RLeXplore: Accelerating Research in Intrinsically-Motivated Reinforcement Learning

[1*]Mingqi Yuan, [2*]Roger Creus Castanyer, [1]Bo Li, [3]Xin Jin, [2]Glen Berseth, [3]Wenjun Zeng
[1]Department of Computing, The Hong Kong Polytechnic University, China
[2]Mila Québec AI Institute & Université de Montréal, Canada
[3]Eastern Institute of Technology, Ningbo, China

Abstract

Extrinsic rewards can effectively guide reinforcement learning (RL) agents in specific tasks. However, extrinsic rewards frequently fall short in complex environments due to the significant human effort needed for their design and annotation. This limitation underscores the necessity for intrinsic rewards, which offer auxiliary and dense signals and can enable agents to learn in an unsupervised manner. Although various intrinsic reward formulations have been proposed, their implementation and optimization details are insufficiently explored and lack standardization, thereby hindering research progress. To address this gap, we introduce RLeXplore, a unified, highly modularized, and plug-and-play framework offering reliable implementations of eight state-of-the-art intrinsic reward algorithms. Furthermore, we conduct an in-depth study that identifies critical implementation details and establishes well-justified standard practices in intrinsically-motivated RL. The source code for RLeXplore is available at `https://github.com/RLE-Foundation/RLeXplore`.

1 Introduction

Reinforcement learning (RL) provides a framework for training agents to solve tasks by learning from interactions with an environment. At the core of RL is the optimization of a reward function, where agents aim to maximize cumulative rewards over time (Sutton & Barto, 2018). However, in complex environments, defining extrinsic rewards that effectively guide an agent's learning process can be impractical, often requiring domain-specific expertise. In practice, poorly defined extrinsic rewards can lead to sparse-reward settings, where RL agents struggle due to the lack of a meaningful learning signal (Burda et al., 2019a).

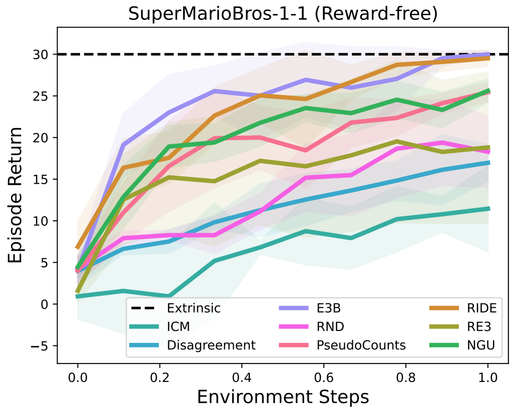

Figure 1: **Episode Return** achieved by the intrinsic rewards in **RLeXplore** in *SuperMarioBros*.

As the RL community tackles increasingly complex problems, such as training generally capable RL agents, there is a need for more autonomous agents capable of learning valuable behaviors without relying on dense supervision (Jiang et al., 2023). To address this challenge, the concept of intrinsic rewards has emerged as a promising approach in the RL community (Burda et al., 2019b; Pathak et al., 2017; Raileanu et al., 2020; Badia et al., 2020; Henaff et al., 2022; Pathak et al., 2019). Intrinsic rewards provide agents with additional learning signals, enabling them to explore and acquire skills across diverse environments beyond what extrinsic rewards alone can offer.

*Equal Contribution.

However, computing intrinsic rewards often requires learning auxiliary models, heavy engineering and performing expensive computations, making reproducibility challenging. While several formulations of intrinsic rewards have been proposed Pathak et al. (2017); Badia et al. (2020); Laskin et al. (2021), each with its potential benefits for improving agent learning, the field lacks a comprehensive understanding of the comparative advantages and challenges posed by these formulations. Importantly, existing literature reports varying performance when using the same intrinsic rewards, reinforcing the need for a standardized framework and a deeper understanding of the optimization and implementation details.

In this paper, we introduce **RLeXplore**, an open-source library containing high-quality implementations of state-of-the-art intrinsic rewards. Our work presents a systematic study aimed at addressing gaps in understanding the critical implementation and optimization details of intrinsic rewards. To guide our investigation, we formulate numerous questions, aiming to uncover the intricacies of intrinsic rewards and their impact on RL agent performance. Our results highlight the importance of thoughtful implementation design for intrinsic rewards, showing that naive implementations can lead to suboptimal performance. Through carefully studied design decisions, we demonstrate significant performance gains.

Our contributions are threefold. Firstly, we provide a high-quality open-source repository for training RL agents, featuring the implementation of the most widely recognized intrinsic rewards. Secondly, we present a systematic evaluation that identifies the key implementation and optimization details critical to the success of intrinsic reward methods in RL. Lastly, we provide a comparative analysis of the agents' performance across challenging environments, establishing a foundation for future research in intrinsically-motivated RL.

2 Background

We frame the RL problem considering a MDP Bellman (1957); Kaelbling et al. (1998) defined by a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, E, P, d_0, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, and $E : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the extrinsic reward function, $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition function that defines a probability distribution over $\mathcal{S}$, $d_0 \in \Delta(\mathcal{S})$ is the distribution of the initial observation $\boldsymbol{s}_0$, and $\gamma \in [0, 1)$ is a discount factor. The goal of RL is to learn a policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})$ to maximize the expected discounted return:

$$J_\pi(\boldsymbol{\theta}) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t E_t \right]. \tag{1}$$

Intrinsic rewards augment the learning objective to improve exploration. Letting $I : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denote the intrinsic reward function, the augmented optimization objective is:

$$J_\pi(\boldsymbol{\theta}) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t (E_t + \beta_t \cdot I_t) \right], \tag{2}$$

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

3 RLeXplore

In this section, we present **RLeXplore**, a unified, highly-modularized and plug-and-play framework that currently provides high-quality and reliable implementations of eight state-of-the-art intrinsic reward algorithms[*]. Comparing multiple intrinsic reward methods under fair conditions is challenging due to various confounding factors, such as using distinct backbone RL algorithms (e.g. PPO Schulman et al. (2017), DQN Mnih et al. (2013), IMPALA Espeholt et al. (2018)), optimization (e.g. reward and observation normalization, network architecture) and evaluation details (e.g. environment configuration, algorithm hyperparameters). RLeXplore is designed to provide a

[*] RLeXplore complies with the MIT License.

unified framework with standardized procedures for implementing, computing, and optimizing intrinsic rewards. We provide the details of the architecture and algorithm baselines of RLeXplore in Appendix B.

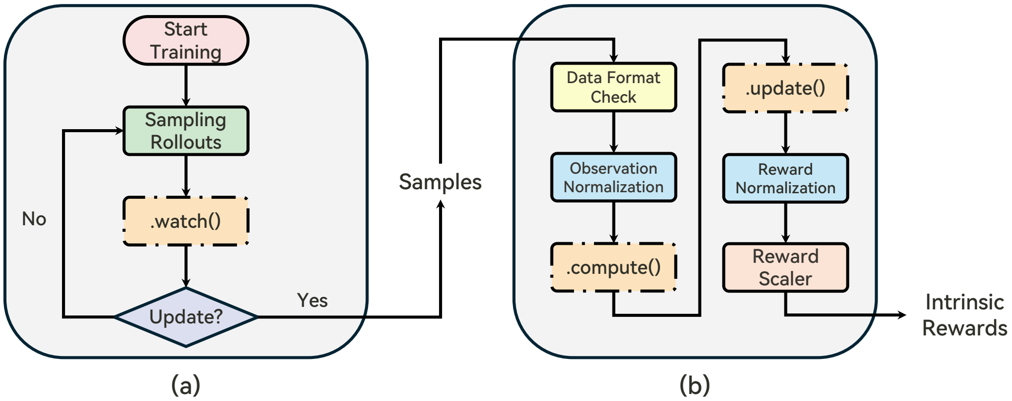

Figure 2: The workflow of RLeXplore. (a) RLeXplore monitors the agent-environment interactions and obtains data samples via a **.watch()** function. (b) For the sampled transitions, RLeXplore calculates the corresponding intrinsic rewards via a **.compute()** function and performs model updating via a **.update()** function.

4 Experiments

We design our experiments with two objectives in mind: (i) evaluating the effectiveness of our implementations and their adaptability to different reinforcement learning (RL) algorithms, and (ii) investigating the tuning of intrinsic rewards to provide well-justified standard implementations.

For the first part, we use the *Montezuma's Revenge* (MR) and *Ant-UMaze* environments, shown in Figure 3. MR is a challenging exploration task from the Atari benchmark Bellemare et al. (2013), while Ant-UMaze is an exploration and locomotion task from the Gymnasium robotics benchmark de Lazcano et al. (2023). We provide the experimental settings and results in Appendix F and Appendix G. In these experiments, we seamlessly integrate RLeXplore with different frameworks (CleanRL Huang et al. (2022), RLLTE Yuan et al. (2023)) and RL algorithms (PPO Schulman et al. (2017), SAC Haarnoja et al. (2018)).

For the second part, we use *SuperMarioBros* to investigate the low-level implementation details in intrinsic reward methods that lead to more robust performance. Additionally, we use *Procgen-Maze* to study how to best combine intrnisic and extrinsic rewards in sparse-reward tasks.

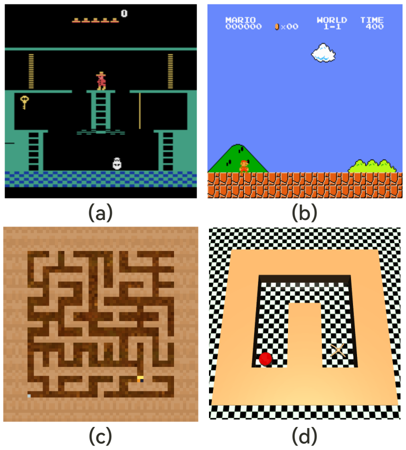

Figure 3: Screenshots of four selected exploration games. (a) *Montezuma's Revenge (MR)*. (b) *SuperMarioBros*. (c) *Procgen-Maze*. (d) *Ant-UMaze*.

