsic return of a random policy, then the PPO-normalized score of algorithm X is defined as: $\frac{p_X - p_{\text{rand}}}{p_{\text{PPO}} - p_{\text{rand}}}$.

A.4.4 λ tuning

Table 5 lists the λ values used in Section 4.3.

A.5 RND-dominating games

Table A.5 shows that the mean and median PPO-normalized score of each method with 95%-confidence interval in the set of games where RND performs better than PPO.

The set of games where RND performs better than PPO are listed below:

[3] Note that Agarwal et al. [21] define probability of improvements as $P(X > Y)$ while we adapt it to $P(X \geq Y)$ as we measure the likelihood an algorithm X can match or exceed an algorithm Y.

Table 5: Tuned λ value for each environment

	$r^E \ll \lambda r^I$	$r^E < \lambda r^I$	$r^E \approx \lambda r^I$	$r^E > \lambda r^I$	$r^E \gg \lambda r^I$
Enduro	38800	600	388	50	0.1
Jamesbond	2000	50	23	0.25	0.1
StarGunner	600	15	6.33	0.1	0.05
TimePilot	500	15	5	0.25	0.1
YarsRevenge	3000	50	30	5	0.1
Venture	500	50	5	0.5	0.05

Table 6: EIPO exhibits higher performance gains than RND in the games where RND is better than PPO. Despite being slightly below RND in terms of median score, EIPO attains the highest median among baselines other than RND.

Algorithm	PPO-normalized score			
	Mean (CI)		Median (CI)	
RND	384.57	(85.57, 756.69)	**1.22**	(1.17, 1.26)
Ext-norm RND	427.08	(86.53, 851.52)	1.05	(1.02, 1.14)
Decay-RND	383.83	(84.19, 753.17)	1.04	(1.01, 1.11)
Decoupled-RND	1.54	(1.09, 2.12)	1.00	(0.96, 1.06)
EIPO-RND	**435.56**	(109.45, 874.88)	1.13	(1.06, 1.23)

- `AirRaid`
- `Alien`
- `Assault`
- `Asteroids`
- `BankHeist`
- `Berzerk`
- `Bowling`
- `Boxing`
- `Breakout`
- `Carnival`
- `Centipede`
- `ChopperCommand`
- `DemonAttack`
- `DoubleDunk`
- `FishingDerby`
- `Frostbite`
- `Gopher`
- `Hero`
- `Kangaroo`
- `KungFuMaster`
- `MontezumaRevenge`
- `MsPacman`
- `Phoenix`
- `Pooyan`
- `Riverraid`
- `RoadRunner`
- `SpaceInvaders`

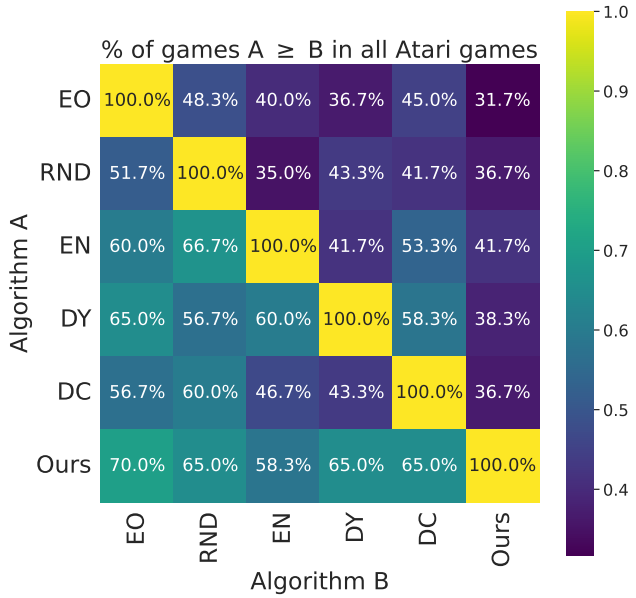


Figure 5: *EIPO-RND* performs better than PPO and all the other baselines in a majority of ATARI games.

