Automatic Intrinsic Reward Shaping for Exploration in Deep Reinforcement Learning

Mingqi Yuan [1] Bo Li [1] Xin Jin [2] Wenjun Zeng [2]

Abstract

We present AIRS: **A**utomatic **I**ntrinsic **R**eward **S**haping that intelligently and adaptively provides high-quality intrinsic rewards to enhance exploration in reinforcement learning (RL). More specifically, AIRS selects shaping function from a predefined set based on the estimated task return in real-time, providing reliable exploration incentives and alleviating the biased objective problem. Moreover, we develop an intrinsic reward toolkit [1] to provide efficient and reliable implementations of diverse intrinsic reward approaches. We test AIRS on various tasks of MiniGrid, Procgen, and DeepMind Control Suite. Extensive simulation demonstrates that AIRS can outperform the benchmarking schemes and achieve superior performance with simple architecture.

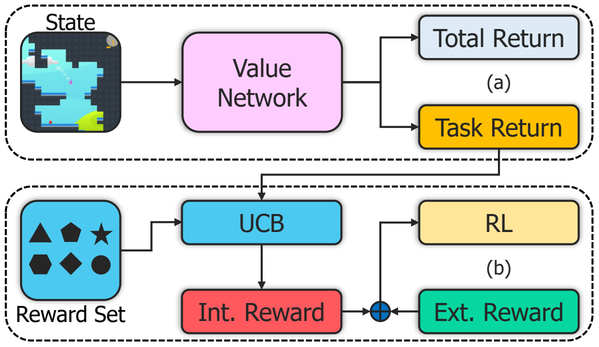

Figure 1. (a) AIRS employs two branches in the value network to disentangle the estimation of total return (evaluated by intrinsic and extrinsic reward function) and task return (only extrinsic reward function). (b) AIRS formulates the intrinsic reward selection as a multi-armed bandit problem and uses the upper confidence bound (UCB) to make decisions based on the estimated task return. **Ext.**: Extrinsic. **Int.**: Intrinsic. $\bigoplus$: Weighted summation operation.

1. Introduction

Striking an appropriate balance between exploration and exploitation remains a long-standing problem in reinforcement learning (RL) (Sutton & Barto, 2018). Sufficient exploration can prevent the RL agent from prematurely falling local optima after finite iterations, contributing to learning better policies (François-Lavet et al., 2018). To address this problem, classical exploration strategies such as ϵ-greedy or Boltzmann exploration will randomly choose all the possible actions with a non-zero probability (Mnih et al., 2015). But these approaches are inefficient when handling complex environments with high-dimensional observations. Moreover, extrinsic rewards are consistently found to be sparse or even absent in many real-world scenarios, and they may completely fail to learn.

Recent approaches have proposed to leverage intrinsic rewards to encourage exploration (Oh et al., 2015; Houthooft et al., 2016; Bellemare et al., 2016; Pathak et al., 2017; Haber et al., 2018; Ostrovski et al., 2017). For instance, (Bellemare et al., 2016) leverage a density model to approximate the state visitation frequency and define the intrinsic reward as inversely proportional to the pseudo-count. As a result, the agent is encouraged to visit the infrequently-seen states, increasing the probability of encountering states with higher task rewards. In contrast, curiosity-driven exploration aims to learn the dynamics of the environment and utilizes the prediction error as the intrinsic reward (Stadie et al., 2015; Yu et al., 2020; Burda et al., 2019b). For example, (Pathak et al., 2017) use an inverse-forward dynamics model to learn the representation of state space, which only encodes the part that affects the decision-making and ignores environment noise and other irrelevant interference. After that, the intrinsic reward is defined as the prediction error of the encoded next-state based on the current state-action pair.

Despite the excellent performance of intrinsic rewards, they cannot guarantee the invariance of the optimal policy, and

[1]Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China [2]Eastern Institute for Advanced Study, Zhejiang, China. Correspondence to: Xin Jin <jinxin@eias.ac.cn>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

[1]https://github.com/RLE-Foundation/rllte

excessive exploration may lead to learning collapse (Ng et al., 1999). Since the joint objective composed of extrinsic and intrinsic rewards is biased, maximizing it only sometimes yields the optimal policy for the extrinsic reward alone. To alleviate this problem, (Chen et al., 2022) proposed an extrinsic-intrinsic policy optimization (EIPO) approach, which automatically tunes the importance of the intrinsic reward via a principled constrained policy optimization procedure. As a result, EIPO can restrain the intrinsic reward when exploration is unnecessary and increase it when exploration is required. However, EIPO is too sophisticated to implement and cannot provide increments for arbitrary RL algorithms. Moreover, many experiments reported in existing work (Burda et al., 2019a;a; Raileanu et al., 2020; Badia et al., 2020; Chen et al., 2022) demonstrate that the performance of the same intrinsic rewards varies significantly in different tasks and learning stages. Selecting the best intrinsic reward for a specific task is always challenging. With this in mind, (Zheng et al., 2018) proposed a stochastic gradient-based method to learn parametric intrinsic rewards, which improved the performance of policy gradient-based algorithms.

In this paper, we solve the aforementioned problems using a novel framework entitled **A**utomatic **I**ntrinsic **R**eward **S**haping (AIRS). Our main contributions can be summarized as follows:

- Firstly, we formulate the intrinsic reward selection as a multi-armed bandit problem, in which different intrinsic reward functions are regarded as arms. AIRS can automatically select the best shaping function at different learning stages based on the estimated task return, providing reliable exploration incentives and alleviating the biased objective problem.
- Secondly, we develop a toolkit that provides high-quality implementations of various intrinsic reward modules based on PyTorch. These modules can be deployed in arbitrary RL algorithms in a plug-and-play manner, providing efficient and robust exploration increments.
- Finally, we test AIRS on MiniGrid, Procgen (*sixteen games with procedurally-generated environments*), and DeepMind Control Suite. Extensive simulation results demonstrate that AIRS can achieve superior performance and generalization ability and outperform the benchmarking schemes.

2. Related Work

2.1. Count-Based Exploration

Count-based exploration provides intrinsic rewards by measuring the novelty of states, which are usually defined to be inversely proportional to the state visit counts (Bellemare et al., 2016; Ostrovski et al., 2017; Tang et al., 2017; Machado et al., 2020; Jo et al., 2022). (Strehl & Littman, 2008) proposed to use the state visit counts as exploration bonuses in the tabular setting and provided a theoretical explanation for its effectiveness. (Bellemare et al., 2016) designed a context-tree switching density model to perform state pseudo-count, and (Ostrovski et al., 2017) considered the environments with high-dimensional observations and used a PixelCNN as a state density estimator. (Martin et al., 2017) proposed to perform pseudo-count by leveraging the learned feature space of value function approximation, which can evaluate the uncertainty associated with any state. (Burda et al., 2019b) proposed a random-network-distillation (RND) method composed of a predictor network and a target network, using the prediction error to reward novel states.

2.2. Curiosity-Driven Exploration

Curiosity-driven exploration encourages the agent to increase its knowledge (e.g., dynamics) about the task environment (Stadie et al., 2015; Pathak et al., 2017; Yu et al., 2020). The most well-known work is the intrinsic-curiosity-module (ICM) proposed by (Pathak et al., 2017). However, (Yu et al., 2020) suggested that the discriminative model may suffer from the compounding error and redesigned the architecture of ICM using a variational auto-encoder (Kingma & Welling, 2014), which circumvents the encoding of state space and can be trained using a one-life demonstration. In particular, (Burda et al., 2019a) attempted to solve visual tasks (e.g., Atari games) using only curiosity-based intrinsic reward, and the agent could still achieve remarkable performance. But the curiosity-driven methods are consistently found to be futile when handling the noisy-TV dilemma (Savinov et al., 2019). To address the problem, (Raileanu et al., 2020) proposed a rewarding-impact-driven-exploration (RIDE) method that uses the difference between two consecutive encoded states as the intrinsic reward and encourages the agent to choose actions that result in significant state changes. In this paper, the RIDE is selected as the candidate for the intrinsic reward set. It can provide aggressive exploration incentives and oblige the agent to adapt to the environment quickly, improving the generalization ability of the agent and facilitating solving tasks with procedurally-generated environments.

2.3. Information Theory-Based Exploration

Another representative idea is to design intrinsic rewards based on information theory (Seo et al., 2021; Mutti et al., 2021; Yuan et al., 2022c;b;a). (Houthooft et al., 2016) proposed to maximize the information gain about the agent's belief of environment dynamics, which uses variational inference in Bayesian neural networks to handle continuous

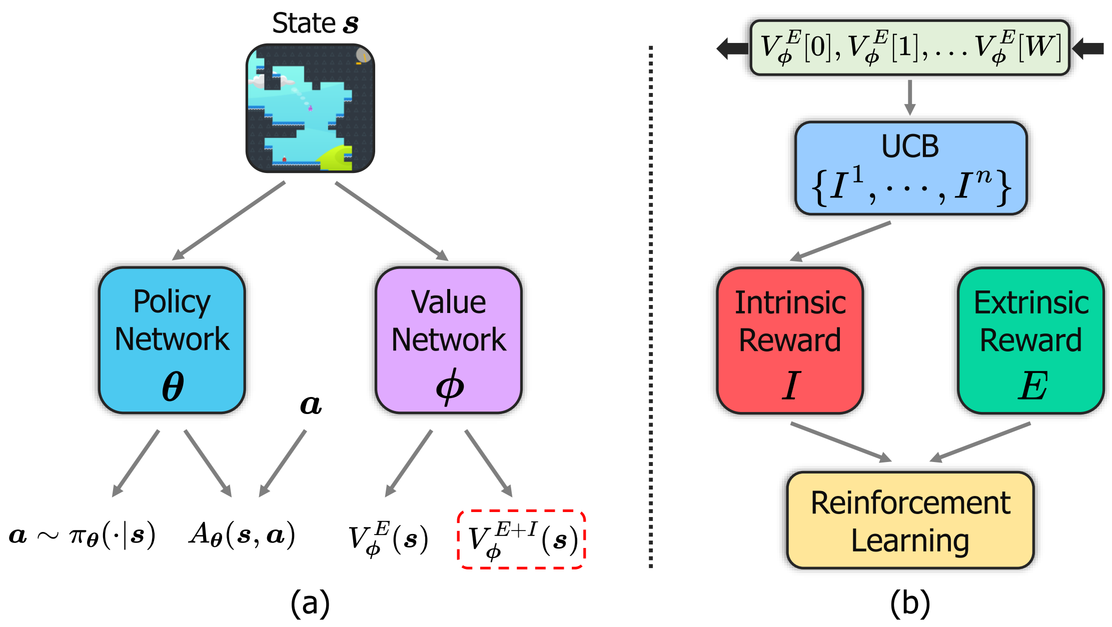

Figure 2. The overview of AIRS. (a) The architecture of DAAC. Here, we use two branches in the value network to estimate $V_\phi^E(s)$ and $V_\phi^{E+I}(s)$. (b) AIRS selects the intrinsic reward function based on a queue (length=W) of estimated task return.

state and action spaces efficiently. (Kim et al., 2019) proposed to maximize the mutual information between related state-action representations and define the intrinsic reward using prediction error under a linear dynamics model. (Seo et al., 2021) proposed to maximize the Shannon entropy of state visit distribution and designed a random-encoder-for-efficient-exploration (RE3) method. RE3 leverages a k-nearest neighbor estimator to estimate entropy efficiently and transforms the sample mean into particle-based intrinsic rewards. (Yuan et al., 2022c) further extended RE3 and proposed a Rényi state entropy maximization (RISE) method, which prevents the agent from visiting some states with a vanishing probability. In this paper, we selected RE3 and RISE as the candidates for the intrinsic reward set, providing powerful exploration incentives from the information theory perspective.