4.1 Low-level Implementation Details of Intrinsic Rewards

The performance of intrinsic rewards is affected by various factors, which tends to vary significantly with the complexity of the task, the RL algorithm used, the architecture of the networks, algorithm-specific hyperparameters, and the joint optimization of intrinsic and extrinsic rewards. As a result, implementing and reproducing intrinsic reward algorithms is challenging. To tackle this problem, we first formulate five research questions (RQs) to investigate how various low-level implementation

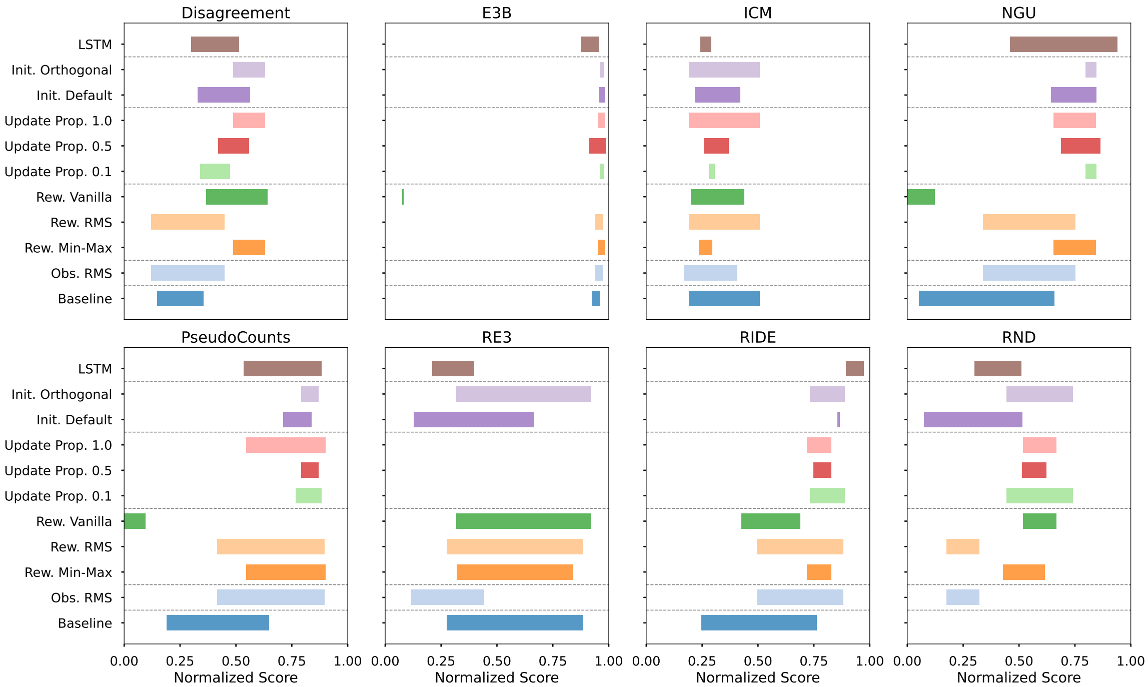

Figure 4: Results for RQ 1, RQ 2, RQ 3, RQ 4, and RQ 5 in *SuperMarioBros-1-1*. The x-axis represents the episode return normalized by the performance of the extrinsic agent after training. A normalized return of 1.0 indicates that the agent solves the 1-1 level. Each bar represents the mean and standard deviation of the normalized episode returns. Results are aggregated over 5 seeds, and each run uses 10M environment interactions.

details impact the training of intrinsically-motivated agents. We first define an initial baseline configuration for optimizing the intrinsic rewards, shown in Table 1. These baseline settings are selected based on the most common configurations reported in the literature. Next, we address each RQ sequentially, modifying the baseline configuration for each intrinsic reward as we gather new evidence regarding their critical implementation details. This iterative process leads to the development of high-quality implementations of state-of-the-art intrinsic reward methods.

In this section, we conduct reward-free experiments (i.e., without access to extrinsic rewards) using the *SuperMarioBros* environment Kauten (2018). *SuperMarioBros* is a widely used benchmark for evaluating exploration in RL Pathak et al. (2017); Raileanu et al. (2020), as efficient exploration is closely related to effectively navigating the game levels and ultimately solving the game. Additional experimental settings can be found in Appendix C.

RQ 1: The impact of observation normalization.

Observation normalization is crucial in deep learning to avoid numerical instabilities during optimization. Image observations, which typically range from 0 to 255, are commonly normalized to a range of 0 to 1 using Min-Max normalization by dividing by 255. However, previous studies suggest that Min-Max normalization may not be ideal for all representation learning algorithms (Burda et al., 2019b).

In RQ 1, we compare Min-Max normalization with using an exponential moving average (EMA) of the mean and standard deviation for observation normalization (RMS). Figure 4 indicates that using RMS for observation normalization generally reduces the variance of the agent's performance. Importantly, some intrinsic rewards, such as RND, NGU, PseudoCounts, and RIDE, critically require RMS normalization. This results indicate that RMS normalization is crucial for intrinsic reward methods that use random networks, since the lack of normalization can result in the embeddings produced by the random networks carrying very little information about the inputs (Burda et al., 2019b).

RQ 2: The impact of reward normalization.

Similarly to RQ 1, reward normalization is also critical when using deep neural networks to compute the intrinsic rewards, since the scale of these rewards can be arbitrary and vary significantly over time. To mitigate the non-stationarity of intrinsic rewards, in RQ 2, we compare three normalization approaches: (1) Min-Max normalization, (2) using an RMS of the standard deviation, and (3) no reward normalization.

Surprisingly, our findings reveal that not normalizing intrinsic rewards at all can improve the performance of some intrinsic reward methods (e.g., RE3 and ICM). However, our results show that Min-Max reward normalization is a considerable option, improving the performance of the majority of the methods.

RQ 3: The co-learning dynamics of policies and representations for intrinsic rewards.

Optimizing intrinsic rewards in deep RL often involves training representations in auxiliary networks (e.g., predictor network in RND, inverse dynamics encoder in ICM, forward dynamics encoders in Disagreement). However, managing the co-learning dynamics of representations and policies is challenging. In RQ 3, we explore three update strategies for representations: (1) updating representations at the same frequency as the policy, (2) updating representations 50% of the time, and (3) updating representations 10% of the time. This comparison sheds light on the trade-off between the number of gradient updates in the representations and the performance of the policy.

Our findings suggest that, in general, the learned representations of the auxiliary networks need to be caught up (i.e., equally trained) with the same data as the RL agents. Our results show that training the representations slowly (e.g., using only 10% of the data at each iteration) can cause significant performance decreases for global intrinsic rewards like ICM, Disagreement and RND. In contrast, for intrinsic rewards that utilizes episodic memories (e.g., NGU and RIDE), training the representations slowly may improve the performance to a certain extent. These observations indicate that training intrinsic rewards and RL agents does not always have to be synchronized entirely. Moreover, we can reduce the computational overhead by reducing the training time of intrinsic rewards.

RQ 4: The impact of weight initialization.

Weight initialization plays a crucial role in optimizing deep neural networks, enabling faster convergence. In RQ 4, we compare two approaches for weight initialization in the auxiliary networks: (1) orthogonal weight initialization and (2) uniform weight initialization (PyTorch's default). Note that the policy and value networks remain unchanged.

Our results show the importance of weight initialization in intrinsically-motivated RL. Specifically, we observe that using orthogonal initialization of weights leads to notable performance improvements for most intrinsic rewards, regardless of their specific optimization tasks (e.g., inverse dynamics, forward dynamics), and even in random networks (e.g., RND and RE3).

RQ 5: Is memory required to optimize intrinsic rewards?

In RQ 5, we investigate whether the intrinsic rewards included in RLeXplore benefit from memory-enabled architectures. We compare the optimization of intrinsic rewards using a vanilla network and one equipped with a long-short term memory (LSTM) Hochreiter & Schmidhuber (1997) module, while keeping PPO as the RL backbone algorithm.

Some intrinsic reward methods exhibit lower performance when using LSTM policies. This observation aligns with the fact that LSTMs provide episodic context to policies, whereas most intrinsic reward methods define exploration as a global problem. Interestingly, for RIDE, which computes the state embedding changes as the intrinsic rewards, the episodic context provided by LSTMs enables agents to better optimize the intrinsic reward.

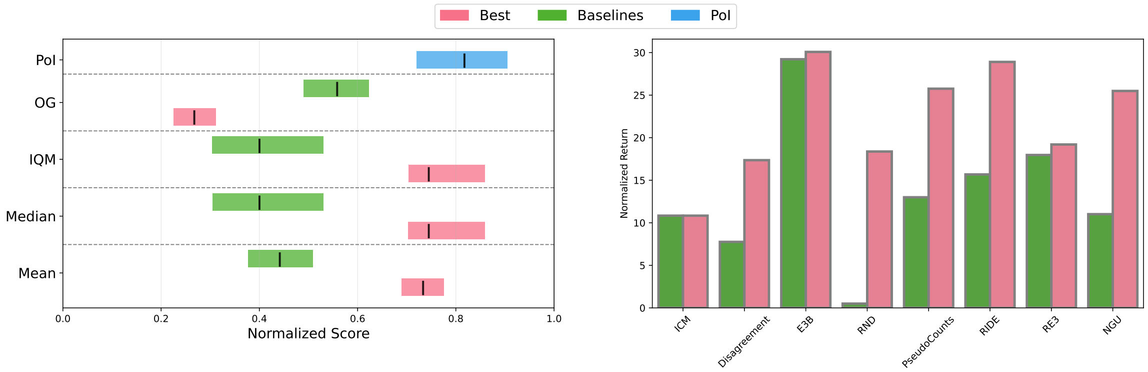

Figure 5: Overall (left) and individual (right) performance comparisons between the baselines and best configurations. For the left figure, **interquartile mean (IQM)**, **mean**, **median**, and **optimality gap (OG, lower is better)** are computed over all the algorithms and runs. **PoI** is the probability of improvement of the best configurations compared to the baselines. Our results show that carefully-tuned implementations for the intrinsic rewards achieve significantly better exploration and task performance.