- `Tutankham`
- `UpNDown`
- `Venture`

A.6 Scores for each ATARI game

The mean scores for each method on all ATARI games are presented in Table 7.

A.7 Complete comparison

In addition, Fig. 5 shows the percentage of ATARI games in which one method matches or exceeds the performance of another method with a pairwise comparison. Note that *"percentage of games* $A \geq B$*"* $R(A, B)$ is not equivalent to probability of improvements $P(A \geq B)$. Let μ_A be the mean score of algorithm A over 5 random seeds in an environment. $R(A, B)$ is defined as: $R(A, B) = \frac{\mathbb{1}\left[\mu_A \geq \mu_B\right]}{N}$ where N denotes the number of ATARI games. The last row of Fig. 5 shows that EIPO performs better than all the baselines in the majority of games (i.e., $R(\text{EIPO}, B) > 0.5$).

A.8 Complete learning curves

We present the learning curves of each method in Figure 6, and the evolution of α in EIPO in Figure 7 on all ATARI games.

A.9 Comparison with Noisy neural network exploration

In Figure 8, we also compare against the noisy network for exploration [22], an exploration strategy that was shown to perform better than ϵ-greedy [11] for deep Q-network [17]. Our method exhibits a higher strict and non-strict probability of improvements over the noisy network. Also, it can be seen that the noisy network does not consistently improve PPO, which mismatches the observation made in Fortunato et al. [22]. We hypothesize that the noisy network is not applicable in on-policy methods like PPO and hence degrades the performance since noisy networks turn sampled trajectories off-policy samples. Noisy networks collect trajectories using a policy network with perturbed weights. Trajectories sampled from perturbed networks cannot be counted as on-policy samples for the unperturbed policy network.