3. Background

3.1. Preliminaries

In this paper, we study the RL and control problems considering a Markov Decision Process (MDP) (Bellman, 1957) defined by a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, E, P, \rho(s_0), \gamma \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $E : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the extrinsic reward function that evaluates the actual task reward, $P(s'|s, a) : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the state-transition function that defines a probability distribution over $\mathcal{S}$, $\rho(s_0)$ is the initial state distribution, and $\gamma \in [0, 1)$ is a discount factor. Denoting by $\pi_\theta(a|s)$ the policy of the agent, the objective of RL is to learn a policy that maximizes the expected discounted return $J_E(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t E_t\right]$ (Sutton & Barto, 2018).

In the following sections, we leverage intrinsic rewards to improve the exploration capability of the agent. Letting $I : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ denote the intrinsic reward function, and the optimization objective is redefined as

$$J_{E+I}(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t (E_t + \beta_t \cdot I_t)\right], \tag{1}$$

where $\beta_t = \beta_0(1 - \kappa)^t$ is a weighting coefficient that controls the degree of exploration, and κ is a decay rate.

3.2. Decoupled Advantage Actor-Critic

Decoupled advantage actor-critic (DAAC) is a state-of-the-art on-policy algorithm that decouples the policy and value optimization for generalization in RL (Raileanu & Fergus, 2021). As shown in Figure 2a, DAAC includes two separate networks, one for learning the policy and advantage and one for learning the state-value function. More specifically, the policy network of DAAC is trained to maximize the following objective:

$$J_{\text{DAAC}}(\theta) = J_\pi(\theta) + H_\pi(\theta) - L_A(\theta), \tag{2}$$

where J_π is the policy gradient term of the proximal policy optimization (PPO) (Schulman et al., 2017), H_π is an entropy bonus to encourage exploration, and $L_A = [A_{\boldsymbol{\theta}}(\boldsymbol{s}_t, \boldsymbol{a}_t) - \hat{A}_t]^2$ is the advantage loss, and $\hat{A}_t$ is the corresponding generalized advantage estimate at time step t (Schulman et al., 2015). Finally, the value network of DAAC is trained to minimize the following loss:

$$L_V(\boldsymbol{\phi}) = \left[V_{\boldsymbol{\phi}}(\boldsymbol{s}_t) - \hat{V}_t\right]^2, \tag{3}$$

where $\hat{V}_t$ is the total discounted return obtained during the corresponding episode after time step t.

4. AIRS

In this section, we propose the AIRS framework that improves the exploration and generalization ability of the RL agent by intelligently providing high-quality intrinsic rewards. Our critical insight is that different tasks and learning stages may benefit from distinct intrinsic reward functions. For instance, in the early stage of learning, aggressive exploration can enable agents to gain a lot of discriminative experiences in a short time. To that end, the simple idea is to design an intrinsic reward function that rewards significant state changes. However, in the later stage of learning, excessive exploration is unnecessary and may disturb the learned policy, and it suffices to keep appropriate exploration. The discussions above can be summarized as the following problems:

- Use or not use intrinsic reward function?
- Use which intrinsic reward function?

Denoting by $\mathcal{I} = \{I^1, \dots, I^n\}$ the set of intrinsic reward functions, the problem of intrinsic reward selection can be formulated as a multi-armed bandit (Figure 2b) (Lattimore & Szepesvári, 2020). Each reward function is considered an arm, and the objective is to maximize the long-term return evaluated by the extrinsic reward function.

4.1. Upper Confidence Bound

Upper confidence bound (UCB) (Auer, 2002) is an effective algorithm to solve the multi-armed bandit problem, which selects actions by the following policy:

$$I_t = \underset{I \in \mathcal{I}}{\operatorname{argmax}} \left[Q_t(I) + c\sqrt{\frac{\log t}{N_t(I)}}\right], \tag{4}$$

where $Q_t(I)$ is the estimated value of action I, $N_t(I)$ is the number of times that action I has been selected prior to time t, and $c > 0$ is a constant that controls the degree of exploration. To update Q_t, we employ two branches (Figure 2a) in the value network to estimate the state-value function evaluated by the mixed reward function $E + I$ and extrinsic reward function E, respectively. For each intrinsic reward function I, a queue of length W is leveraged to store the average estimated task return, and $Q_t(I)$ is computed as

$$Q_t(I) = \frac{1}{W}\sum_{i=1}^{W} \bar{V}^E_{\boldsymbol{\phi}}[i], \tag{5}$$

where $\bar{V}^E_{\boldsymbol{\phi}}$ is the average estimated return of a rollout. Meanwhile, the loss function of the value network is redefined as

$$\begin{aligned} L_V(\boldsymbol{\phi}) = &\left[V^{E+I}_{\boldsymbol{\phi}}(\boldsymbol{s}_t) - \hat{V}^{E+I}_t\right]^2 \\ &+ \left[V^{E}_{\boldsymbol{\phi}}(\boldsymbol{s}_t) - \hat{V}^{E}_t\right]^2. \end{aligned} \tag{6}$$

Finally, the detailed workflow of AIRS based on DAAC is summarized in Algorithm 1. We want to highlight two critical facts about AIRS. Here we use DAAC as the benchmark due to its improvement in generalization ability. But AIRS can be combined with many RL algorithms, and it suffices to use two branches in the value network to estimate task and total return, respectively. In addition, any appropriate bandit algorithms can replace the UCB algorithm. For example, the Thompson sampling (Russo et al., 2018) can be deployed when the intrinsic reward pool only has two reward functions. Therefore, AIRS is a very open architecture, and various attempts are possible.

4.2. Intrinsic Reward Toolkit

To facilitate the experiments and inspire subsequent research on intrinsic rewards, we developed a toolkit that provides high-quality implementations of diverse intrinsic reward modules based on PyTorch. This toolkit is designed to be highly modular and scalable. Each intrinsic reward module can be deployed in arbitrary algorithms in a plug-and-play manner, providing efficient and robust exploration increments. The following experiments are also performed based on this toolkit. More details about this toolkit can be found in Appendix D.

5. Experiments

In this section, we designed experiments to answer the following questions:

- Can AIRS alleviate the aforementioned biased objective problem? (See Figure 3)
- Can AIRS improve policy performance in both continuous and discrete control tasks? (See Table 5, Table 6, Figure 3, and Figure 9)
- How does AIRS compare to single intrinsic reward-driven approaches? (See Figure 4, Figure 5, Figure 6, Table 5, and Table 6)

Algorithm 1 Automatic Intrinsic Reward Shaping (AIRS)

```
1: Initialize policy network $\pi_{\boldsymbol{\theta}}$ and value network $V_{\boldsymbol{\phi}}$ with parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$;
2: Initialize the set of intrinsic reward functions $\mathcal{I} = \{I^1, \dots, I^n\}$, exploration coefficient $c$, window length $W$ for estimating the Q-functions, and total number of updates $K$;
3: $N(I) = 1, \forall I \in \mathcal{I}$;        ▷ Initialize the number of times each $I$ was selected.
4: $Q(I) = 0, \forall I \in \mathcal{I}$;        ▷ Initialize the Q-functions for all $I$.
5: $R(I) = \mathrm{FIFO}(W), \forall I \in \mathcal{I}$;        ▷ Initialize the lists of returns for all $I$.
6: for $k = 1, \dots, K$ do
7:     $I_k = \underset{I \in \mathcal{I}}{\operatorname{argmax}} \left[ Q(I) + c\sqrt{\frac{\log k}{N(I)}} \right]$;        ▷ Use UCB to select an $I$.
8:     Collect $\{(\boldsymbol{s}_t, \boldsymbol{a}_t, E_t, \boldsymbol{s}_{t+1})\}_{t=1}^{T}$ using $\pi_{\boldsymbol{\theta}}$;
9:     Use $I_k$ to compute intrinsic rewards;
10:    Compute the value and advantage targets $\hat{V}_t^E, \hat{V}_t^{E+I}$ and $\hat{A}_t$ for all states $\boldsymbol{s}_t$;
11:    $\boldsymbol{\theta} \leftarrow \underset{\boldsymbol{\theta}}{\operatorname{argmax}} J_{\mathrm{DAAC}}$;        ▷ Update the policy network.
12:    $\boldsymbol{\phi} \leftarrow \underset{\boldsymbol{\phi}}{\operatorname{argmax}} L_V$;        ▷ Update the value network.
13:    Compute the mean return $\bar{V}_{\boldsymbol{\phi}}$ obtained by the new policy;
14:    Add $\bar{V}_{\boldsymbol{\phi}}$ to the $R(I_k)$ list using the first-in-first-out rule;
15:    $Q(I_k) \leftarrow \frac{1}{|R(I_k)|} \sum_{\bar{V}_{\boldsymbol{\phi}} \in R(I_k)} \bar{V}_{\boldsymbol{\phi}}$;
16:    $N(I_k) \leftarrow N(I_k) + 1$.
17: end for
```

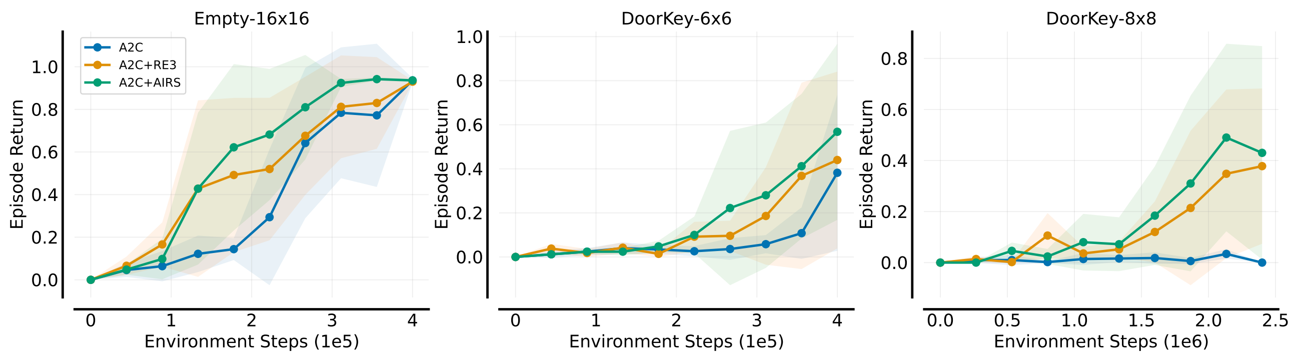

Figure 3. Sample efficiency comparison between AIRS and benchmarks. The solid line and shaded regions represent the mean and standard deviation, which are computed over five random seeds.

- Can the redesigned value network make effective value estimation? (See Figure 14)
- What is the detailed decision process of AIRS during training? (See Figure 7)

In particular, we report all the experiment results using a reasonable and reliable toolbox entitled rliable to guarantee reliability and reduce the uncertainty (Agarwal et al., 2021). Four metrics are introduced in the following sections, **Median**: Aggregate median performance, higher is better. **IQM**: Aggregate interquartile mean performance, higher is better. **Mean**: Aggregate mean performance, higher is better. **Optimality Gap**: The amount by which the algorithm fails to meet a minimum score, lower is better.