Based on the results from RQ 1-5, we have identified the optimal configurations that surpass the predefined baselines. Reward and observation normalization are notably influential on performance, and by adjusting the update proportion and using an LSTM, it is possible to further boost the performance of intrinsic rewards. Figure 5 demonstrates both the comprehensive and individual performance comparisons before and after tuning. The average probability of improvement across all intrinsic rewards is 82%, with performance increases of 86% and 66% for the aggregate IQM and mean, respectively.

4.2 Combination of Intrinsic and Extrinsic Rewards

RQ 6: Joint Optimization of Intrinsic and Extrinsic Rewards

Training agents to maximize two learning signals concurrently can be challenging. In sparse-reward environments, the objective is for agents to explore the state space by optimizing intrinsic rewards until they discover the task rewards, at which point they should focus solely on optimizing the task rewards. However, many intrinsically-motivated RL applications naively optimize the sum of intrinsic and extrinsic rewards, potentially leading to learning fuzzy value functions and suboptimal policies. In this section, we compare this approach with learning two separate value functions, one for each stream of rewards. The advantages of the latter include the ability to disentangle the effects of intrinsic and extrinsic rewards on the agent's behavior, leading to clearer learning dynamics and potentially more efficient exploration.

For this analysis, we used the *Procgen-Maze* task Cobbe et al. (2020) as a sparse-reward benchmark. RL agents often struggle to learn meaningful behaviors from the extrinsic reward alone in this task. We evaluate different variants of the task (e.g., 1 maze vs. 200 mazes) to examine singleton versus contextual MDPs. Figure 6 demonstrates that learning two separate value functions, which we refer as the *TwoHead* architecture, outperforms the naive approach of simply adding the two rewards in the complex sparse-reward environment of *Procgen-Maze*, both in singleton and contextual settings. Importantly, all methods outperform the extrinsic agent, especially in the *1 Maze* environment.

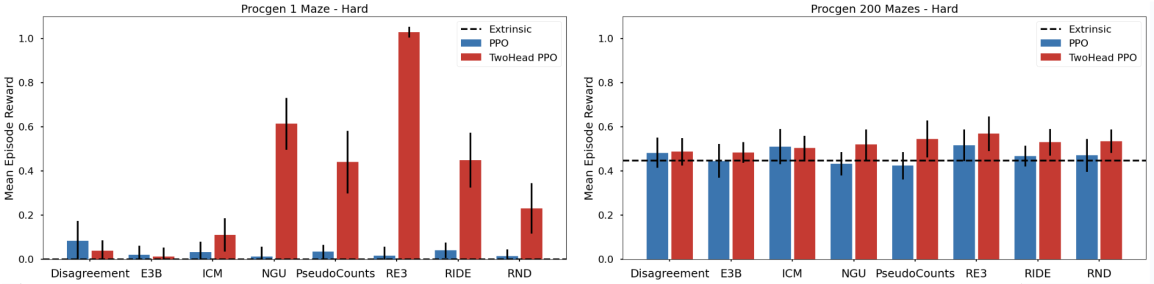

Figure 6: (Left) During training, the extrinsic agent struggles to find the goal in the selected Maze, resulting in a reward of 0. While some intrinsic reward methods yield occasional non-zero rewards, the algorithms perform significantly better when intrinsic and extrinsic value estimation are decoupled using two distinct value heads in the agent's network. (Right) In the Procgen variant where each maze represents a unique level, the baseline extrinsic agent achieves the goal 50% of the time, and intrinsic rewards don't outperform the baseline significantly. We note that the presence of easier levels, where the goal may occasionally be near the agent's starting point results in generally less sparse rewards and an easier task to learn.

4.3 Unlocking the Potential of Intrinsic Rewards

RQ 1-6 extensively discuss the tuning of intrinsic rewards under both normal and reward-free scenarios, revealing significant insights into the optimization processes. However, we aim to delve deeper into the capabilities of intrinsic rewards to address the evolving challenges in the RL community. Specifically, in RQ 7 and 8, we investigate recent developments in the exploration literature in RL, such as combined intrinsic rewards and exploration in contextual MDPs. For our experiments, we use the *SuperMarioBrosRandomStages* environment variant, where agents play a different level in the game at each episode. Our results indicate that the recent developments in combined intrinsic rewards merit further research, as we demonstrate that such methods can enable agents to learn exploratory behaviors of exceptional quality in both singleton and contextual MDPs.

RQ 7: Which intrinsic rewards generalize better in contextual MDPs?

In contextual MDPs, there is little shared structure across episodes, since the episodic context can vary the environment significantly. In this settings, global intrinsic rewards, which re-use experience from past episodes to compute novelty in the current episode can provide wrong estimations. Conversely, episodic intrinsic rewards, such as E3B and PseudoCounts, are specifically designed to estimate novelty within each new episode, aligning better with the dynamic nature of contextual MDPs. As shown in Figure 15, E3B achieves the highest performance among all the intrinsic rewards, while other intrinsic rewards struggle to adapt and nearly fail to learn. This distinct advantage underscores the importance of designing intrinsic rewards that are context-sensitive and capable of updating their novelty detection mechanisms based on the specific characteristics of each episode.

RQ 8: The performance of mixed intrinsic rewards.

Finally, we study the potential of combined intrinsic rewards (Henaff et al., 2023). We run experiments using all the levels in the game of *SuperMarioBros*, and we sample them uniformly during training. As in RQ 1-5, we do not use the extrinsic reward for training the agents but use it as an evaluation metric to show how much agents actively explore the environment.

Our results show that combined objectives enable emergent behaviors of much better quality than single objectives. Interestingly, E3B and RIDE are the best performing single objectives, and

E3B+RIDE also achieves the highest performance among all the combinations. Similarly, RND and ICM, combined with other intrinsic rewards, outperform their original performance. This indicates that different intrinsic rewards can provide orthogonal gains that can be leveraged together.

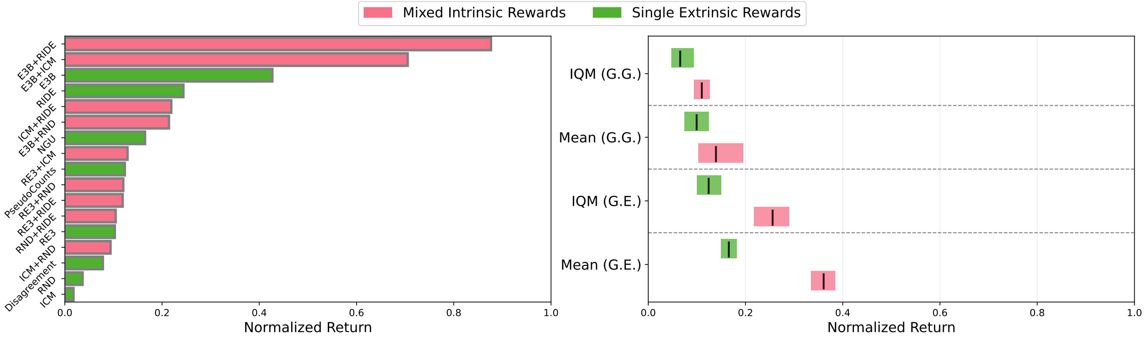

Figure 7: (Left) The performance ranking of single and mixed intrinsic rewards on the *SuperMarioBros-RandomLevels*. As expected, episodic bonuses (such as E3B and RIDE) demonstrate superior performance, attributed to the environment's non-singleton MDP nature. (Right) Overall performance comparisons between the single and mixed intrinsic rewards. Here, **G.E.** denotes the six "global+episodic" combinations, and **G.G.** denotes the three "global+global" combinations, as illustrated in Table 3.

5 Conclusion

Our work introduces RLeXplore, a comprehensive open-source repository that not only implements state-of-the-art intrinsic rewards but also provides a systematic evaluation framework for understanding their impact on agent performance. Our results show that with RLeXplore, RL agents can learn emergent behaviours autonomously, solving multiple levels of *SuperMarioBros* without task rewards. Additionally, we show that intrinsic rewards enable RL agents to solve complex sparse-reward tasks like *Procgen-Maze*. Finally, RLeXplore facilitates further research in combined intrinsic rewards, uncovering the potential of such methods.

Through our study, we emphasize the importance of thoughtful implementation design, demonstrating that well-considered approaches lead to significant performance gains over naive implementations. Our contributions extend to establishing standardized practices for implementing and optimizing intrinsic rewards, laying the groundwork for future advancements in intrinsically motivated RL.

References

Arthur Aubret, Laetitia Matignon, and Salima Hassas. An information-theoretic perspective on intrinsic motivation in reinforcement learning: A survey. *Entropy*, 25(2):327, 2023.

Adrià Puigdomènech Badia, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, Bilal Piot, Steven Kapturowski, Olivier Tieleman, Martin Arjovsky, Alexander Pritzel, Andrew Bolt, and Charles Blundell. Never give up: Learning directed exploration strategies. In *Proceedings of the International Conference on Learning Representations*, 2020.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Proceedings of Advances in Neural Information Processing Systems*, 29:1471–1479, 2016.

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.

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

Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. *Proceedings of the International Conference on Learning Representations*, pp. 1–17, 2019a.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *Proceedings of the 7th International Conference on Learning Representations*, pp. 1–17, 2019b.

Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pp. 2048–2056. PMLR, 2020.

Rodrigo de Lazcano, Kallinteris Andreas, Jun Jet Tai, Seungjae Ryan Lee, and Jordan Terry. Gymnasium robotics, 2023. URL http://github.com/Farama-Foundation/Gymnasium-Robotics.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pp. 1407–1416. PMLR, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

Mikael Henaff, Roberta Raileanu, Minqi Jiang, and Tim Rocktäschel. Exploration via elliptical episodic bonuses. *Advances in Neural Information Processing Systems*, 35:37631–37646, 2022.

Mikael Henaff, Minqi Jiang, and Roberta Raileanu. A study of global and episodic bonuses for exploration in contextual mdps. *arXiv preprint arXiv:2306.03236*, 2023.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022. URL http://jmlr.org/papers/v23/21-1342.html.

Minqi Jiang, Tim Rocktäschel, and Edward Grefenstette. General intelligence requires rethinking exploration. *Royal Society Open Science*, 10(6):230539, 2023.

Daejin Jo, Sungwoong Kim, Daniel Nam, Taehwan Kwon, Seungeun Rho, Jongmin Kim, and Donghoon Lee. Leco: Learnable episodic count for task-specific intrinsic reward. *Advances in Neural Information Processing Systems*, 35:30432–30445, 2022.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

Christian Kauten. Super Mario Bros for OpenAI Gym. GitHub, 2018. URL https://github.com/Kautenja/gym-super-mario-bros.

Michael Laskin, Denis Yarats, Hao Liu, Kimin Lee, Albert Zhan, Kevin Lu, Catherine Cang, Lerrel Pinto, and Pieter Abbeel. Urlb: Unsupervised reinforcement learning benchmark. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

Sam Lobel, Akhil Bagaria, and George Konidaris. Flipping coins to estimate pseudocounts for exploration in reinforcement learning. *arXiv preprint arXiv:2306.03186*, 2023.

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

Jarryd Martin, Suraj Narayanan Sasikumar, Tom Everitt, and Marcus Hutter. Count-based exploration in feature space for reinforcement learning. In *IJCAI*, 2017.

Michael Matthews, Michael Beukman, Benjamin Ellis, Mikayel Samvelyan, Matthew Jackson, Samuel Coward, and Jakob Foerster. Craftax: A lightning-fast benchmark for open-ended reinforcement learning. *arXiv preprint arXiv:2402.16801*, 2024.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

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

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

Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-supervised exploration via disagreement. In *International conference on machine learning*, pp. 5062–5071. PMLR, 2019.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

Roberta Raileanu, Tim Rocktäschel, and Roberta Raileanu. 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.

Nikolay Savinov, Anton Raichuk, Damien Vincent, Raphael Marinier, Marc Pollefeys, Timothy Lillicrap, and Sylvain Gelly. Episodic curiosity through reachability. In *Proceedings of the International Conference on Learning Representations*, 2019.

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

Younggyo Seo, Lili Chen, Jinwoo Shin, Honglak Lee, Pieter Abbeel, and Kimin Lee. State entropy maximization with random encoders for efficient exploration. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 9443–9454, 2021.

Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

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

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

Adrien Ali Taiga, William Fedus, Marlos C Machado, Aaron Courville, and Marc G Bellemare. On bonus-based exploration methods in the arcade learning environment. *arXiv preprint arXiv:2109.11052*, 2021.

Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023. URL https://zenodo.org/record/8127025.

Kaixin Wang, Kuangqi Zhou, Bingyi Kang, Jiashi Feng, and YAN Shuicheng. Revisiting intrinsic reward for exploration in procedurally generated environments. In *The Eleventh International Conference on Learning Representations*, 2022.

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

Mingqi Yuan, Zequn Zhang, Yang Xu, Shihao Luo, Bo Li, Xin Jin, and Wenjun Zeng. Rllte: Long-term evolution project of reinforcement learning. *arXiv preprint arXiv:2309.16382*, 2023.

A Related Work

Intrinsic rewards often rely on heavily engineered implementations to stabilize their optimization. The latter difficulties reproducibility and causes different works in the literature to report varying performance in popular environments. Some works have benchmarked intrinsic rewards in specific environments Taiga et al. (2021); Wang et al. (2022); Laskin et al. (2021), yet they do not provide details on the importance of the design decisions in the implementation and optimization of the intrinsic rewards. In this work, we introduce **RLeXplore**, a more comprehensive framework that contains the most widely-used intrinsic rewards and provides the RL community with a unified framework to accelerate research and compare baselines in intrinsically-motivated RL. In the following, we overview existing formulations for intrinsic rewards of different natures and introduce the methods included in RLeXplore.

A.1 Count-Based Exploration

Count-based exploration methods provide intrinsic rewards by measuring the novelty of states, defined to be inversely proportional to the state visitation counts (Strehl & Littman, 2008; Tang et al., 2017; Machado et al., 2020; Jo et al., 2022). In finite state spaces, count-based methods perform near optimally (Strehl & Littman, 2008). For this reason, these methods have been established as appealing techniques for driving structured exploration in RL. However, they do not scale well to high-dimensional state spaces (Bellemare et al., 2016; Lobel et al., 2023). Pseudo-counts provide a framework to generalize count-based methods to high-dimensional and partially observed environments (Bellemare et al., 2016; Ostrovski et al., 2017; Martin et al., 2017). Burda et al. (2019b) proposed random network distillation (RND), which uses the prediction error against a fixed network as a learning signal that is correlated to counts. Recently, Henaff et al. (2022) used an elliptical bonus (E3B) and showed that such an objective provides a generalization of counts to high-dimensional spaces. In RLeXplore, we include Pseudo-counts, RND, and E3B as representatives of the state-of-the-art count-based methods.

A.2 Curiosity-Driven Exploration

Curiosity-based objectives train agents to interact with the environment seeking to experience outcomes that are not aligned with the agents' predictions (Aubret et al., 2023). Hence, curiosity-driven exploration usually involves training an agent to increase its knowledge about the environment (e.g., environment dynamics) (Stadie et al., 2015; Pathak et al., 2017; Yu et al., 2020). The intrinsic curiosity module (ICM) Pathak et al. (2017); Burda et al. (2019a) learns a joint embedding space with inverse and forward dynamics losses and was the first curiosity-based method successfully applied to deep RL settings. Disagreement Pathak et al. (2019) further extended ICM by using the variance over an ensemble of forward-dynamics models to compute curiosity. However, curiosity-driven methods are consistently found to be unsuccessful when the environment has irreducible noise (Savinov et al., 2019). To address the problem, Raileanu et al. (2020) proposed RIDE, which uses the difference between two consecutive state embeddings as the intrinsic reward and encourages the agent to choose actions that result in significant state changes. In general, curiosity-based objectives remain amongst the most popular intrinsic rewards in deep RL applications to this day. In RLeXplore, we include ICM, Disagreement, and RIDE as representatives of the state-of-the-art curiosity-driven methods.

A.3 Global and Episodic Exploration

Towards more general and adaptive agents, recent works have studied decision-making problems in contextual Markov decision processes (MDPs) (e.g., procedurally-generated environments) (Raileanu et al., 2020; Henaff et al., 2022; Matthews et al., 2024). Contextual MDPs require episodic-level exploration, where novelty estimates are reset at the beginning of each episode. Henaff et al. (2023) showed that both global and episodic exploration modalities have unique benefits and proposed combined objectives that achieve remarkable performance across many MDPs of different structures. NGU Badia et al. (2020) and RIDE Raileanu et al. (2020) also instantiate both global and episodic bonuses. Inspired by this recent line of works, in this paper, we study novel combinations of objectives for exploration that achieve impressive results in contextual MDPs.

B Details of RLeXplore

B.1 Architecture

The core design decision of RLeXplore involves decoupling the intrinsic reward modules from the RL optimization algorithms, which enables our intrinsic reward implementations to be integrated with any desired RL algorithm (or existing library, see Appendix D). Figure 2 illustrates the basic workflow of RLeXplore, which consists of two parts: data collection (i.e., policy rollout) and reward computation.

At each time step, the agent receives observations from the environment and predicts actions. The environment then executes the actions and returns feedback to the agent, which consists of a next observation, a reward, and a terminal signal. During the data collection process, the **`.watch()`** function is used to monitor the agent-environment interactions. For instance, E3B Henaff et al. (2022) updates an estimate of an ellipsoid in an embedding space after observing every state. At the end of the data collection rollouts, **`.compute()`** computes the corresponding intrinsic rewards. Note that **`.compute()`** is only called once per rollout using batched operations, which makes RLeXplore a highly efficient framework. Additionally, RLeXplore provides several utilities for reward and observation normalization. Finally, the **`.update()`** function is called immediately after **`.compute()`** to update the reward module if necessary (e.g., train the forward dynamics models in Disagreement Pathak et al. (2019) or the predictor network in RND Burda et al. (2019b)). Section D illustrates the usage of the aforementioned functions. All operations are subject to the standard workflow of the Gymnasium API (Towers et al., 2023).

We provide various examples of using RLeXplore with popular RL frameworks (e.g., stable-baselines3 Raffin et al. (2021) and CleanRL Huang et al. (2022)) in Appendix D.

B.2 Algorithmic Baselines

In RLeXplore, we implement eight widely-recognized intrinsic reward algorithms spanning the different categories described in Appendix A, namely ICM Pathak et al. (2017), RND Burda et al. (2019b), Disagreement Pathak et al. (2019), NGU Badia et al. (2020), PseudoCounts Badia et al. (2020), RIDE Raileanu et al. (2020), RE3 Seo et al. (2021), and E3B Henaff et al. (2022), respectively. We selected them based on the following tenet:

- The algorithm represents a unique design philosophy;
- The algorithm achieved superior performance on well-recognized benchmarks;
- The algorithm can adapt to arbitrary tasks and can be combined with arbitrary RL algorithms.