	EO	RND	Ext-norm RND	Decay-RND	Decouple-RND	Ours
Adventure	**0.0**	0.0	0.0	0.0	0.0	0.0
AirRaid	34693.2	42219.9	36462.4	36444.7	30356.4	**50418.2**
Alien	1891.0	2434.9	2152.1	2148.3	2386.9	**2536.7**
Amidar	**1053.4**	1037.0	736.4	909.5	987.1	901.3
Assault	8131.9	10592.2	**10985.1**	9504.3	8404.5	10771.1
Asterix	14313.0	14112.9	16872.5	**20078.0**	11292.2	12471.8
Asteroids	1360.9	1431.1	**1433.8**	1385.0	1426.7	1389.4
BankHeist	1336.3	1345.1	1339.0	**1346.0**	1334.8	1333.2
BattleZone	83826.0	47128.0	72117.0	61939.0	59461.7	**87478.0**
BeamRider	7278.7	7085.1	7460.0	7802.5	7215.4	**7854.6**
Berzerk	1113.8	**1478.5**	1459.0	1455.9	1196.4	1426.6
Bowling	17.4	14.6	26.0	32.6	19.0	**52.3**
Boxing	79.5	79.9	**79.9**	60.3	1.9	79.5
Breakout	565.7	**658.6**	570.6	545.7	479.3	529.5
Carnival	5019.3	5052.9	4513.4	4790.8	4964.7	**5534.3**
Centipede	5938.2	6444.4	6832.3	**6860.0**	6675.3	6460.8
ChopperCommand	8225.1	**9465.9**	8629.8	8559.0	6649.7	8008.4
CrazyClimber	**151202.6**	147676.5	135970.3	140333.9	138956.7	137036.7
DemonAttack	5678.8	7070.2	9039.0	6707.0	8990.1	**9984.4**
DoubleDunk	-1.3	**18.0**	-1.1	-1.0	-1.0	-1.9
ElevatorAction	45703.7	9777.6	12121.4	19250.5	42557.3	**48303.7**
Enduro	1024.7	797.5	815.0	**1095.9**	677.7	1092.6
FishingDerby	35.3	**47.8**	28.9	36.3	36.7	37.5
Freeway	31.1	25.8	33.4	**33.4**	33.1	33.3
Frostbite	1011.3	3445.3	1731.4	3368.2	2115.2	**5289.6**
Gopher	5544.2	**13035.8**	2859.6	11034.9	9964.6	4928.8
Gravitar	1682.2	1089.8	1874.1	1437.0	1253.4	**1921.1**
Hero	29883.7	**36850.3**	26781.2	29842.4	33889.1	36101.3
IceHockey	6.0	4.4	8.7	6.9	9.9	**10.4**
Jamesbond	13415.9	3971.6	13474.4	12322.4	14995.6	**15352.0**
JourneyEscape	-429.7	-1035.0	-663.7	-413.2	-327.8	**-309.3**
Kaboom	**1883.5**	1592.5	1866.6	1860.8	1830.7	1852.3
Kangaroo	6092.4	8058.9	8293.4	9361.9	**12043.3**	10150.8
Krull	9874.1	8199.4	9921.4	9832.0	9551.3	**10006.2**
KungFuMaster	47266.5	**66954.2**	48944.5	47403.2	45666.8	48329.4
MontezumaRevenge	0.2	2280.0	**2500.0**	2217.0	0.0	2485.0
MsPacman	4996.9	**5326.6**	5289.7	4792.5	4325.0	4767.4
NameThisGame	11127.7	10596.1	10300.7	11831.5	**11918.0**	11294.9
Phoenix	8265.0	10537.9	10922.9	11494.5	**17960.8**	16344.1
Pitfall	**0.0**	-2.7	-6.1	-0.6	-1.5	-0.3
Pong	20.9	20.9	**20.9**	20.9	20.9	20.9
Pooyan	5773.4	**7535.8**	5508.7	5430.9	4834.7	5924.6
PrivateEye	97.5	86.0	**114.9**	98.8	99.7	99.5
Qbert	**23863.8**	16530.9	22387.8	22443.3	22289.5	22750.7
Riverraid	10231.3	11073.6	11700.4	13365.7	13285.1	**14978.4**
RoadRunner	45922.6	46518.4	**58777.7**	44684.2	42694.3	58708.8
Robotank	37.4	24.9	38.5	40.1	40.7	**40.9**
Seaquest	1453.9	1128.6	**1986.0**	1426.6	1821.5	1838.3
Skiing	-12243.3	-14780.8	-11594.8	-11093.5	**-8986.6**	-9238.4
Solaris	2357.7	2006.5	2120.9	2251.7	**2751.0**	2572.0
SpaceInvaders	1621.0	**1871.4**	1495.3	1692.0	1375.7	1637.6
StarGunner	21036.0	16394.9	16884.7	32325.8	42299.5	**50798.5**
Tennis	-0.1	-4.7	**4.6**	-0.1	-8.2	-0.1
TimePilot	19544.5	9180.5	**21409.4**	20034.2	19223.8	21039.8
Tutankham	199.9	**235.3**	230.6	214.0	216.1	231.8
UpNDown	276884.8	**317426.2**	310520.6	266774.5	290323.4	294218.8
Venture	102.1	1149.7	1348.6	**1451.8**	1438.8	1146.3
VideoPinball	360562.5	327741.8	350534.3	**406508.8**	389578.5	392005.7
WizardOfWor	11912.8	9580.3	11845.2	11751.7	10732.7	**12512.8**
YarsRevenge	92555.9	73411.4	85851.9	77850.0	124983.6	**149710.8**
Zaxxon	14418.2	11801.9	11779.6	15085.5	**16813.3**	12713.3

Table 7: The mean scores of each method in 61 ATARI games.