5.1. MiniGrid Games

5.1.1. Setup

We first tested AIRS on the MiniGrid benchmark to highlight the effectiveness of automatic intrinsic reward shaping (Chevalier-Boisvert et al., 2018). MiniGrid contains a collection of 2D grid-world environments with goal-oriented and sparse-reward tasks that are extremely hard-exploration. We selected advantage actor-critic (A2C) (Mnih et al., 2016) as the baseline and considered the combination of A2C and RE3. For AIRS, we build it on top of A2C+RE3, in which the intrinsic reward function pool only has ID and RE3. See Table 1 for the details of the two functions. In particular, we added the intrinsic rewards into estimated advantages

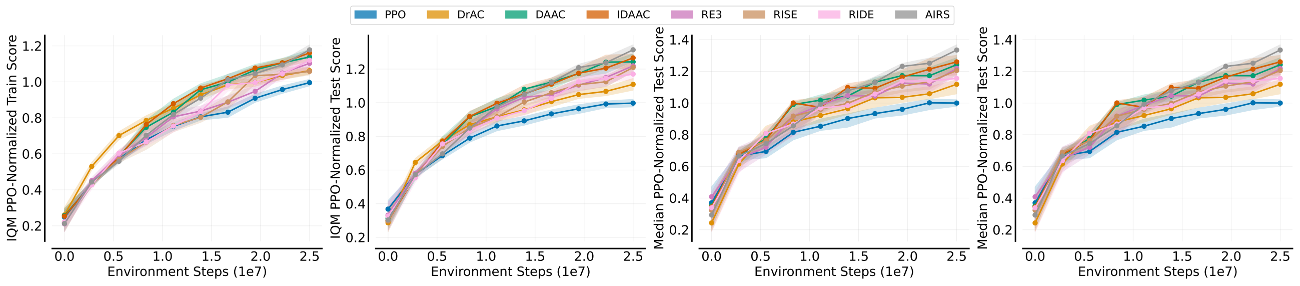

Figure 4. Comparing Median vs IQM on train and test levels. Sample-efficiency of agents as a function of number of frames measured via **IQM** (two on the left) and **Median** (two on the right) PPO-Normalized scores. Shaded regions show pointwise 95% percentile stratified bootstrap confidence intervals.

Table 1. Four intrinsic reward functions are employed in the Procgen experiments, where $\boldsymbol{e} = g(\boldsymbol{s})$ is the representation of $\boldsymbol{s}$, $N_{\rm ep}$ is the episodic state visitation frequency. For RE3 and RISE, g is a random and fixed encoder, while g is a learned encoder for RIDE. $\tilde{\boldsymbol{e}}$ is the k-nearest neighbor of $\boldsymbol{s}$ in the encoding space. More details about the implementations can be found in Appendix B.

Intrinsic reward module	**Formulation**	**Remark**
RE3 (Seo et al., 2021)	$I_t = \frac{1}{k}\sum_{i=1}^{k}\log(\Vert\boldsymbol{e}_t - \tilde{\boldsymbol{e}}_t^i\Vert_2 + 1)$	Shannon entropy maximization
RISE (Yuan et al., 2022c)	$I_t = \frac{1}{k}\sum_{i=1}^{k}(\Vert\boldsymbol{e}_t - \tilde{\boldsymbol{e}}_t^i\Vert_2)^{1-\alpha}$	Rényi entropy maximization
RIDE (Raileanu et al., 2020)	$I_t = \Vert\boldsymbol{e}_{t+1} - \boldsymbol{e}_t\Vert_2/\sqrt{N_{\rm ep}(\boldsymbol{s}_{t+1})}$	Significant state changes
ID	$I_t = 0$	No shaping

directly rather than using an additional branch in the value network. For each benchmark, we only report the best results after a hyper-parameter search, and more details on MiniGrid experiments are provided in Appendix A.

5.1.2. Results

Figure 3 illustrates the sample efficiency comparison between AIRS and benchmarking schemes. A2C+RE3 successfully outperformed the vanilla A2C agent in all three games, demonstrating that RE3 can effectively promote the sample efficiency using the same environment steps. In contrast, A2C+AIRS achieves the highest performance and convergence rate in all three games. This indicates that AIRS successfully alleviated the biased objective problem and controlled the exploration degree intelligently in the training process.

5.2. Procgen Games

5.2.1. Setup

Next, we tested AIRS on the full Procgen benchmark containing sixteen games with procedurally-generated environments (Cobbe et al., 2020). Procgen is developed similarly to the ALE benchmark, in which the agents have to learn motor control directly from images (Bellemare et al., 2013). But Procgen has higher requirements for the generalization ability, and the agent has to make sufficient exploration and learn transferable skills rather than memorizing specific trajectories. These attributes make it a good benchmark for our research. All Procgen games utilize a discrete action space with 15 possible actions and produce $64 \times 64 \times 3$ RGB observations. The agents were trained on the easy level and tested on the entire distribution of levels.

To construct the intrinsic reward set, four intrinsic reward functions were selected to serve as the candidates, namely RE3, RISE, RIDE, and ID, respectively (Seo et al., 2021; Yuan et al., 2022c; Raileanu et al., 2020). The detailed information of these functions is illustrated in Table 1. The reasons for selection are as follows: (i) they can provide sustainable exploration incentives, *i.e.*, the intrinsic rewards will not vanish with visits; (ii) they are computation-efficeint and easy to implement, and (iii) they can leverage fixed and random encoder to encode the state space, which provides relatively stable intrinsic reward space and guarantees the algorithm convergence.

Furthermore, we selected data-regularized actor-critic (DrAC), IDAAC, DAAC, and PPO to serve as the benchmarks (Raileanu et al., 2021; Raileanu & Fergus, 2021; Schulman et al., 2017). DrAC combines data augmentation

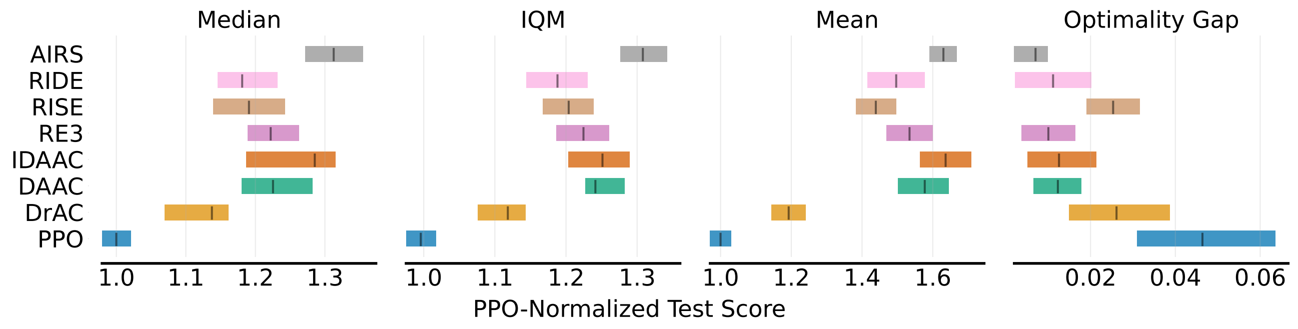

Figure 5. PPO-Normalized test performance for the Procgen benchmark, which are aggregated over all sixteen tasks and ten randoms seeds.

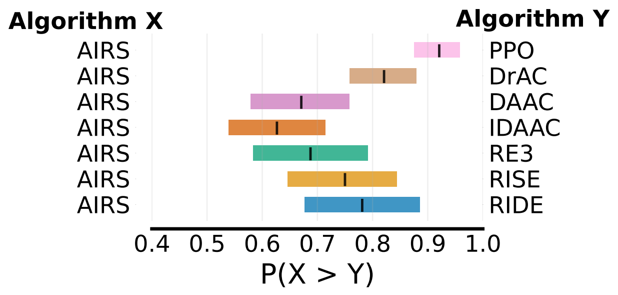

Figure 6. Each row shows the probability of improvement, with 95% bootstrap confidence intervals, that the algorithm X on the left outperforms algorithm Y on the right, given that X was claimed to be better than Y. For all algorithms, results are based on ten random seeds per task.

and actor-critic algorithms theoretically, demonstrating remarkable performance on Procgen benchmark. IDAAC is a variant of DAAC that adds an additional regularizer to the DAAC policy encoder to guarantee that it only contains episode-specific information. For each benchmark method, we only report its best performance after a hyper-parameter search, and more details on Prcogen games are provided in Appendix B.

5.2.2. Results

The game complexity of test levels is much higher than the train levels, which provides a direct measure to evaluate the generalization ability of the agents. Figure 5 illustrates the test performance comparison evaluated by four metrics. AIRS achieved the highest performance in all four metrics, especially for the **Median** and **IQM**. In contrast, IDAAC achieved the second highest performance in all four metrics. The performance of IDAAC and DAAC is very close in most games. This indicated that the additional regularization term did not significantly improve the performance of DAAC, which was the same as the results reported in (Raileanu & Fergus, 2021). Moreover, the combination of DAAC and single intrinsic rewards achieved lower performance than the vanilla DAAC agent. Furthermore, we computed the probability of improvement of AIRS over the benchmarks. Figure 6 indicates that the probability of AIRS being better than DAAC is 67%, and the probability of AIRS being better than IDAAC is 61%. We also provided a complete list of final performance comparisons in Table 5 and Table 6.

Figure 4 illustrates the sample efficiency comparison between AIRS and benchmarking schemes on the full Procgen benchmark. AIRS had a lower convergence rate in the early stage, which was mainly limited by the estimation accuracy of the value network. Figure 8 illustrates the value loss of eight games during training, and full loss curves can be found in Appendix B. It is evident that the loss of the two branches tended to stabilize after 1M environment steps. But AIRS still achieved remarkable performance gains. This indicates that AIRS can effectively improve the exploration of the agent and keep tracking the fundamental objective, *i.e.*, the task reward.

Furthermore, we documented the specific decision-making process of AIRS. Figure 7 illustrates the cumulative number of times AIRS selects each intrinsic reward function over the training progress of eight games, and full curves can be found in Appendix B. In *BigFish* game, the RIDE was the most selected function while AIRS chose no shaping in 33.7% of the training time. In *Miner* game, the RISE was the most selected function while AIRS chose no shaping in 59.3% of the training time. Similarly, the RISE was also the most selected function in *Ninja* game, and AIRS chose no shaping in 66.5% of the training time. Meanwhile, AIRS divided the choice opportunity into three functions in *CoinRun* game and *BossFight* game. It is evident that no shaping played an essential role in the whole training process, and appropriate exploration can improve sample efficiency and gain higher performance. In contrast, AIRS

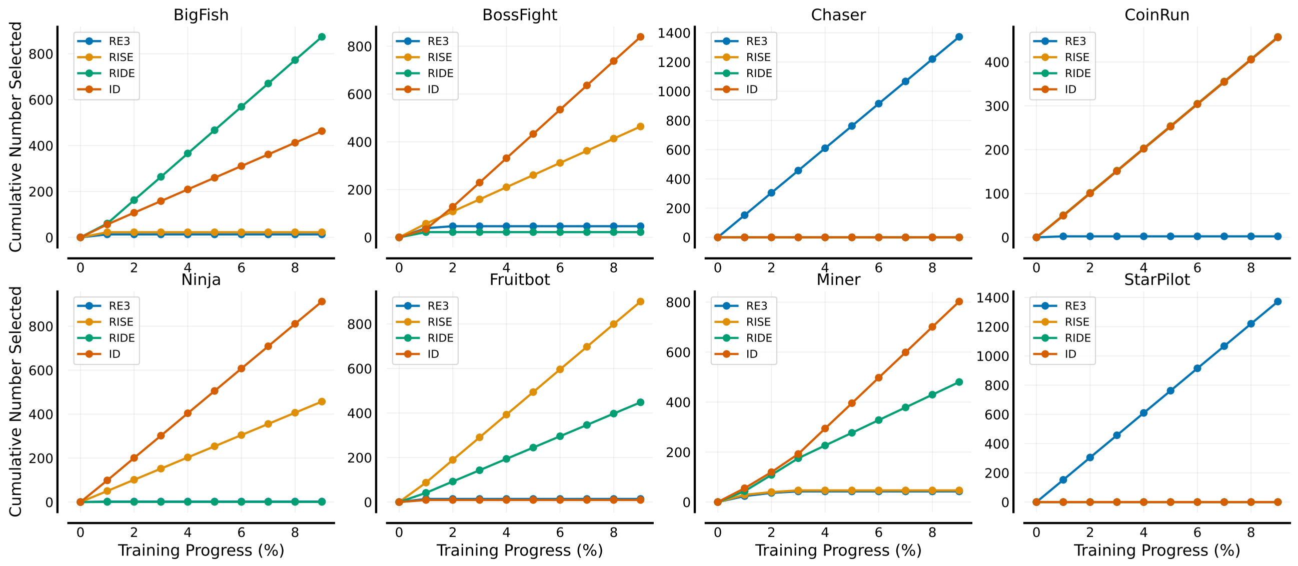

Figure 7. The cumulative number of times AIRS selects each intrinsic reward function over the training progress, computed with ten random seeds.