Detailed descriptions of each method are as follows: **ICM** (Pathak et al., 2017). ICM leverages a inverse-forward model to learn the dynamics of the environment and uses the prediction error as the curiosity reward. Specifically, the inverse model inferences the current action $\boldsymbol{a}_t$ based on the encoded states $\boldsymbol{e}_t$ and $\boldsymbol{e}_{t+1}$, where $\boldsymbol{e} = \psi(\boldsymbol{s})$ and $\psi(\cdot)$ is an embedding network. Meanwhile, the forward model f predicts the encoded next-state $\boldsymbol{e}_t$ based on $(\boldsymbol{e}_t, \boldsymbol{a}_t)$. Finally, the intrinsic reward is defined as

$$I_t = \|f(\boldsymbol{e}_t, \boldsymbol{a}_t) - \boldsymbol{e}_{t+1}\|_2^2. \tag{3}$$

RND (Burda et al., 2019b). RND produces intrinsic rewards via a self-supervised manner, in which a predictor network $\hat{f}$ is trained to approximate a fixed and randomly-initialized target network $\hat{f}$. As a result, the agent is motivated to explore unseen parts of the state space. The intrinsic reward is defined as

$$I_t = \|\hat{f}(\boldsymbol{s}_{t+1}) - f(\boldsymbol{s}_{t+1})\|_2^2. \tag{4}$$

Disagreement (Pathak et al., 2019). Disagreement is variant of ICM that leverages an ensemble of forward models and calculates the intrinsic reward as the variance among these models. Accordingly, the intrinsic reward is defined as

$$I_t = \text{Var}\{f_i(\boldsymbol{e}_t, \boldsymbol{a}_t)\}, i = 0, ..., N \tag{5}$$

NGU (Badia et al., 2020). NGU is a mixed intrinsic reward approach that combines global and episodic exploration and the first algorithm to achieve non-zero rewards in the game of *Pitfall!* without using demonstrations or hand-crafted features. The intrinsic reward is defined as

$$I_t = \min\{\max\{\alpha_t\}, C\}/\sqrt{N_{\text{ep}}(\boldsymbol{s}_t)}, \tag{6}$$

where α_t is a life-long curiosity factor computed following the RND method, C is a chosen maximum reward scaling, and N_{ep} is the episodic state visitation frequency computed by pseudo-counts.

PseudoCounts (Badia et al., 2020). Pseudo-counts has been widely used in count-based exploration approaches Bellemare et al. (2016); Ostrovski et al. (2017) with diverse implementations like neural density models. In this paper, we follow NGU Badia et al. (2020) that computes pseudo-counts via k-nearest neighbor estimation, which is highly efficient and can be applied to arbitrary task. Given the encoded observations $\{\boldsymbol{e}_0, \dots, \boldsymbol{e}_{T-1}\}$ visited in the an episode, we have

$$\sqrt{N_{\text{ep}}(\boldsymbol{s}_t)} \approx \sqrt{\sum_{\tilde{\boldsymbol{e}}_i} K(\tilde{\boldsymbol{e}}_i, \boldsymbol{e}_t)} + c, \tag{7}$$

where $\tilde{\boldsymbol{e}}_i$ is the first k nearest neighbors of $\boldsymbol{e}$, K is a Dirac delta function, and c guarantees a minimum amount of pseudo-counts. Finally, the intrinsic reward is defined as

$$I_t = 1/\sqrt{N_{\text{ep}}(\boldsymbol{s}_t)} \tag{8}$$

RIDE (Raileanu et al., 2020). RIDE is designed based on ICM that learns the dynamics of the environment and rewards significant state changes. Accordingly, the intrinsic reward is defined as

$$I_t = \|\boldsymbol{e}_{t+1} - \boldsymbol{e}_t\|_2/\sqrt{N_{\text{ep}}(\boldsymbol{s}_{t+1})}, \tag{9}$$

where $N_{\text{ep}}(\boldsymbol{s}_{t+1})$ is used to discount the intrinsic reward and prevent the agent from lingering in a sequence of states with a large difference in their embeddings.

RE3 (Seo et al., 2021). RE3 is an information theory-based and computation-efficient exploration approach, which aims to maximize the Shannon entropy of the state visiting distribution. In particular, RE3 leverages a random and fixed neural network to encode the state space and employs a k-nearest neighbor estimator to estimate the entropy efficiently. Then the estimated entropy is transformed into particle-base intrinsic rewards. Specifically, the intrinsic reward is defined as

$$I_t = \frac{1}{k}\sum_{i=1}^{k} \log(\|\boldsymbol{e}_t - \tilde{\boldsymbol{e}}_t^i\|_2 + 1). \tag{10}$$

E3B (Henaff et al., 2022). E3B provides a generalization of count-based rewards to continuous spaces using an elliptical bonus. E3B learns a representation mapping from observations to a latent space (e.g. using inverse dynamics). At each episode, the sequence of latent observations parameterize an ellipsoid which is used to measure the novelty of the subsequent observations. In tabular settings, the E3B ellipsoid reduces to the table of inverse state-visitation frequencies (Henaff et al., 2022). The reward of a new observation is computed as:

$$I_t = f(\boldsymbol{e}_t)^T C_{t-1} f(\boldsymbol{e}_t) \tag{11}$$

where f is the learned representation mapping and C_{t-1} is the episodic ellipsoid.

C Experimental Settings

C.1 Baselines

We designed the following settings for the baseline experiments, and all the subsequent RQs were progressively adjusted based on the baselines. Moreover, all the experiments are performed using the proximal policy optimization (PPO) Schulman et al. (2017) implementation from RLLTE (Yuan et al., 2023).

Hyperparameter	**Value**
Observation normalization	image = image / 255.0
Reward normalization	RMS
Weight initialization	Orthogonal
Update proportion	1.0
with LSTM	False

Table 1: Details of baseline settings.

Hyperparameter	**SuperMarioBros**	**Procgen**
Observation downsampling	(84, 84)	(64, 64)
Stacked frames	No	No
Environment steps	10000000	25000000
Episode steps	128	256
Number of workers	1	1
Environments per worker	8	64
Optimizer	Adam	Adam
Learning rate	2.5e-4	5e-4
GAE coefficient	0.95	0.95
Action entropy coefficient	0.01	0.01
Value loss coefficient	0.5	0.5
Value clip range	0.1	0.2
Max gradient norm	0.5	0.5
Epochs per rollout	4	3
Batch size	256	2048
Discount factor	0.99	0.999

Table 2: PPO hyperparameters for *SuperMarioBros* and *Procgen* games.

C.2 Details of RQs

Table 3 illustrates the details of the candidates for all RQs. For RQ1-5, we designed the experiments sequentially and modified the configuration for each intrinsic reward based on the best results of previous RQs. For instance, experiments of RQ1 only change the technique of observation normalization, and RQ2 will use the best observation normalization method for each reward module obtained in RQ1. Likewise, RQ3 will follow the best results obtained in RQ1-2 and only change the proportion of samples used for model update. However, we kept using the baselines settings for each reward in RQ8 to explore the most original performance of the mixed intrinsic rewards.

#	Candidate	Detail
RQ1	Min-Max	image = image / 255.0
	RMS	image = Clip $\left(\frac{\text{image}-\text{running mean}}{\text{running std.}}, -5.0, 5.0\right)$
RQ2	Vanilla	$\mathbf{I} = \mathbf{I}$
	RMS	$\mathbf{I} = \frac{\mathbf{I}}{\text{running std}}$
	Min-Max	$\mathbf{I} = \frac{\mathbf{I}-\min(\mathbf{I})}{\max(\mathbf{I})-\min(\mathbf{I})}$
RQ3	0.1	Use 10% of the samples to update the intrinsic reward module.
	0.5	Use 50% of the samples to update the intrinsic reward module.
	1.0	Use 100% of the samples to update the intrinsic reward module.
RQ4	Vanilla	Fill the input tensor with values drawn from the uniform distribution.
	Orthogonal	Fill the input tensor with a (semi) orthogonal matrix.
RQ5	Vanilla	Policy network with only convolutional and linear layers.
	LSTM	Policy network that includes an LSTM layer.
RQ6	Vanilla	$R = E + I$
	Two-head	Value network uses two separate branches for E and I.
RQ7	N/A	N/A
RQ8	Global+Episodic	E3B+RND, E3B+ICM, E3B+RIDE, RE3+RND, RE3+ICM, RE3+RIDE
	Global+Global	RND+ICM, RND+RIDE, ICM+RIDE

Table 3: Details of candidates for all RQs, where **I** is a batch of intrinsic rewards.

C.3 Best Configurations

Reward	Obs. Norm.	Reward Norm.	Update Prop.	Weight Init.	Memory Required
PseudoCounts	✓	Min-Max	0.5	Orthogonal	✗
ICM	✗	RMS	1.0	Orthogonal	✗
RND	✓	Vanilla	0.1	Orthogonal	✗
E3B	✓	Min-Max	0.1	Orthogonal	✗
RIDE	✓	Min-Max	0.1	Default	✓
RE3	✗	Vanilla	N/A	Orthogonal	✗
NGU	✓	Min-Max	0.1	Orthogonal	✗
Disagreement	✓	Min-Max	1.0	Orthogonal	✗

Table 4: The best configurations for each intrinsic reward on *SuperMarioBros* games.