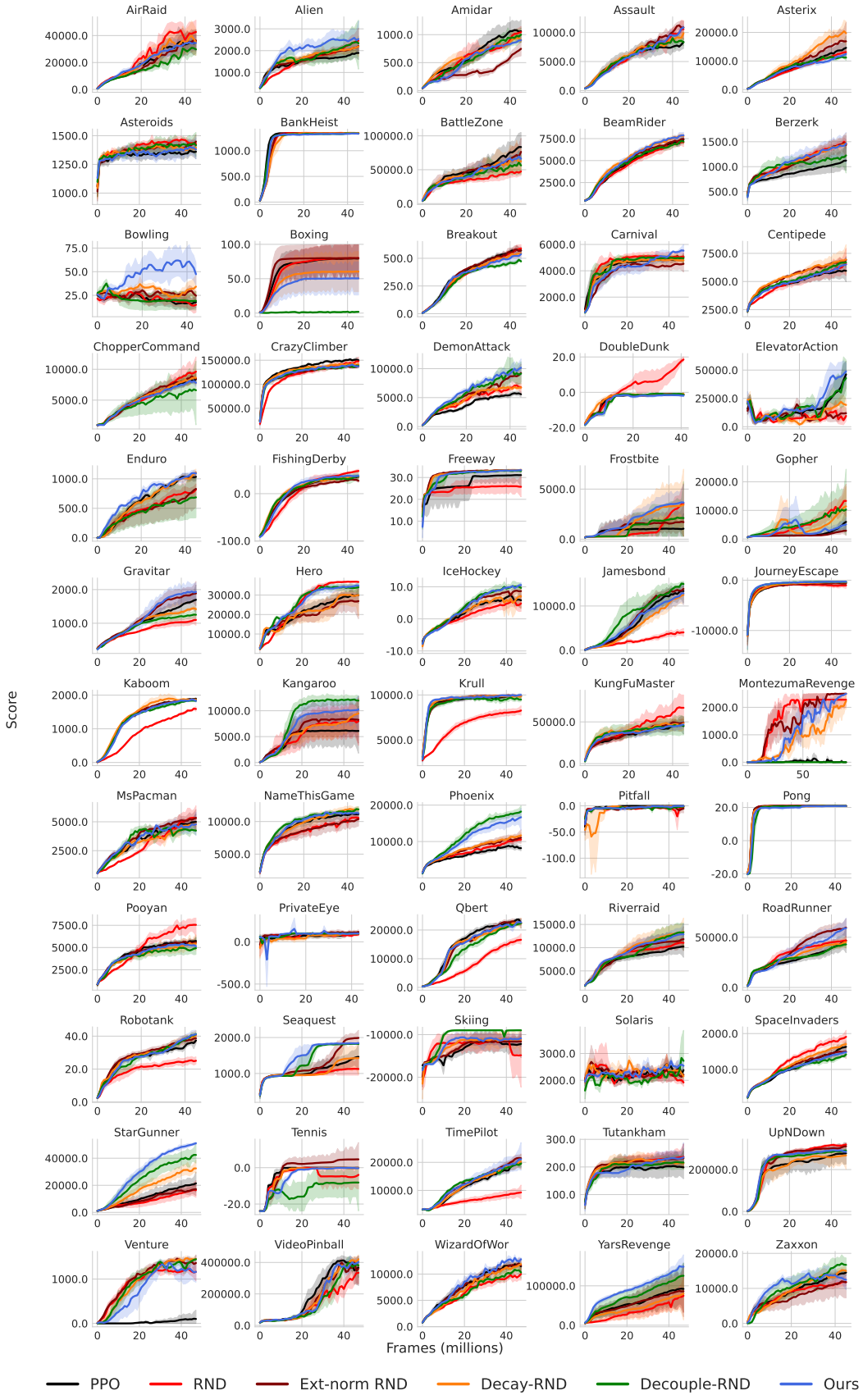


Figure 6: Game score for each baseline on 60 ATARI games. Each curve represents the average score across at least 5 random seeds. In all games, we either match or outperform PPO. In a large majority of games, we either match or outperform RND. In a handful of games, our method does significantly better than both PPO and RND (*Star Gunner*, *Bowling*, *Yars Revenge*, *Phoenix*, *Seaquest*).