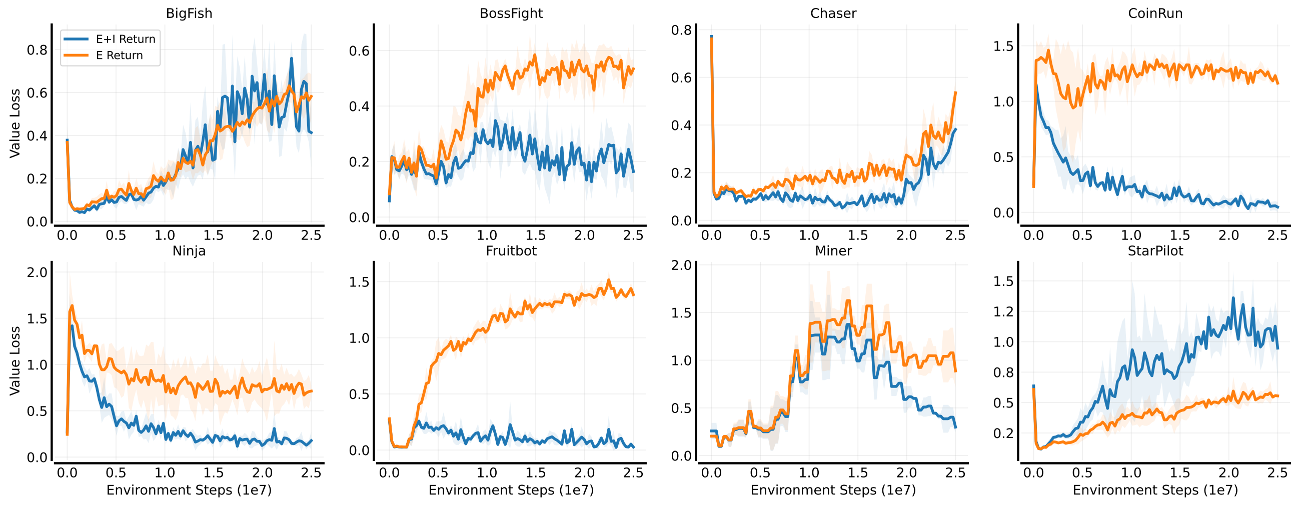

Figure 8. Value loss of eight selected games during training. "E Return" indicates the branch for extrinsic value estimation in the value network. Full loss curves can be found in in Appendix A. The solid line and shaded regions represent the mean and standard deviation over 10 random seeds, respectively.

selected RISE and RIDE at most in *Fruitbot* game. In *Chaser* and *StarPilot* game, AIRS only selected the RE3 function during the training, and the final performance of AIRS and DAAC+RE3 is almost the same.

5.2.3. Ablations

Like the MiniGrid experiments, we performed an ablation study by setting the intrinsic reward pool to have only two modules: RE3 and ID. As a result, the test **IQM** and **OG** are 1.27 and 0.015, which was better than DAAC+RE3. The ablation results further proved that AIRS can effectively alleviate the bias objective problem, providing appropriate exploration incentives when necessary. It also demonstrated that AIRS could assemble advantages by automatically selecting from multiple intrinsic rewards.

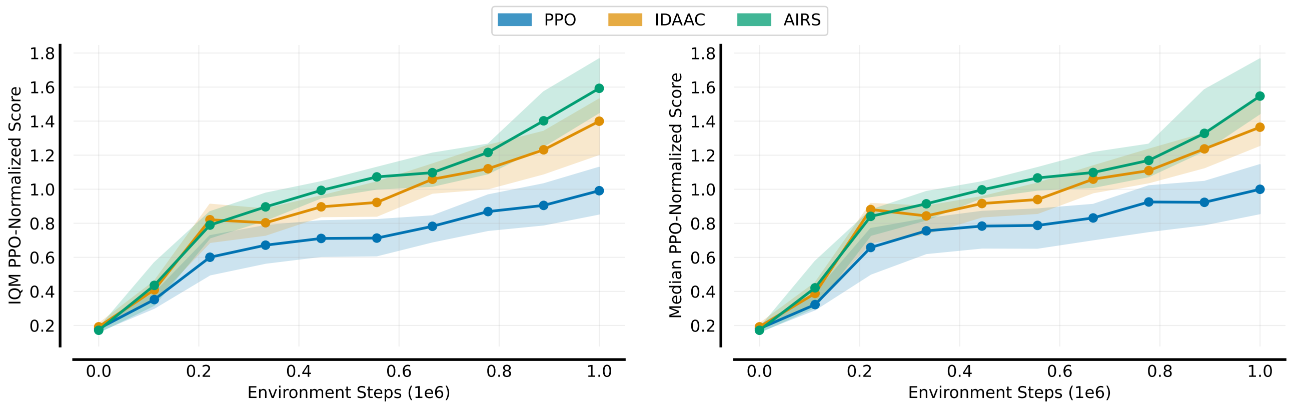

Figure 9. Sample-efficiency of agents as a function of number of frames measured via **IQM** and **Median** PPO-Normalized scores. Shaded regions show pointwise 95% percentile stratified bootstrap confidence intervals.

5.3. DeepMind Control Suite

5.3.1. Setup

Finally, we tested AIRS on the DeepMind Control Suite with distractors, and four tasks were introduced, namely *Cartpole Balance*, *Cheetah Run*, *Finger Spin*, and *Walker Walk*, respectively (Tassa et al., 2018). For each task, the backgrounds of image observations are replaced using natural videos from the Kinetics dataset (Cobbe et al., 2019). Note that the background is sampled from a list of videos at the beginning of each episode, which creates spurious correlations between backgrounds and rewards. For each benchmark, we only report the best results after a hyperparameter search, and more details on DeepMind Control Suite experiments are provided in Appendix C.

5.3.2. Results

Figure 9 illustrates the sample efficiency comparison between AIRS and benchmarking schemes after training 500K frames. AIRS achieved the highest performance in all the tasks, while IDAAC outperformed the vanilla PPO agent in all the tasks. Therefore, AIRS can improve the performance of RL agents on both discrete and continuous control tasks.

6. Discussion

In this paper, we investigate the problem of enhancing exploration in RL and propose an intrinsic reward-driven method entitled AIRS. AIRS can intelligently select the best intrinsic reward function from a pre-defined set in real-time, providing reliable exploration incentives and guaranteeing policy invariance. In particular, we develop a toolkit to provide high-quality implementations of diverse intrinsic reward approaches, which is expected to inspire more subsequent research. We test AIRS on multiple tasks from MiniGrid, Procgen, and DeepMind Control Suite. Extensive simulation results demonstrate that our method can achieve superior performance.

However, there are some remaining limitations of AIRS. Firstly, AIRS performs selection based on a pre-defined reward set. The quality of selected reward modules will inevitably affect the final policy performance. It's critical to find an effective method to build the reward set. Meanwhile, different intrinsic reward spaces may interfere with each other, resulting in unexpected learning collapse. Secondly, AIRS makes decisions based on the estimated task return, which requires an independent module to make estimations. The estimation accuracy also determines the reliability of actions of AIRS, and additional modules will increase the complexity of the deployment of AIRS. For algorithms like A2C and PPO, a simple method is to add the intrinsic rewards into estimated advantages directly rather than using an additional branch in the value network, which is used in our MiniGrid experiments. Finally, AIRS formulates the reward selection as a bandit problem and uses UCB to solve it, in which enough attempts are required to estimate the bound. This will significantly increase the computational overhead and decrease the convergence rate. We will address these issues in future work.

Acknowledgements

This work was supported, in part, by NSFC under Grant No. 62102333, HKSAR RGC under Grant No. PolyU 25211321, and ZJNSFC under Grant LQ23F010008. We also sincerely thank all the reviewers and editors for their hard work and insightful comments, which dramatically enhances the quality of our manuscript.

References

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. C., and Bellemare, M. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems*, 34:29304–29320, 2021.

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

Badia, A. P., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A., and Blundell, C. Never give up: Learning directed exploration strategies. In *Proceedings of the International Conference on Learning Representations*, 2020.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying count-based exploration and intrinsic motivation. *Proceedings of Advances in Neural Information Processing Systems*, 29:1471–1479, 2016.

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

Bellman, R. A markovian decision process. *Journal of mathematics and mechanics*, pp. 679–684, 1957.

Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., and Efros, A. A. Large-scale study of curiosity-driven learning. *Proceedings of the International Conference on Learning Representations*, pp. 1–17, 2019a.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. Exploration by random network distillation. *Proceedings of the 7th International Conference on Learning Representations*, pp. 1–17, 2019b.

Chen, E. R., Hong, Z.-W., Pajarinen, J., and Agrawal, P. Redeeming intrinsic rewards via constrained policy optimization. In *Advances in Neural Information Processing Systems*, 2022.

Chevalier-Boisvert, M., Willems, L., and Pal, S. Minimalistic gridworld environment for gymnasium, 2018. URL `https://github.com/Farama-Foundation/Minigrid`.

Cobbe, K., Klimov, O., Hesse, C., Kim, T., and Schulman, J. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, pp. 1282–1289. PMLR, 2019.

Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pp. 2048–2056. PMLR, 2020.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pp. 1407–1416. PMLR, 2018.

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., Pineau, J., et al. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4):219–354, 2018.

Glorot, X., Bordes, A., and Bengio, Y. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323. JMLR Workshop and Conference Proceedings, 2011.

Haber, N., Mrowca, D., Wang, S., Fei-Fei, L. F., and Yamins, D. L. Learning to play with intrinsically-motivated, self-aware agents. *Advances in neural information processing systems*, 31, 2018.

Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. Vime: Variational information maximizing exploration. *Proceedings of Advances in neural information processing systems*, 29:1109–1117, 2016.

Jo, D., Kim, S., Nam, D., Kwon, T., Rho, S., Kim, J., and Lee, D. Leco: Learnable episodic count for task-specific intrinsic reward. *Advances in Neural Information Processing Systems*, 35:30432–30445, 2022.

Kim, H., Kim, J., Jeong, Y., Levine, S., and Song, H. O. Emi: Exploration with mutual information. In *International Conference on Machine Learning*, pp. 3360–3369. PMLR, 2019.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *Proceedings of the International Conference on Learning Representations*, 2014.

Lattimore, T. and Szepesvári, C. *Bandit algorithms*. Cambridge University Press, 2020.

Machado, M. C., Bellemare, M. G., and Bowling, M. Count-based exploration with the successor representation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 5125–5133, 2020.

Martin, J., Sasikumar, S. N., Everitt, T., and Hutter, M. Count-based exploration in feature space for reinforcement learning. In *IJCAI*, 2017.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533, 2015.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937. PMLR, 2016.

Mutti, M., Pratissoli, L., and Restelli, M. Task-agnostic exploration via policy gradient of a non-parametric state entropy estimate. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 9028–9036, 2021.

Ng, A. Y., Harada, D., and Russell, S. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pp. 278–287, 1999.

Oh, J., Guo, X., Lee, H., Lewis, R. L., and Singh, S. Action-conditional video prediction using deep networks in atari games. *Advances in neural information processing systems*, 28, 2015.

Ostrovski, G., Bellemare, M. G., Oord, A., and Munos, R. Count-based exploration with neural density models. In *Proceedings of the International Conference on Machine Learning*, pp. 2721–2730, 2017.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 16–17, 2017.

Raileanu, R. and Fergus, R. Decoupling value and policy for generalization in reinforcement learning. In *International Conference on Machine Learning*, pp. 8787–8798. PMLR, 2021.

Raileanu, R., Rocktäschel, T., and Raileanu, R. Ride: Rewarding impact-driven exploration for procedurally-generated environments. In *Proceedings of the International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rkg-TJBFPB`.