D Usage Examples

D.1 Workflow of RLeXplore

The following code provides an example when using RLeXplore with on-policy algorithms. At each time step, the agent first observes the vectorized environments before making actions. Then the environments execute the actions and return the step information, which is processed by the **.watch()** function to extract necessary data for the current intrinsic reward. Finally, the intrinsic rewards will be computed and the module will updated concurrently at the end of the episode.

```
# load the library
from rllte.xplore.reward import RE3
# create the reward module
irs = RE3(...)
# reset the environment
obs, infos = envs.reset()
# a rollout storage
rs = RolloutStorage(...)
# training loop
for episode in range(...):
    for step in range(...):
        # sample actions
        actions = agent(obs)
        # step the environment
        next_obs, rwds, terms, truncs, infos = envs.step(actions)
        # get data from the transitions
        irs.watch(obs, actions, rwds, next_obs, terms, truncs, infos)
        ...
    # prepare the samples
    samples = dict(observations=rs.obs, actions=rs.actions,
                   rewards=rs.rewards, terminateds=rs.terminateds,
                   truncateds=rs.truncateds, next_observations=rs.next_obs
    )
    # compute the intrinsic rewards
    ## sync (bool): Whether to update the reward module after the
    ## `compute` function, default is `True`.
    intrinsic_rewards = irs.compute(samples, sync=True)
```

In contrast, the workflow is a bit different when using RLeXplore with off-policy algorithms. As shown in the following example, the intrinsic reward will computed at each time step rather than the end of each episode. Moreover, the intrinsic reward module will be updated using the same samples for policy update.

```
# load the library
from rllte.xplore.reward import RE3
# create the reward module
irs = RE3(...)
# reset the environment
obs, infos = envs.reset()
# training loop
while True:
    # sample actions
    actions = agent(obs)
    # step the environment
    next_obs, rwds, terms, truncs, infos = envs.step(actions)
    # get data from the transitions
    irs.watch(obs, actions, rwds, next_obs, terms, truncs, infos)
    # compute the intrinsic rewards at each step
    ## sync (bool): Whether to update the reward module after the
    ## `compute` function, default is `True`
    intrinsic_rewards = irs.compute(
        samples=dict(observations=obs, actions=actions,
                     rewards=rwds, terminateds=terms,
                     truncateds=terms, next_observations=next_obs),
        sync=False)
    ...
    # update the reward module
    batch = replay_storage.sample()
    irs.update(samples=dict(observations=batch.obs,
                            actions=batch.actions,
                            rewards=batch.rewards,
                            terminateds=batch.terminateds,
                            truncateds=batch.truncateds,
                            next_observations=batch.next_obs)
    )
    ...
```

D.2 RLeXplore with Stable-Baselines3

Stable-Baselines3 (SB3) Raffin et al. (2021) is one of the most successful and popular RL frameworks that provides a set of reliable implementations of RL algorithms in Python. SB3 provides a convenient callback function that can be called at given stages of the training procedure, the following code example demonstrates how to use RLeXplore in SB3 for on-policy RL algorithms:

```
class RLeXploreWithOnPolicyRL(BaseCallback):
    """
    Combining RLeXplore and on-policy algorithms from SB3.
    """
    def __init__(self, irs, verbose=0):
        super(RLeXploreWithOnPolicyRL, self).__init__(verbose)
        self.irs = irs
        self.buffer = None

    def init_callback(self, model: BaseAlgorithm) -> None:
        super().init_callback(model)
        self.buffer = self.model.rollout_buffer

    def _on_step(self) -> bool:
        """
        This method will be called by the model after each call to `env.step()`.

        :return: (bool) If the callback returns False, training is aborted early.
        """
        observations = self.locals["obs_tensor"]
        device = observations.device
        actions = th.as_tensor(self.locals["actions"], device=device)
        rewards = th.as_tensor(self.locals["rewards"], device=device)
        dones = th.as_tensor(self.locals["dones"], device=device)
        next_observations = th.as_tensor(self.locals["new_obs"], device=device)

        # get data from the transitions
        self.irs.watch(observations, actions, rewards, dones, dones,
                                                  next_observations)

        return True

    def _on_rollout_end(self) -> None:
        # prepare the data samples
        obs = th.as_tensor(self.buffer.observations)
        # get the new observations
        new_obs = obs.clone()
        new_obs[:-1] = obs[1:]
        new_obs[-1] = th.as_tensor(self.locals["new_obs"])
        actions = th.as_tensor(self.buffer.actions)
        rewards = th.as_tensor(self.buffer.rewards)
        dones = th.as_tensor(self.buffer.episode_starts)
        print(obs.shape, actions.shape, rewards.shape, dones.shape, obs.shape)
        # compute the intrinsic rewards
        intrinsic_rewards = irs.compute(
            samples=dict(observations=obs, actions=actions,
                         rewards=rewards, terminateds=dones,
                         truncateds=dones, next_observations=new_obs),
```

More detailed code examples can be found in the attached supplementary materials.

D.3 RLeXplore with CleanRL

CleanRL Huang et al. (2022) is an open-source project focused on implementing RL algorithms with clean, understandable, and reproducible code. It aims to make RL more accessible by providing implementations that are simpler and more transparent than those typically found in research papers or larger libraries. The following code example demonstrates how to use RLeXplore in CleanRL for on-policy RL algorithms:

```
# load the library
from rllte.xplore.reward import RE3
# create the reward module
irs = RE3(envs=envs, device=device)
...
# get data from the transitions
irs.watch(observations=obs[step], actions=actions[step],
          rewards=rewards[step], terminateds=dones[step],
          truncateds=dones[step], next_observations=next_obs
          )
...
next_obs = obs.clone()
next_obs[:-1] = obs[1:]
next_obs[-1] = next_obs
# compute the intrinsic rewards
intrinsic_rewards = irs.compute(
    samples=dict(observations=obs, actions=actions,
                 rewards=rewards, terminateds=dones,
                 truncateds=dones, next_observations=next_obs),
    sync=True)
# add the intrinsic rewards to the rewards
rewards += intrinsic_rewards
```

More detailed code examples can be found in the attached supplementary materials.

E Learning Curves

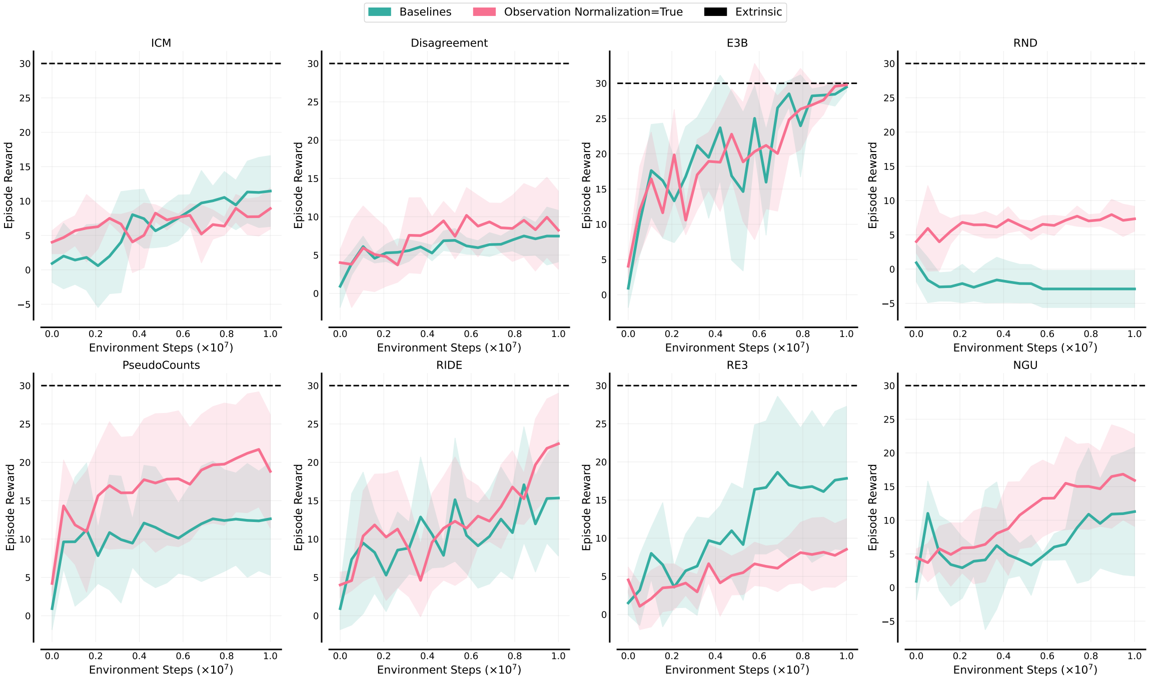

Figure 8: Learning curves of the baselines and RQ1 on *SuperMarioBros-1-1-v3*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