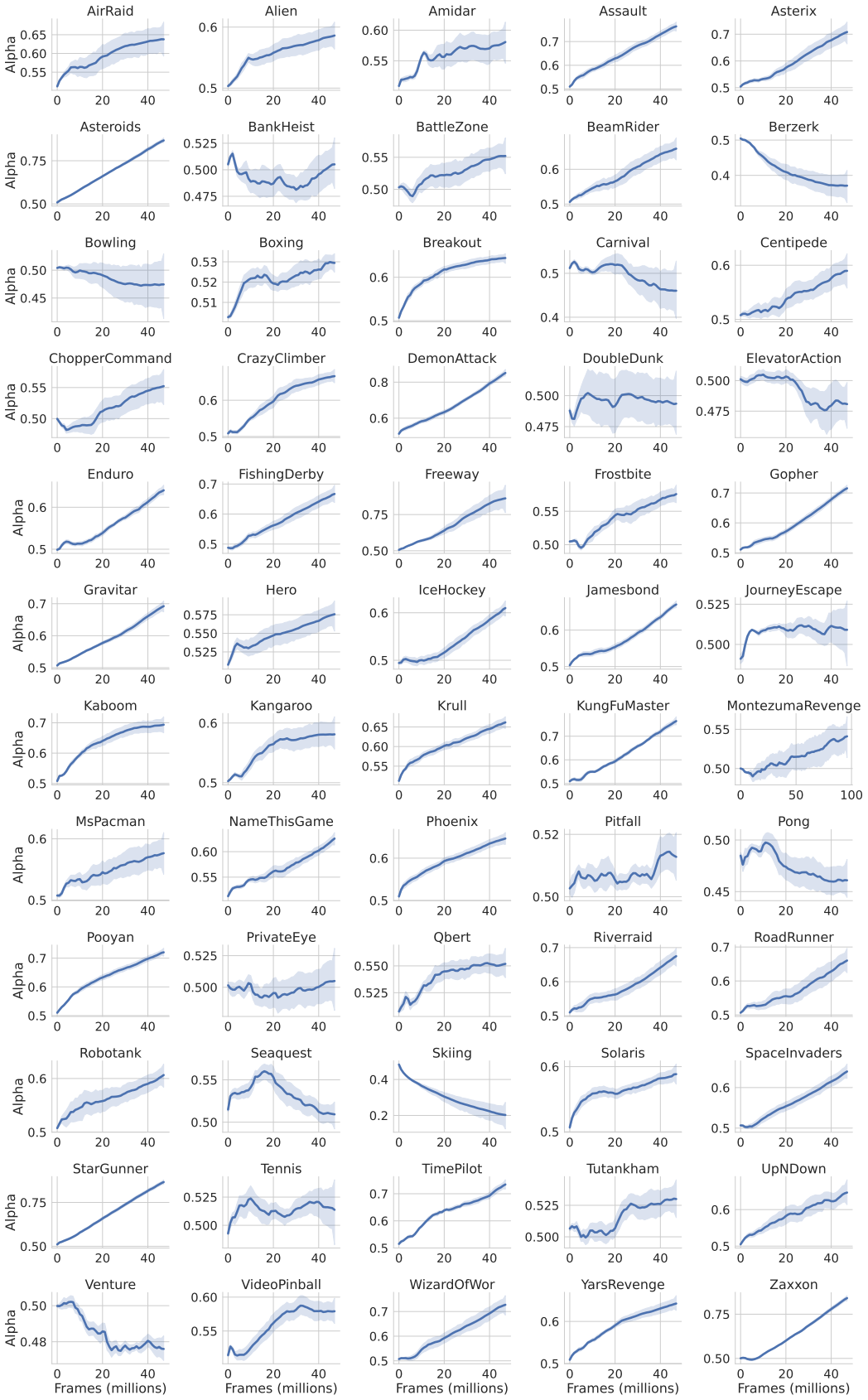
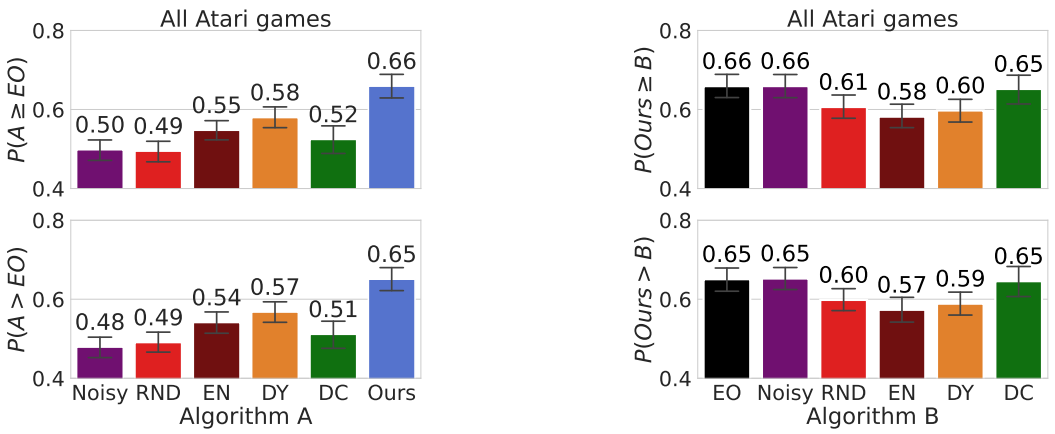