Raileanu, R., Goldstein, M., Yarats, D., Kostrikov, I., and Fergus, R. Automatic data augmentation for generalization in reinforcement learning. *Advances in Neural Information Processing Systems*, 34:5402–5415, 2021.

Russo, D. J., Van Roy, B., Kazerouni, A., Osband, I., Wen, Z., et al. A tutorial on thompson sampling. *Foundations and Trends® in Machine Learning*, 11(1):1–96, 2018.

Ryan, R. M. and Deci, E. L. Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary educational psychology*, 25(1):54–67, 2000.

Savinov, N., Raichuk, A., Vincent, D., Marinier, R., Pollefeys, M., Lillicrap, T., and Gelly, S. Episodic curiosity through reachability. In *Proceedings of the International Conference on Learning Representations*, 2019.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. *Proceedings of the International Conference on Learning Representations*, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Seo, Y., Chen, L., Shin, J., Lee, H., Abbeel, P., and Lee, K. State entropy maximization with random encoders for efficient exploration. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 9443–9454, 2021.

Şimşek, Ö. and Barto, A. G. An intrinsic reward mechanism for efficient exploration. In *Proceedings of the 23rd International Conference on Machine Learning*, pp. 833–840, 2006.

Stadie, B. C., Levine, S., and Abbeel, P. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

Strehl, A. L. and Littman, M. L. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.

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

Tang, H., Houthooft, R., Foote, D., Stooke, A., Xi Chen, O., Duan, Y., Schulman, J., DeTurck, F., and Abbeel, P. # exploration: A study of count-based exploration for deep reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Yu, X., Lyu, Y., and Tsang, I. Intrinsic reward driven imitation learning via generative model. In *Proceedings of the International Conference on Machine Learning*, pp. 10925–10935, 2020.

Yuan, M., Jin, X., Li, B., and Zeng, W. Tackling visual control via multi-view exploration maximization. *arXiv preprint arXiv:2211.15233*, 2022a.

Yuan, M., Li, B., Jin, X., and Zeng, W. Rewarding episodic visitation discrepancy for exploration in reinforcement learning. In *Deep RL Workshop NeurIPS 2022*, 2022b.

Yuan, M., Pun, M.-O., and Wang, D. Rényi state entropy maximization for exploration acceleration in reinforcement learning. *IEEE Transactions on Artificial Intelligence*, 2022c.

Zhang, A., McAllister, R. T., Calandra, R., Gal, Y., and Levine, S. Learning invariant representations for reinforcement learning without reconstruction. In *International Conference on Learning Representations*, 2020.

Zheng, Z., Oh, J., and Singh, S. On learning intrinsic rewards for policy gradient methods. *Advances in Neural Information Processing Systems*, 31, 2018.

A. Details on MiniGrid Experiments

A.1. Environment Setting

In this section, we evaluated the performance of AIRS on three tasks from MiniGrid games (Chevalier-Boisvert et al., 2018), namely *Empty-16x16*, *DoorKey-6x6*, and *DoorKey-8x8*, respectively. The environment code can be found in the publicly available released repository (`https://github.com/Farama-Foundation/MiniGrid`). Figure 10 illustrates the screenshots of the three games. The girdworld was set to be fully observable, and a compact grid encoding was used to generate observations (7×7×3).

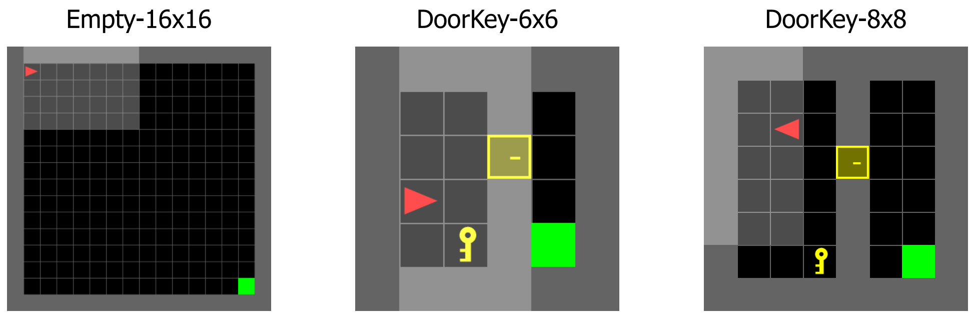

Figure 10. Screenshots of three MiniGrid games.

A.2. Experiment Setting

A2C. (Mnih et al., 2016) For A2C, we followed the implementation in the publicly released repository (`https://github.com/lcswillems/rl-starter-files`), and used their default hyperparameters listed in Table 2.

A2C+RE3. (Seo et al., 2021) For RE3, we followed the implementation in the publicly released repository (`https://github.com/younggyoseo/RE3`). Here, the intrinsic reward is computed as $I_t = \log(\|\boldsymbol{e}_t - \tilde{\boldsymbol{e}}_t\|_2 + 1)$, where $\boldsymbol{e}_t = g(\boldsymbol{s}_t)$ and g is a random and fixed encoder. The total reward of time step t is computed as $E_t + \beta_t \cdot I_t$, where $\beta_t = \beta_0(1-\kappa)^t$. Moreover, the average distance of $\boldsymbol{e}_t$ and its k-nearest neighbors was used to replace the single k nearest neighbor to provide a less noisy state entropy estimate. As for hyperparameters related to exploration, we used $k = 3, \kappa = 0$ for all three games, $\beta_0 = 0.1$ for Empty-16x16, $\beta_0 = 0.005$ for DoorKey-6x6, and $\beta_0 = 0.01$ for DoorKey-8x8, respectively.

A2C+AIRS. For AIRS, we built it on top of A2C+RE3, which uses same policy and value network architectures with A2C and A2C+RE3. We performed a hyperparameter search over the initial exploration degree $\beta_0 \in \{0.005, 0.01, 0.1\}$, the decay rate $\kappa \in \{0.0, 0.00001, 0.000025, 0.00005\}$, the exploration coefficient $c \in \{0.0, 0.1, 0.5, 1.0, 5.0\}$ and the size of sliding window used to compute the Q-values $W \in \{10, 50, 100\}$. We found that the best values are $\beta_0 = 0.1$ for Empty-16x16, $\beta_0 = 0.005$ for DoorKey-6x6, $\beta_0 = 0.01$ for DoorKey-8x8, $\kappa = 0$, $c = 0.1$, and $W = 10$, which were used to obtain the results reported here.

Table 2. General hyperparameters used to obtain the MiniGrid results.

Hyperparameter	**Value**
Observation downsampling	(7, 7, 3)
Stacked frames	No
Environment steps	400000 Empty-16x16, 400000 DoorKey-6x6, 2400000 DoorKey-8x8
Episode steps	5
Number of workers	1
Environments per worker	16
Optimizer	RMSprop
Learning rate	0.001
GAE coefficient	0.95
Action entropy coefficient	0.01
Value loss coefficient	0.5
Value clip range	0.2
RMSprop ϵ	0.01
Max gradient norm	0.5
Epochs per rollout	3
Mini-batches per epoch	8
LSTM	No
Gamma γ	0.99

B. Details on Procgen Experiments

B.1. Environment Setting

In this section, we evaluated the performance of AIRS on the full Procgen benchmark (Cobbe et al., 2020). All Procgen games utilize a discrete action space with 15 possible actions and produce $64 \times 64 \times 3$ RGB observations. Figure 11 illustrates the screenshots of multiple procedurally-generated levels from 12 Procgen environments. The environment code can be found in the publicly available released repository (`https://github.com/openai/procgen`). The agents were trained on a fixed set of 200 levels (*i.e.*, generated using seeds from 1 to 200) and tested on the full distribution of levels (*i.e.*, generated using randomly-sampled computer integers).

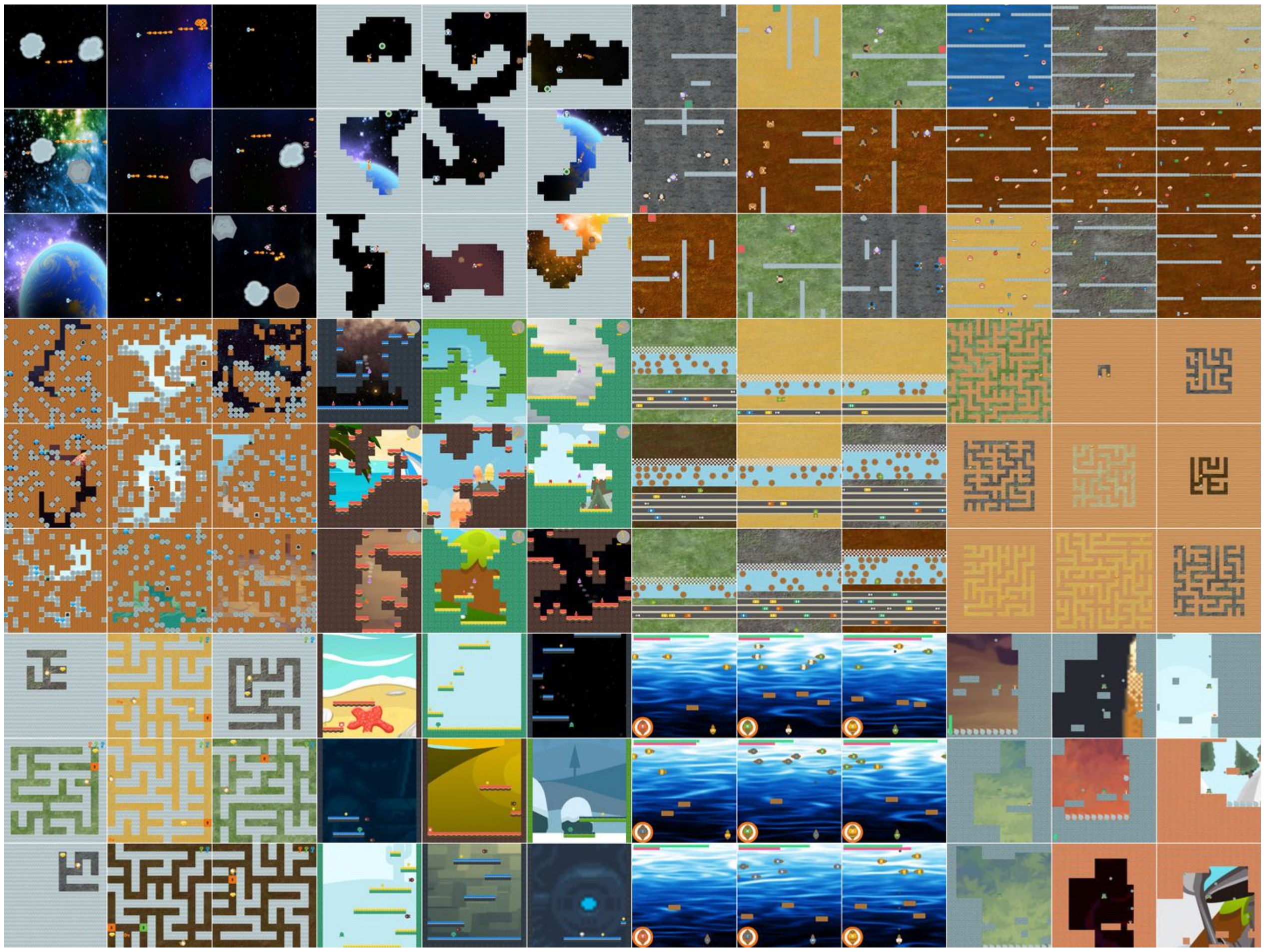

Figure 11. Screenshots of multiple procedurally-generated levels from 12 Procgen environments: StarPilot, CaveFlyer, Dodgeball, FruitBot, Miner, Jumper, Leaper, Maze, Heist, Climber, Plunder, Ninja (from left to right, top to bottom).