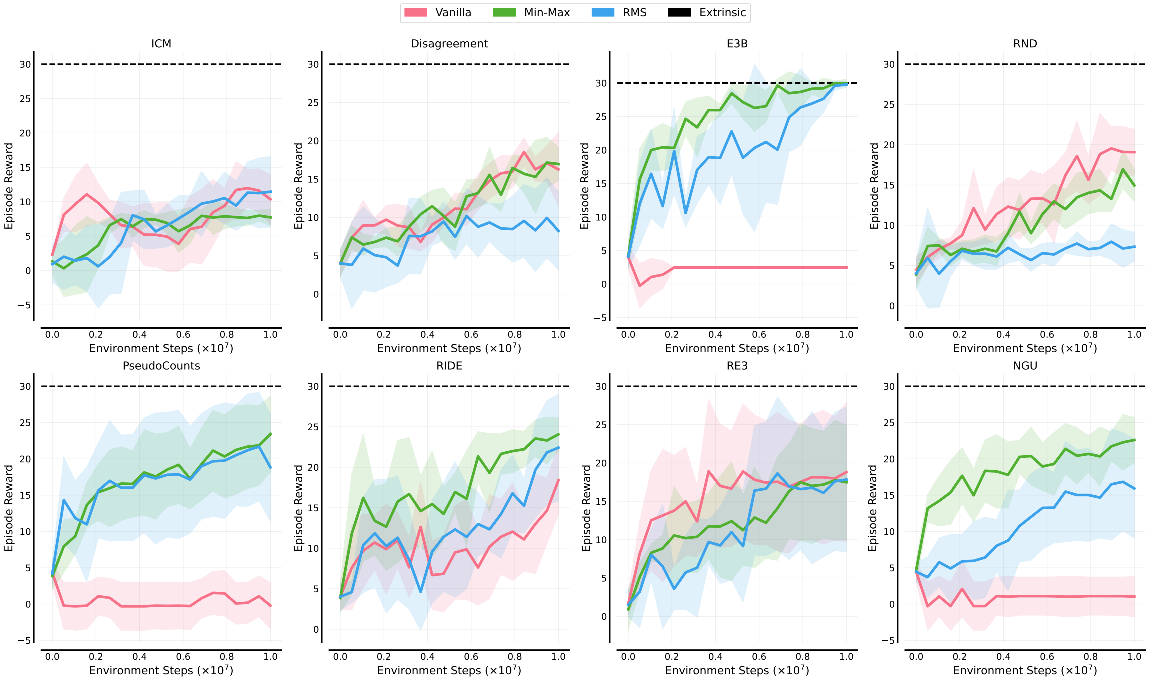

Figure 9: Learning curves of the RQ2 on *SuperMarioBros-1-1-v3*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

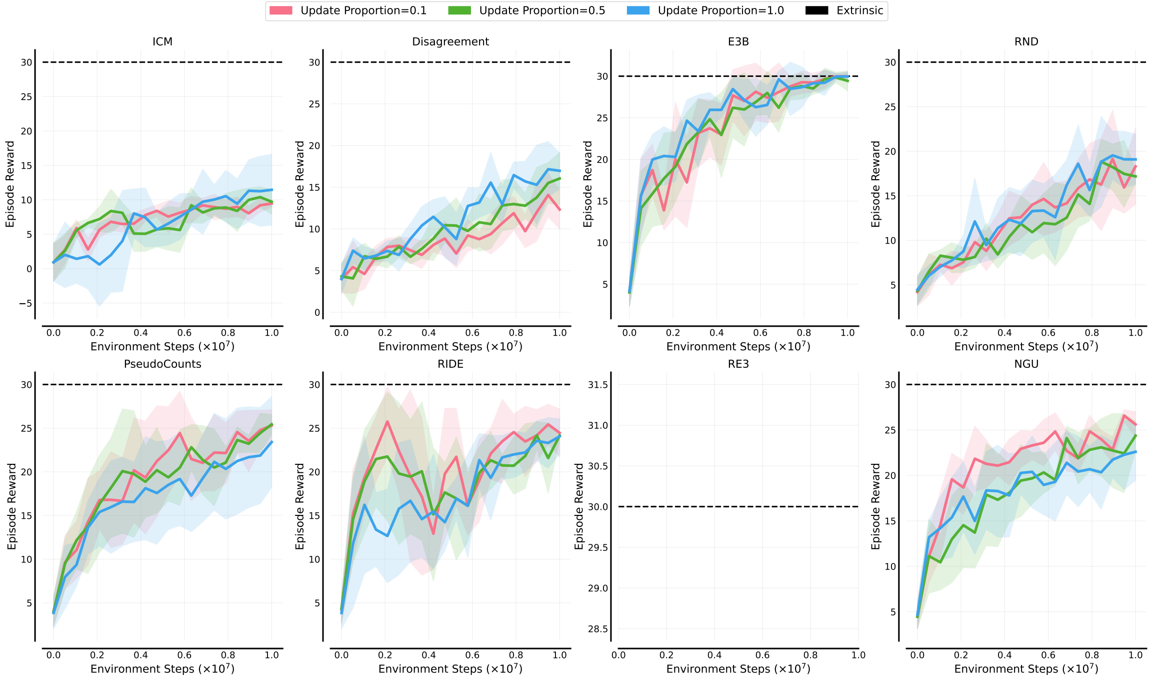

Figure 10: Learning curves of the RQ3 on *SuperMarioBros-1-1-v3*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

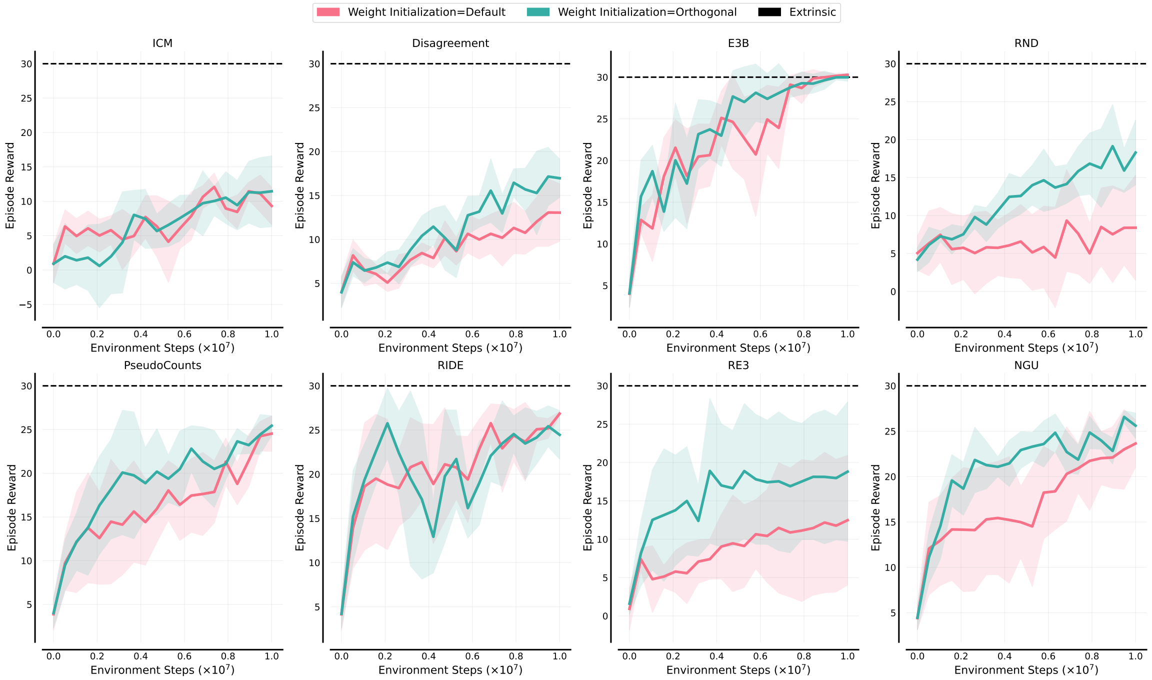

Figure 11: Learning curves of the RQ4 on *SuperMarioBros-1-1-v3*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

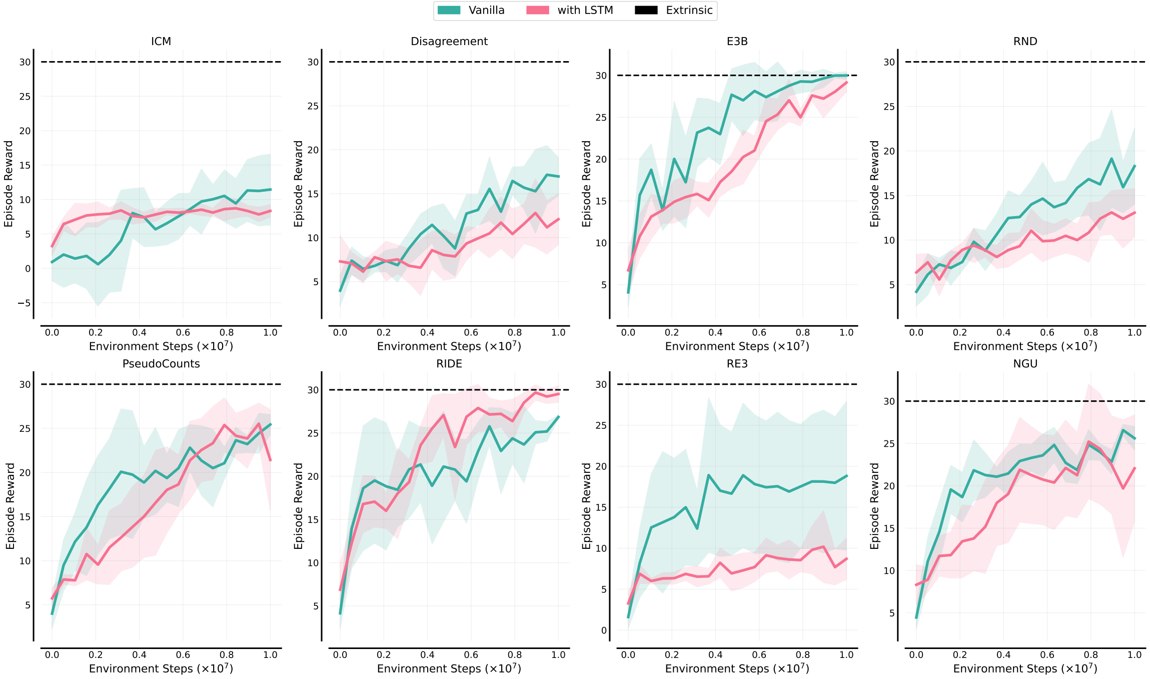

Figure 12: Learning curves of the RQ5 on *SuperMarioBros-1-1-v3*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