Figure 7: The evolution of α in EIPO on all 61 ATARI environments. The variance in α trajectories across environments supports the hypothesis that decaying the intrinsic reward is difficult to hand-tune, and may not always be the best strategy.



(a) (b)

Figure 8: **(a)** *EIPO-RND* (ours) has a higher probability of improvement $P(\textit{EIPO-RND} > \text{EO})$ over EO than all other baselines. Surprisingly, the noisy network for exploration (*Noisy*) [22] does not consistently improve the performance of PPO (EO) even though Taïga et al. [11] showed that noisy network improves ϵ-greedy exploration strategy used in deep Q-network (DQN). **(b)** Probability of improvement $P(\textit{EIPO-RND} > B) > 0.5\ \forall B$ indicates that *EIPO-RND* performs strictly better than the baselines in the majority of trials. Notably, *EIPO-RND* also outperforms *Noisy*, which is shown to be consistently better than extrinsic-reward-only exploration strategies (e.g., ϵ-greedy).

A.10 ICM

In addition to RND, we test our method on ICM [8] - another popular bonus-based exploration method. The learning curves on 6 ATARI environments can be seen in Fig. 9.

A.11 MuJoCo results

We present the experimental results of EIPO and other baselines on MuJoCo [27] in Figure 10. It can be seen that our method exceeds or matches PPO in most tasks. However, as the reward function of MuJoCo are often dense, there is no visble difference between RND and PPO.

A.12 Related Work

Reward design. Our work is related to the paradigm of reward design. Meriçli et al. [28] uses genetic programming to optimize the reward function for robot soccer. Sorg et al. [29] learns a reward function for planning via gradient ascent on the expected return of a tree search planning algorithm (e.g., Monte Carlo Tree Search). Guo et al. [30] extends [29] using deep learning, improving tree search performance in ATARI games. The work [31] learns a reward function to improve the performance of model-free RL algorithms by performing policy gradient updates on the reward function. Zheng et al. [32] takes a meta-learning approach to learn a reward function that improves an RL algorithm's sample efficiency in unseen environments. Hu et al. [33] learns a weighting function that scales the given shaping rewards [23] at each state and action. These lines of work are complimentary to EIPO, which is agnostic to the choice of intrinsic reward and could be used in tandem with a learned reward function.

Multi-objective reinforcement learning. The problem of balancing intrinsic and extrinsic rewards is related to multi-objective reinforcement learning (MORL) [34–37] since both optimize a policy with the linear combination of two or multiple rewards. Chen et al. [34] learns a meta-policy that jointly optimizes all the objectives and adapts to each objective by fine-tuning the meta-policy. Ultes et al. [36] search for the desired weighting between dialog length and success rate in conversation AI system using Bayesian optimization. Abels et al. [38], Yang et al. [37] make the policy conditional on the preference of each reward and train the policy using the weighted rewards. However, MORL approaches could not resolve the issue of intrinsic rewards bias in our problem setting since intrinsic