B.2. Experiment Setting

AIRS. In this experiment, we built AIRS on top of DAAC (Raileanu & Fergus, 2021) and employed two branches in the value network to predict the state-value function evaluated by the mixed reward function and extrinsic reward function, respectively. Figure 12 illustrates the ResNet-based architectures of the policy network and the value network (Espeholt et al., 2018). For each game, the agents were trained for 25M frames on the easy mode and tested using the entire distribution of levels.

We performed a hyperparameter search over the initial exploration degree $\beta_0 \in \{0.01, 0.05, 0.1\}$, the decay rate $\kappa \in \{0.00001, 0.000025, 0.00005\}$, the exploration coefficient $c \in \{0.0, 0.1, 0.5, 1.0, 5.0\}$ and the size of sliding window used

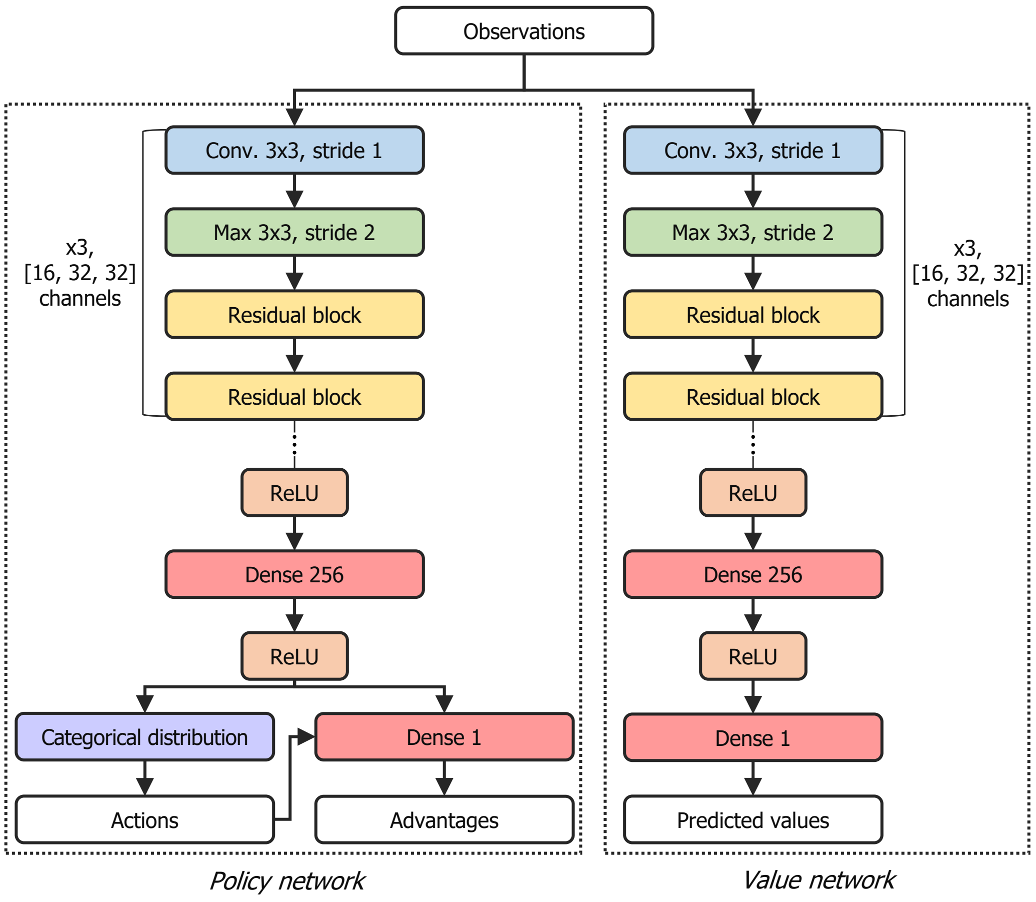

Figure 12. ResNet-based architectures of the policy network and the value network. Here the residual block consists of two convolutional layers with ReLU function (Glorot et al., 2011; He et al., 2016).

to compute the Q-values $W \in \{10, 50, 100\}$. We found that the best values are $\beta_0 = 0.05$, $\kappa = 0.000025$, $c = 0.1$, and $W = 10$, which were used to obtain the results reported here.

PPO. (Schulman et al., 2017) For PPO, we followed the implementation in the publicly released repository (`https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail`). To make a fair comparison, the agent was parameterized by the same ResNet architecture in Figure 12 to obtain the best results. More detailed hyperparameters are illustrated in Table 4.

DrAC. (Raileanu et al., 2021) For DrAC, we followed the implementation in the publicly released repository (`https://github.com/rraileanu/auto-drac`). DrAC was implemented based on the PPO repository above and added an augmentation loss term into the original PPO loss function. For each game, we used the best augmentation type reported in (Raileanu et al., 2021), which is illustrated in Table 3. The coefficient of the augmentation loss was 0.1, and the remaining hyperparameters were the same as Table 4.

DAAC. (Raileanu & Fergus, 2021) For DAAC, we followed the implementation in the publicly released repository (`https://github.com/rraileanu/idaac`). DAAC was also implemented based on PPO and used two separate neural networks to perform the policy and value optimization. As reported in (Raileanu & Fergus, 2021), the number of epochs used during each update of the policy network was 1, the number epochs used during each update of the value network was 9, the value update frequency was 1, the coefficient of the advantage loss was 0.25, and the remaining hyperparameters were the same as Table 4.

IDAAC. (Raileanu & Fergus, 2021) For IDAAC, we followed the implementation in the publicly released repository

Table 3. Augmentation type used for each game in DrAC.

Game	BigFish	StarPilot	FruitBot	BossFight	Ninja	Plunder	CaveFlyer	CoinRun
Augmentation	crop	crop	crop	flip	color-jitter	crop	rotate	random-conv
Game	Jumper	Chaser	Climber	Dodgeball	Heist	Leaper	Maze	Miner
Augmentation	random-conv	crop	color-jitter	crop	crop	crop	crop	color-jitter

(`https://github.com/rraileanu/idaac`). IDAAC was a variant of DAAC that uses an auxiliary loss which constrains the policy representation to be invariant to the task instance. As reported in (Raileanu & Fergus, 2021), the number of epochs used during each update of the policy network was 1, the number epochs used during each update of the value network was 9, the value update frequency was 1, the coefficient of instance-invariant (adversarial) loss was 0.001, the coefficient of the advantage loss was 0.25, and the remaining hyperparameters were the same as Table 4.

DAAC+RE3. (Seo et al., 2021) For RE3, we followed the implementation in the publicly released repository (`https://github.com/younggyoseo/RE3`). Here, the intrinsic reward is computed as $I_t = \log(\|\boldsymbol{e}_t - \tilde{\boldsymbol{e}}_t\|_2 + 1)$, where $\boldsymbol{e}_t = g(\boldsymbol{s}_t)$ and g is a random and fixed encoder. The total reward of time step t is computed as $E_t + \beta_t \cdot I_t$, where $\beta_t = \beta_0(1-\kappa)^t$. Moreover, the average distance of $\boldsymbol{e}_t$ and its k-nearest neighbors was used to replace the single k nearest neighbor to provide a less noisy state entropy estimate. As for hyperparameters related to exploration, we used $k = 5$, $\beta_0 = 0.05$, and performed hyperparameter search over $\kappa \in \{0.00001, 0.000025\}$. Finally, the policy was updated using DAAC with general hyperparameters listed in Table 4.

DAAC+RISE. (Yuan et al., 2022c) For RISE, we followed the implementation in the publicly released repository (`https://github.com/yuanmingqi/rise`). Here, the intrinsic reward is computed as $I_t = (\|\boldsymbol{e}_t - \tilde{\boldsymbol{e}}_t\|_2)^{1-\alpha}$, where $\boldsymbol{e}_t = g(\boldsymbol{s}_t)$ and g is a random and fixed encoder. The total reward of time step t is computed as $E_t + \beta_t \cdot I_t$, where $\beta_t = \beta_0(1-\kappa)^t$. As for hyperparameters related to exploration, we used $k = 5$, $\alpha = 0.05$, $\beta_0 = 0.1$, and performed hyperparameter search over $\kappa \in \{0.00001, 0.000025\}$. Finally, the policy was updated using DAAC with general hyperparameters listed in Table 4.

DAAC+RIDE. (Raileanu et al., 2020) For RIDE, we followed the implementation in the publicly released repository (`https://github.com/facebookresearch/impact-driven-exploration`). In practice, we trained a single forward dynamics model g to predict the encoded next-state $\psi(\boldsymbol{s}_{t+1})$ based on the current encoded state and action $(\psi(\boldsymbol{s}_t), \boldsymbol{a}_t)$, whose loss function was $\|g(\psi(\boldsymbol{s}_t), \boldsymbol{a}_t) - \psi(\boldsymbol{s}_{t+1})\|_2$. Then the intrinsic reward was computed as $I_t = \|\psi(\boldsymbol{s}_{t+1}) - \psi(\boldsymbol{s}_t)\|_2 / \sqrt{N_{\text{ep}}(\boldsymbol{s}_{t+1})}$, where N_{ep} is the state visitation frequency during the current episode. To estimate the state visitation frequency of $\boldsymbol{s}_{t+1}$, we leveraged a pseudo-count method that approximates the frequency using the distance between $\psi(\boldsymbol{s}_t)$ and its k-nearest neighbor within episode (Badia et al., 2020). Finally, the policy was updated using DAAC with general hyperparameters listed in Table 4.

Table 4. General hyperparameters used to obtain the Procgen results.

Hyperparameter	**Value**
Observation downsampling	(84, 84)
Stacked frames	No
Environment steps	25000000
Episode steps	265
Number of workers	1
Environments per worker	64
Optimizer	Adam
Learning rate	0.0005
GAE coefficient	0.95
Action entropy coefficient	0.05
Value loss coefficient	0.5
Value clip range	0.2
Max gradient norm	0.5
Epochs per rollout	3
Mini-batches per epoch	8
Reward normalization	Yes
LSTM	No
Gamma γ	0.99

Table 5. Procgen scores on train levels after training on 25M environment steps. (i) The mean and standard deviation are computed using 10 random seeds, and the highest average score is marked in color. (ii) The best data augmentation for each game is used when computing the results for DrAC. (iii) RE3 represents the combination of DAAC and RE3, RISE represents the combination of DAAC and RISE, and RIDE represents the combination of DAAC and RIDE. These three methods only use a fixed intrinsic reward function during the training, while AIRS automatically selects the most appropriate intrinsic reward in real-time. (iv) AIRS achieves the highest performance in 8 out of 16 games, especially in the *StarPilot* game.