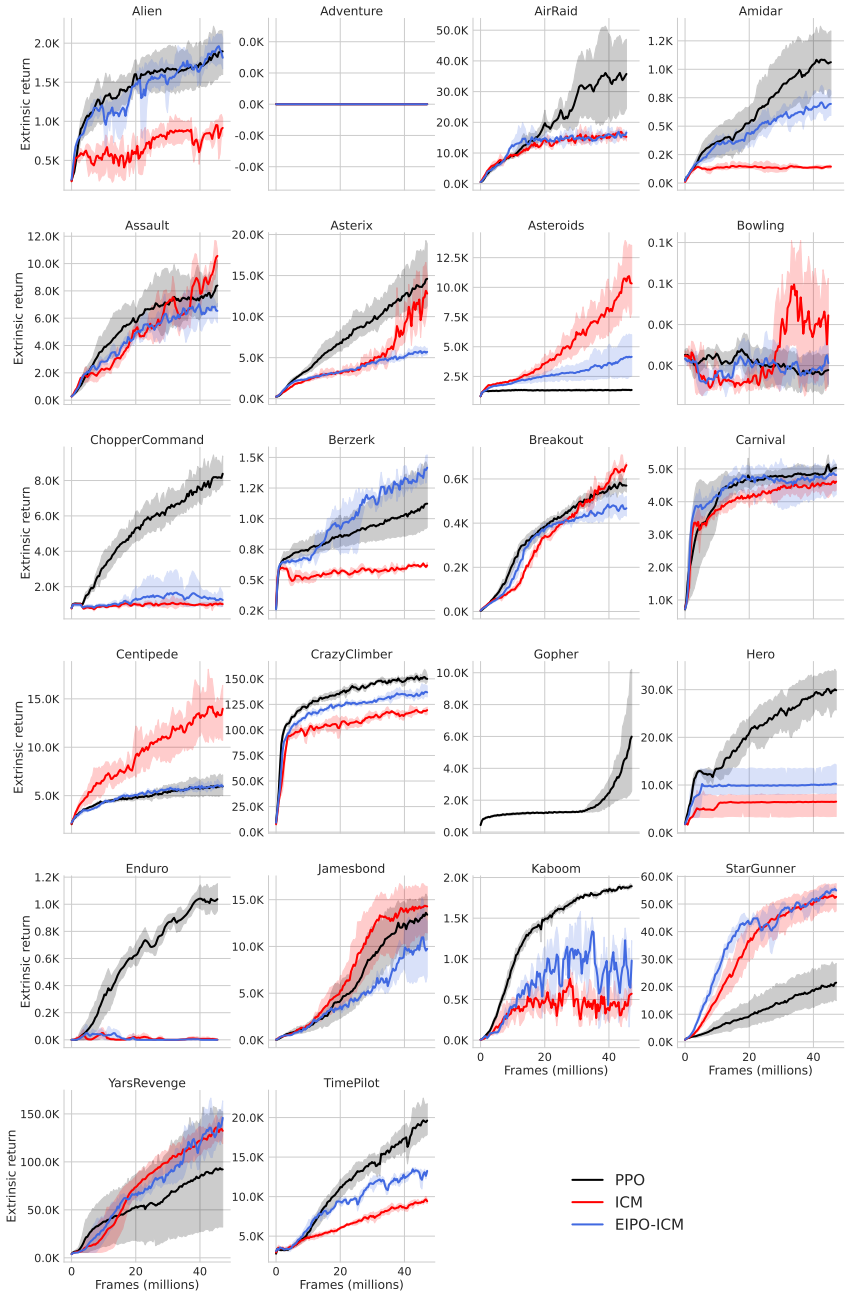


Figure 9: EIPO-ICM successfully matches ICM when it outperforms PPO, and closes the gap with PPO when ICM under-performs in some games. Note that we use the same α (Section 3.1) and β (Section 3.1) used in RND in this experiments. This suggests that even without tuning hyperparameters for ICM, EIPO is effective in some games. For instance, in *Kaboom*, the screen flashes a rapid sequence of bright colors when the agent dies, causing ICM to generate high intrinsic reward at these states. Even in such games where the intrinsic and extrinsic reward signals are misaligned, our method is able to shrink the performance gap. In extreme cases where the intrinsic and extrinsic rewards are steeply misaligned (*Enduro*), our methods inability to completely turn off the effects of intrinsic rewards results in subpar performance. On the same environment however, we see that RND does perform well (Fig. 6). This implies that we need more investigation to uncover the relationship between EIPO and intrinsic rewards functions.

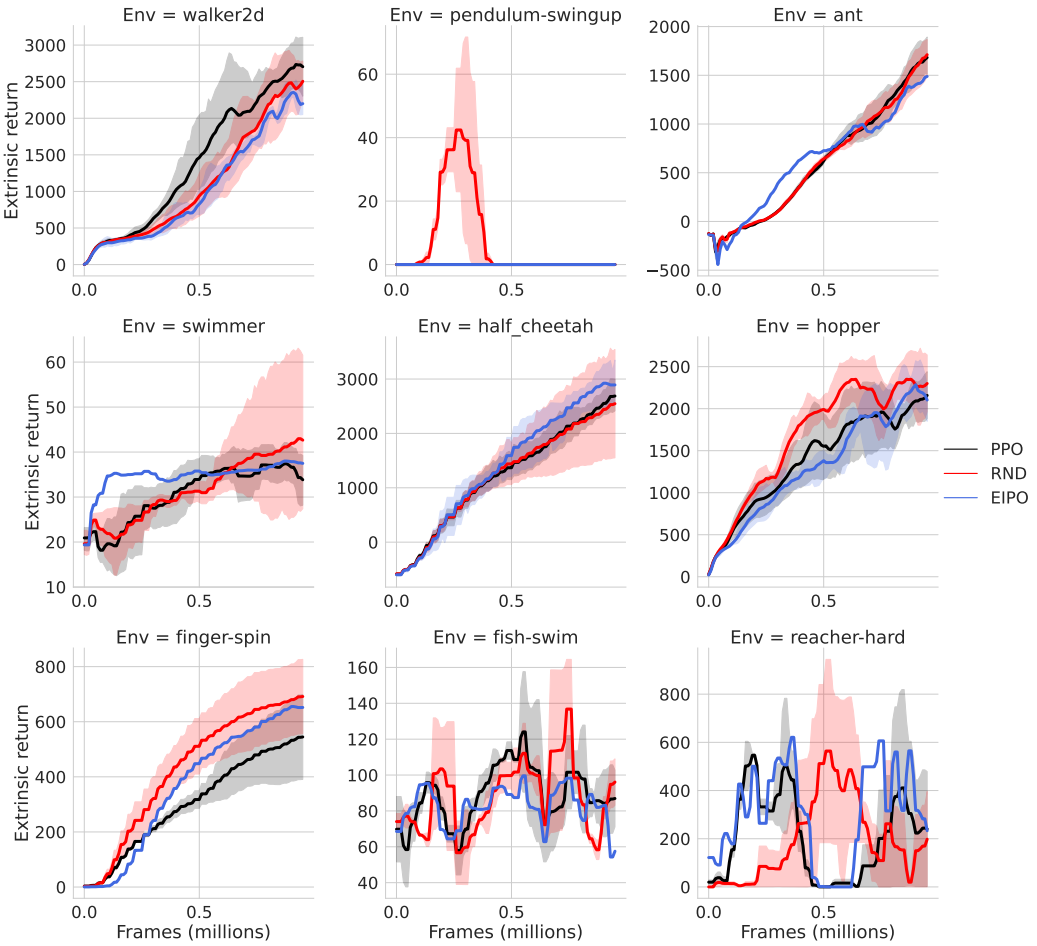


Figure 10: EIPO matches RND and PPO in all tasks in terms of the final performance. RND and PPO do not have visible difference since the reward functions in MuJoCo are often dense.

rewards are non-stationary and change over the training time, yet MORL approaches require the reward functions to be fixed. Also, MORL is aimed at learning a Pareto-optimal policy that satisfies all the objectives, but intrinsic rewards are just auxiliary training signals, and thus, we do not require maximizing intrinsic rewards.