Game	PPO	DrAC	DAAC	IDAAC	RE3	RISE	RIDE	AIRS (ours)
BigFish	10.2±2.0	10.2±1.4	20.0±2.9	19.7±1.1	19.4±1.9	17.0±1.3	19.1±2.2	20.2±1.2
BossFight	7.6±0.6	8.3±0.9	10.4±0.6	10.1±0.3	10.5±0.1	10.6±0.4	10.4±0.2	10.7±0.4
CaveFlyer	6.1±0.4	6.6±1.2	6.4±0.7	5.9±0.4	5.8±0.8	4.9±0.6	5.7±0.3	6.0±0.3
Chaser	4.7±0.5	5.5±1.1	5.1±0.3	7.2±0.8	5.2±0.7	4.1±1.1	5.6±0.5	5.4±0.4
Climber	8.2±0.6	8.8±0.8	9.4±0.7	9.3±0.4	9.2±0.5	9.6±0.4	9.3±0.2	9.8±0.4
CoinRun	8.8±0.2	9.4±0.2	9.9±0.2	9.9±0.2	10.0±0.3	10.0±0.7	10.0±0.5	10.0±0.4
Dodgeball	5.8±0.8	4.5±0.5	4.6±0.5	4.6±0.4	4.4±0.7	3.2±0.2	4.4±0.3	5.4±0.4
FruitBot	27.4±0.4	29.8±0.7	29.6±1.0	29.0±0.4	30.0±1.1	28.9±1.5	29.5±1.8	30.3±0.3
Heist	4.6±0.5	8.0±0.4	5.7±0.3	5.4±0.8	4.5±0.3	4.7±0.3	5.2±0.3	5.7±0.4
Jumper	8.6±0.3	8.6±0.3	8.5±0.5	8.5±0.3	8.3±0.4	7.6±0.3	8.4±0.7	8.9±0.7
Leaper	3.7±0.7	3.6±0.6	7.9±0.8	8.4±0.9	2.9±1.0	4.2±0.3	4.3±0.3	4.2±0.4
Maze	8.0±0.5	8.6±0.3	5.6±0.8	6.4±0.4	6.3±0.7	5.9±0.3	5.9±0.7	6.6±0.4
Miner	10.1±0.6	12.0±0.2	11.4±1.1	11.3±0.5	11.8±0.9	10.6±0.1	12.2±1.7	12.4±0.2
Ninja	7.8±0.3	7.7±0.8	9.0±0.2	9.1±0.2	9.0±0.7	9.3±0.7	9.6±0.9	9.5±0.5
Plunder	6.3±0.4	6.2±1.1	23.0±1.7	24.6±2.1	22.2±0.4	20.3±0.8	21.2±0.6	23.2±1.3
StarPilot	30.3±1.9	31.2±2.9	38.3±2.0	34.3±2.0	40.0±3.0	36.0±1.3	36.0±2.1	40.1±0.6

Table 6. Procgen scores on test levels after training on 25M environment steps. (i) The mean and standard deviation are computed using 10 random seeds, and the highest average score is marked in color. (ii) The best data augmentation for each game is used when computing the results for DrAC. (iii) RE3 represents the combination of DAAC and RE3, RISE represents the combination of DAAC and RISE, and RIDE represents the combination of DAAC and RIDE. These three methods only use a fixed intrinsic reward function during the training, while AIRS automatically selects the most appropriate intrinsic reward in real-time. (iv) AIRS achieves the highest performance in 9 out of 16 games, especially in the *BigFish* game and *StarPilot* game.

Game	PPO	DrAC	DAAC	IDAAC	RE3	RISE	RIDE	AIRS (ours)
BigFish	4.2±0.9	8.2±1.1	17.0±3.6	17.9±0.7	18.0±1.3	15.2±4.9	16.5±3.5	19.0±0.6
BossFight	7.0±0.6	7.9±0.6	9.7±0.4	9.7±0.4	8.8±0.2	9.2±0.4	9.8±0.8	9.9±0.4
CaveFlyer	5.4±0.8	5.3±0.5	4.7±0.4	5.1±0.3	4.7±0.6	3.8±0.5	4.7±0.7	4.8±0.4
Chaser	4.6±0.8	5.2±1.0	5.5±0.4	6.2±0.4	6.2±0.7	4.4±0.3	5.2±0.5	6.2±0.4
Climber	5.7±0.6	6.0±0.6	7.2±0.6	7.8±0.3	7.5±0.1	7.7±0.3	6.5±0.3	7.5±0.1
CoinRun	8.8±0.4	9.0±0.2	9.2±0.3	9.1±0.4	9.2±0.5	9.9±0.6	9.9±0.5	9.9±0.3
Dodgeball	2.3±0.4	2.9±0.2	3.3±0.2	3.2±0.6	2.7±0.1	2.7±0.3	2.7±0.1	3.4±1.0
FruitBot	23.4±0.7	26.7±0.9	27.6±0.8	28.4±0.8	27.9±1.5	28.3±1.2	28.6±0.6	30.0±0.4
Heist	2.8±0.7	3.6±0.5	3.5±0.5	3.0±0.3	3.4±0.1	3.3±0.6	3.4±0.3	3.7±0.2
Jumper	5.9±0.4	5.8±0.3	6.3±0.5	5.9±0.3	6.7±0.3	5.9±0.7	6.1±0.6	6.6±0.2
Leaper	2.9±0.4	3.7±0.6	7.0±0.5	7.5±1.1	3.6±0.3	4.0±0.3	3.6±0.7	4.2±0.7
Maze	5.6±0.7	5.1±0.5	5.5±1.1	5.6±0.6	5.7±1.0	6.2±0.3	5.4±0.3	5.9±0.5
Miner	7.6±0.7	9.1±0.4	8.5±0.8	9.6±0.1	8.8±1.3	9.1±0.6	8.7±1.0	9.0±0.5
Ninja	5.8±0.5	6.1±0.5	6.9±0.6	6.7±0.3	6.9±0.4	6.3±0.7	7.0±0.2	7.5±0.3
Plunder	5.9±0.6	9.4±0.4	20.6±1.5	23.5±1.1	21.2±0.1	19.0±0.2	22.1±0.3	22.5±0.9
StarPilot	24.7±1.3	27.0±3.4	36.8±1.7	35.9±3.4	36.7±2.0	35.6±3.0	37.1±2.4	37.5±2.7

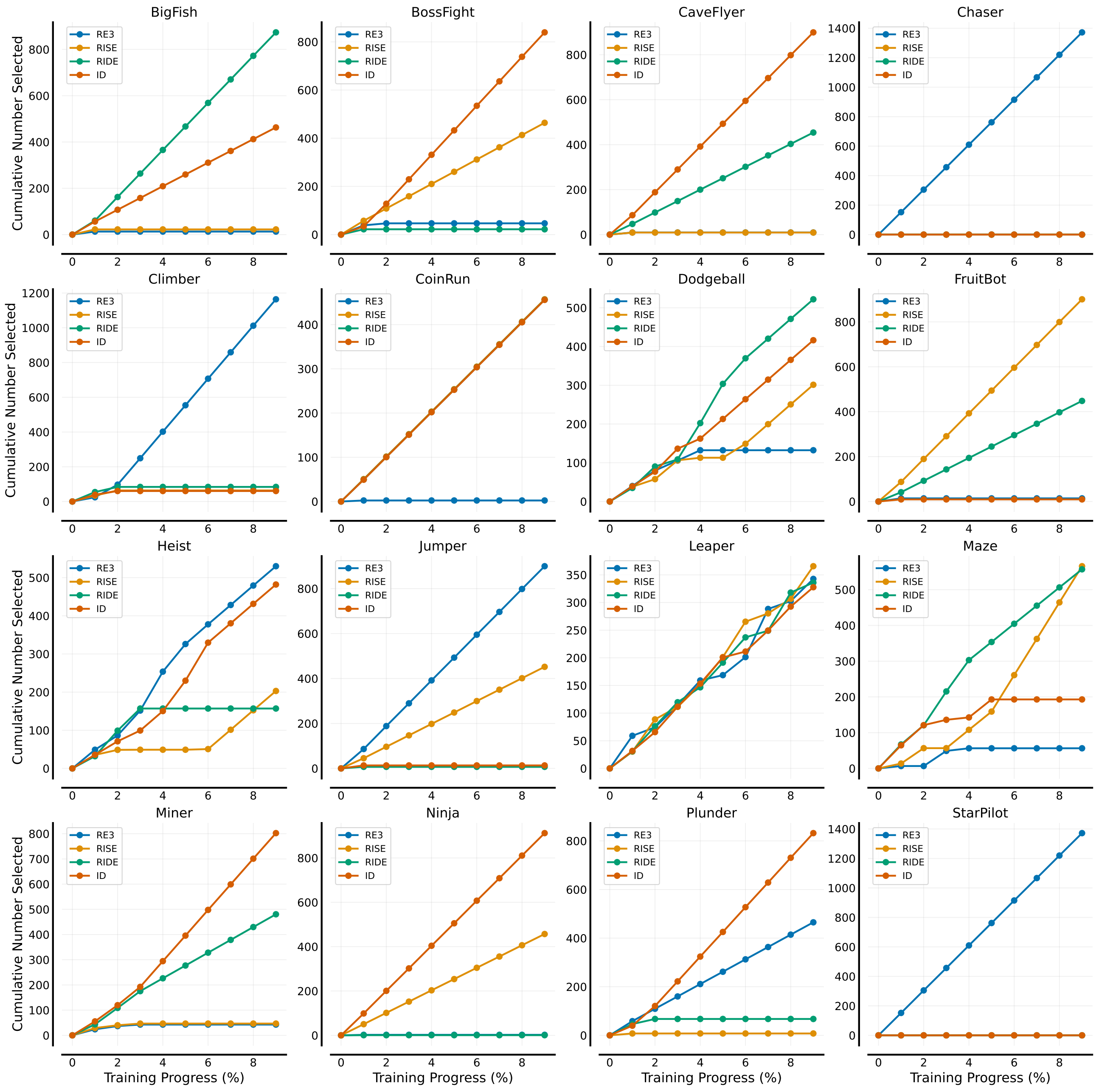

Figure 13. The cumulative number of times AIRS selects each intrinsic reward function over the training progress, computed with ten random seeds.

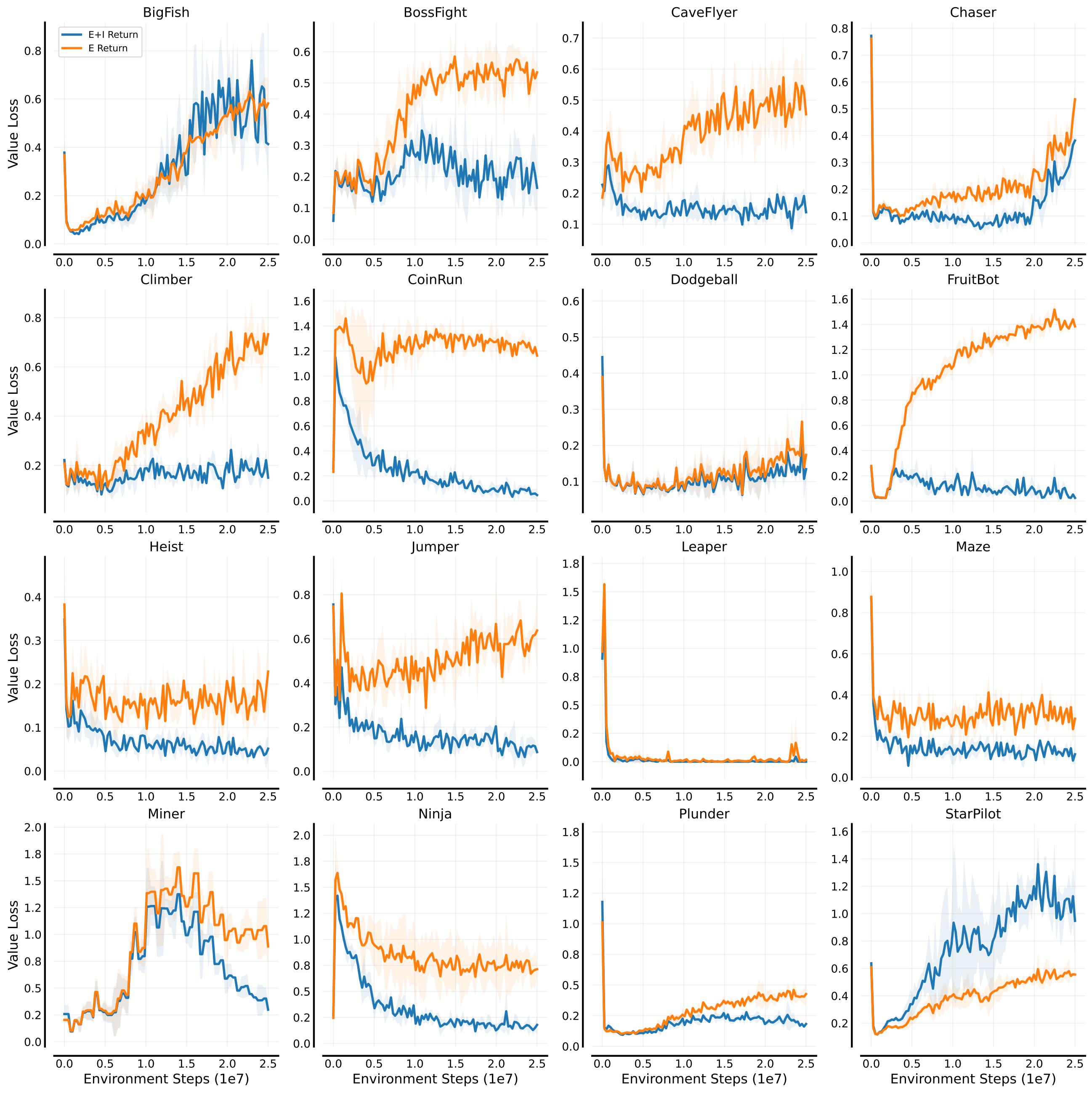

Figure 14. The loss curves of value estimation. Here, "E Return" represents the estimated return evaluated by the extrinsic reward function. The solid line and shaded regions represent the mean and standard deviation over ten random seeds, respectively.

C. Details on DeepMind Control Suite Experiments

C.1. Environment Setting

In this section, we evaluated the performance of AIRS on four tasks from DeepMind Control Suite (Tassa et al., 2018), namely *Cartpole Balance*, *Cheetah Run*, *Finger Spin*, and *Walker Walk*, respectively. To test the generalization ability of AIRS, we followed (Zhang et al., 2020) to replace the task background using synthetic distractors. Figure 15 illustrates the derived observations with synthetic distractor backgrounds and default backgrounds, and the code for background generation can be found in (`https://github.com/facebookresearch/deep_bisim4control`). For each task, we stacked three consecutive frames as one observation, and these frames were further cropped to the size of (84, 84) to reduce the computational resource request.

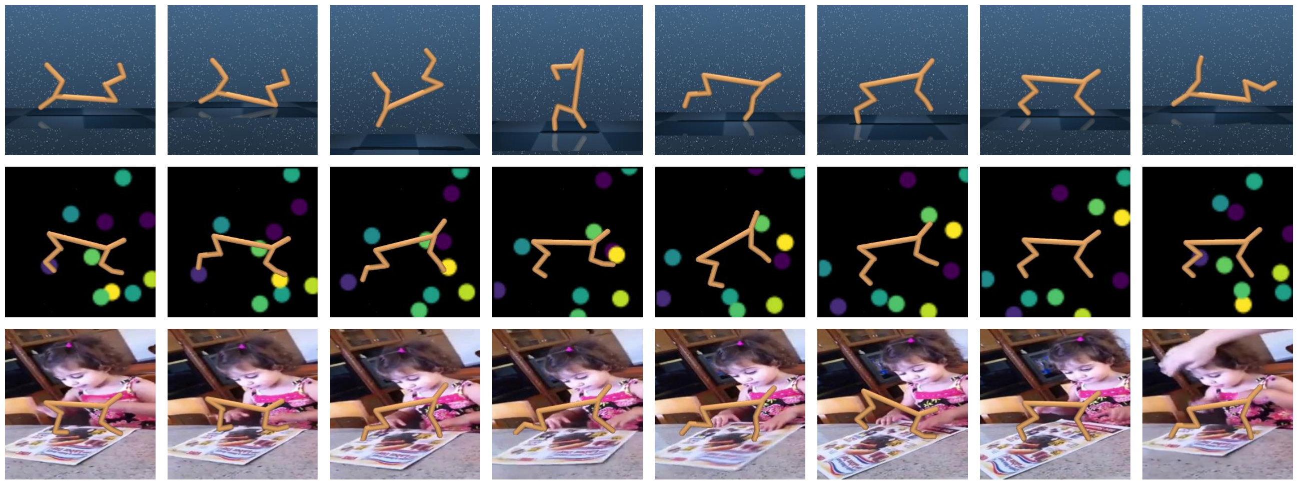

Figure 15. Environment examples of DeepMind Control Suite. Top row: default backgrounds without any distractors. Middle row: synthetic distractor backgrounds with ideal gas videos. Bottom row: natural distractor backgrounds with Kinetics videos.

C.2. Experiment Setting

We compared AIRS with PPO (Schulman et al., 2017) and IDAAC (Raileanu & Fergus, 2021). For each task, the agents were trained for 1M frames and evaluated every 10K frames. Table 7 illustrates the hyperparemeters derived by grid search. Any other hyperaparameters not mentioned here were set to the same values as the ones used for Procgen games in Table 4.

Table 7. Hyperparameters used to obtain the DeepMind Control Suite results.

Hyperparameter	**Value**
Observation downsampling	(84, 84)
Stacked frames	3
Environment steps	1000000
Episode steps	2048
Number of workers	1
Environments per worker	1
Learning rate	0.001 *Cartpole Balance*, *Cheetah Run*, and *Walker Walk*; 0.0001 *Finger Spin*
GAE coefficient	0.99
Action entropy coefficient	0.0001 *Cheetah Run*; 0.0 *Cartpole Balance* and *Finger Spin*; 0.001 *Walker Walk*;
Epochs per rollout	0.0001 *Cheetah Run*; 10 *Cartpole Balance* and *Finger Spin*; 5 *Walker Walk*
Mini-batches per epoch	64 *Cheetah Run*; 16 *Cartpole Balance* and *Finger Spin*; 32 *Walker Walk*

D. Intrinsic Reward Toolkit

D.1. Introduction

Intrinsic rewards have been widely used to improve the exploration, and generalization ability of RL agents (Ryan & Deci, 2000; Şimşek & Barto, 2006; Stadie et al., 2015; Bellemare et al., 2016; Ostrovski et al., 2017; Burda et al., 2019b; Savinov et al., 2019; Raileanu et al., 2020; Yu et al., 2020; Yuan et al., 2022c;b). However, there are great differences in the implementation of various intrinsic reward modules, which cannot provide efficient and reliable baselines. To facilitate our experiments and inspire subsequent research on intrinsic rewards, we developed a toolkit that provides high-quality implementations of various intrinsic reward modules based on PyTorch. This toolkit is designed to be highly modular and scalable. Each intrinsic reward module can be deployed in arbitrary algorithms in a plug-and-play manner. Table 8 illustrates currently included implementations in the toolkit, and the code is available at `https://github.com/RLE-Foundation/rllte`.

Table 8. Included implementations of the intrinsic reward toolkit. (i) Here, $\boldsymbol{e} = \psi(\boldsymbol{s})$ is the learned representation of state $\boldsymbol{s}$, and $N_{\rm ep}$ is the episodic state visitation frequency. For ICM and RIDE, $\psi(\cdot)$ is learned by reconstructing the transition process. For RE3, RISE and REVD, $\psi(\cdot)$ is a random and fixed encoder. (iv) GIRM is a variant of ICM that utilizes variational auto-encoder (Kingma & Welling, 2014) to reconstruct the transition process and computes the intrinsic reward in a end-to-end manner. (iii) For RND, $\hat{f}$ is the target network that is fixed and randomly-initialized neural network, and f is the predictor network that aims to approximate $\hat{f}$. (iv) The intrinsic reward produced by NGU is composed of episodic state novelty and life-long state novelty. Here, α_t is life-long curiosity factor computed following the RND method and C is is a chosen maximum reward scaling. (v) For RE3 and RISE, $\tilde{\boldsymbol{e}}$ is the k-nearest neighbor of $\boldsymbol{s}$ in the encoding space. For REVD, $\tilde{\boldsymbol{e}}$ is the k-nearest neighbor of $\boldsymbol{s}$ within the current episode, and $\breve{\boldsymbol{e}}$ is the k-nearest neighbor of $\boldsymbol{s}$ within the former episode. (vi) PseudoCounts is an ablation of NGU in which the life-long module is deprecated.

Intrinsic reward module	**Formulation**	**Remark**
PseudoCounts (Badia et al., 2020)	$I_t = 1/\sqrt{N_{\rm ep}(\boldsymbol{s}_{t+1})}$	Count-based exploration
ICM (Pathak et al., 2017)	$I_t = \Vert f(\boldsymbol{e}_t, \boldsymbol{a}_t) - \boldsymbol{e}_{t+1}\Vert_2^2$	Curiosity-driven exploration
RND (Burda et al., 2019b)	$I_t = \Vert \hat{f}(\boldsymbol{s}_{t+1}) - f(\boldsymbol{s}_{t+1})\Vert_2^2$	Count-based exploration
GIRM (Yu et al., 2020)	$I_t = \Vert \hat{\boldsymbol{s}}_t - \boldsymbol{s}_{t+1}\Vert_2^2$	Curiosity-driven exploration
NGU (Badia et al., 2020)	$I_t = \min\{\max\{\alpha_t\}, C\}/\sqrt{N_{\rm ep}(\boldsymbol{s}_{t+1})}$	Memory-based exploration
RIDE (Raileanu et al., 2020)	$I_t = \Vert \boldsymbol{e}_{t+1} - \boldsymbol{e}_t\Vert_2/\sqrt{N_{\rm ep}(\boldsymbol{s}_{t+1})}$	Significant state changes
RE3 (Seo et al., 2021)	$I_t = \frac{1}{k}\sum_{i=1}^{k}\log(\Vert \boldsymbol{e}_t - \tilde{\boldsymbol{e}}_t^i\Vert_2 + 1)$	Shannon entropy maximization
RISE (Yuan et al., 2022c)	$I_t = \frac{1}{k}\sum_{i=1}^{k}(\Vert \boldsymbol{e}_t - \tilde{\boldsymbol{e}}_t^i\Vert_2)^{1-\alpha}$	Rényi entropy maximization
REVD (Yuan et al., 2022c)	$I_t = \frac{1}{k}\sum_{i=1}^{k}(\Vert \boldsymbol{e}_t - \breve{\boldsymbol{e}}_t^i\Vert_2/\Vert \boldsymbol{e}_t - \tilde{\boldsymbol{e}}_t^i\Vert_2)^{1-\alpha}$	Rényi divergence maximization

D.2. Usage Example

Due to the large differences in the calculation of different intrinsic reward methods, the toolkit has the following rules:

- The environments are assumed to be gym-like and vectorized;
- Each intrinsic reward module has a compute_irs function with a mandatory argument rollouts. It is a Python dict like (Harris et al., 2020):

Key	**Data shape**	**Data type**
observations	(n_steps, n_envs, *obs_shape)	PyTorch Tensor
actions	(n_steps, n_envs, *action_shape)	PyTorch Tensor
rewards	(n_steps, n_envs)	PyTorch Tensor
next observations	(n_steps, n_envs, *obs_shape)	PyTorch Tensor

Take RE3 for instance, it computes the intrinsic reward for each state s based on the Euclidean distance between the state s and its k-nearest neighbor within a mini-batch. Thus it suffices to provide observations data to compute the reward. The following code provides a usage example of RE3:

```python
"""Load package"""
from rllte.xplore.reward import RE3
from rllte.env import make_atari_env
import torch as th

if __name__ == '__main__':
    """ Environment setup """
    num_envs = 8
    num_steps = 128
    device = "cpu"
    env = make_atari_env(
        env_id="PongNoFrameskip-v4",
        num_envs=num_envs,
        device=device
    )
    print(env.observation_space, env.action_space)

    """ Create RE3 instance """
    re3 = RE3(
        observation_space=env.observation_space,
        action_space=env.action_space,
        device=device
    )
    """ Compute intrinsic rewards """
    obs = th.rand(size=(num_steps, num_envs, *env.observation_space.shape))
    intrinsic_rewards = re3.compute_irs(samples={'obs': obs})

    print(intrinsic_rewards.shape, type(intrinsic_rewards))
    print(intrinsic_rewards)

# Output:
# Box(0, 255, (4, 84, 84), uint8) Discrete(6)
# torch.Size([128, 8]) <class 'torch.Tensor'>
```

Figure 16. Code example of the intrinsic reward toolkit.

We are testing this toolkit on various tasks (e.g., OpenAI Gym and DeepMind Control Suite) and will establish a complete test dataset to provide convenient baselines. More consequent results can be found at `https://hub.rllte.dev/`. We will also continue to follow up the latest research on intrinsic reward-driven exploration and provide reliable implementations.