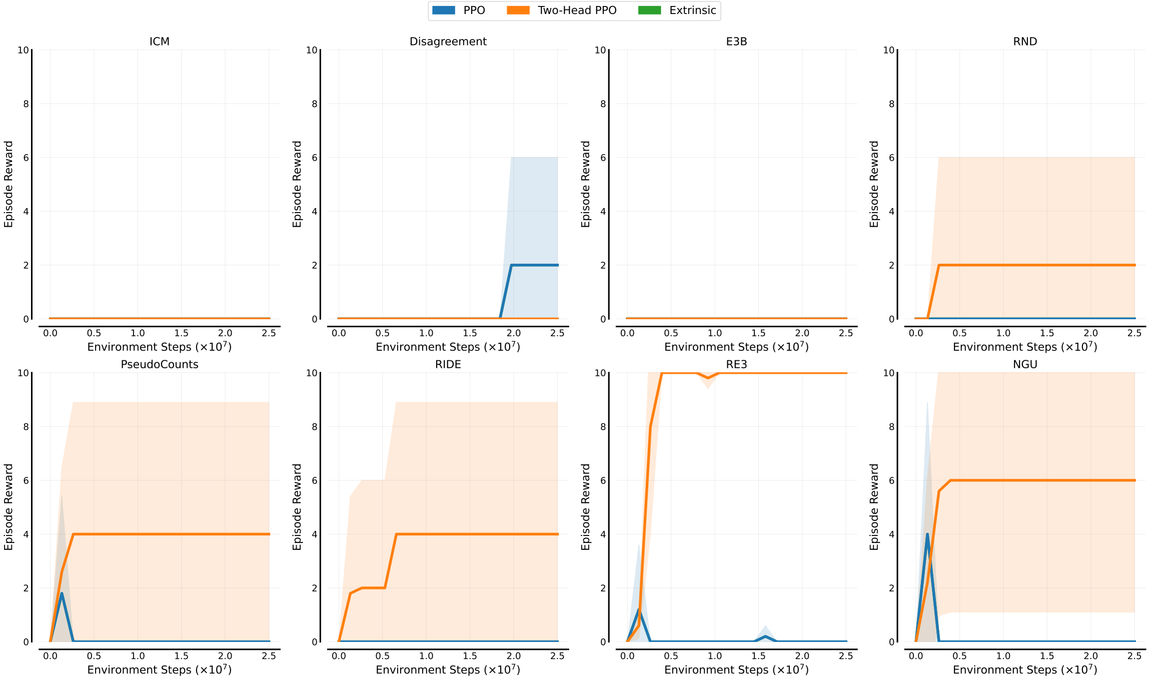

Figure 13: Learning curves of RQ6 on *Procgen-1MazeHard*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

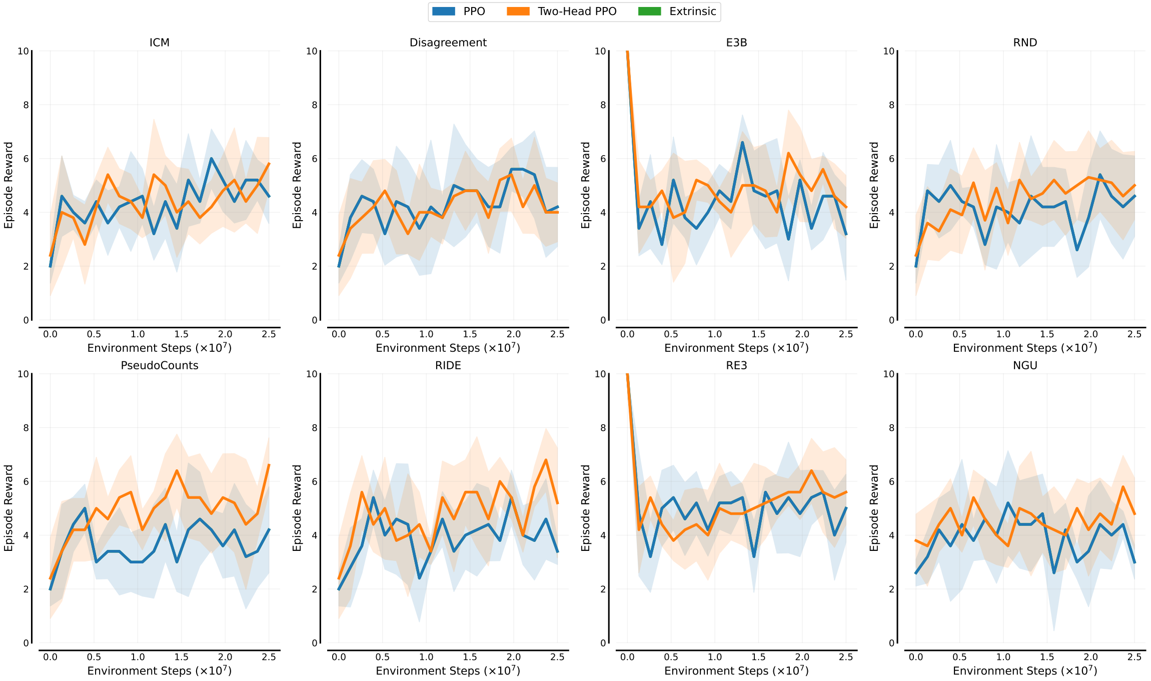

Figure 14: Learning curves of RQ6 on *Procgen-AllMazeHard*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

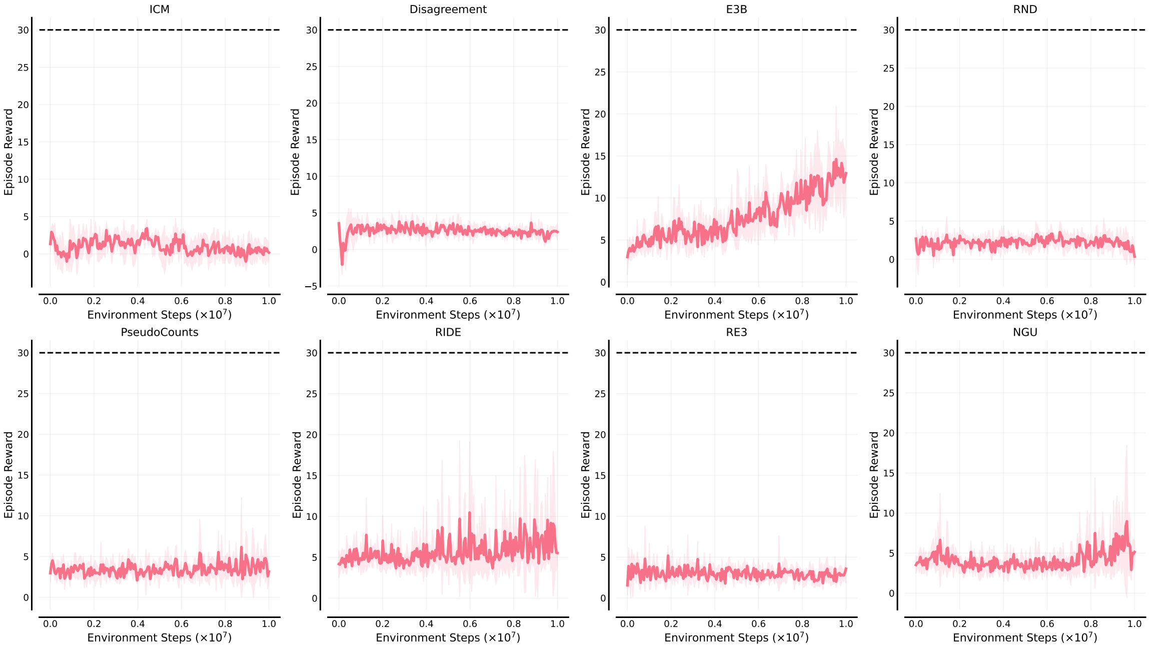

Figure 15: Learning curves of RQ7 on *SuperMarioBrosRandomStages-v3*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

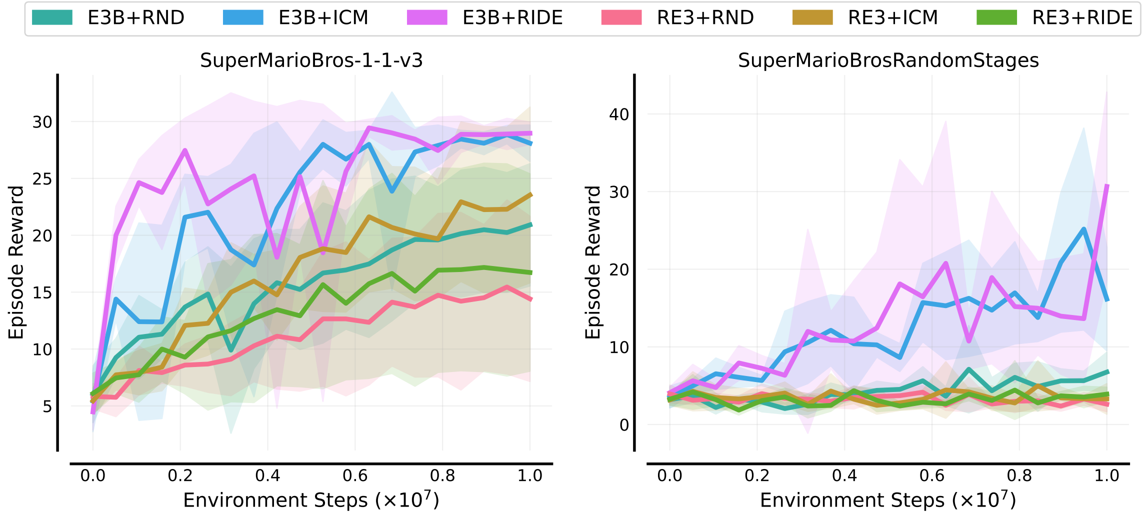

Figure 16: Learning curves of RQ8 (global+episodic exploration) on *SuperMarioBros-1-1-v3* and *SuperMarioBrosRandomStages-v3*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

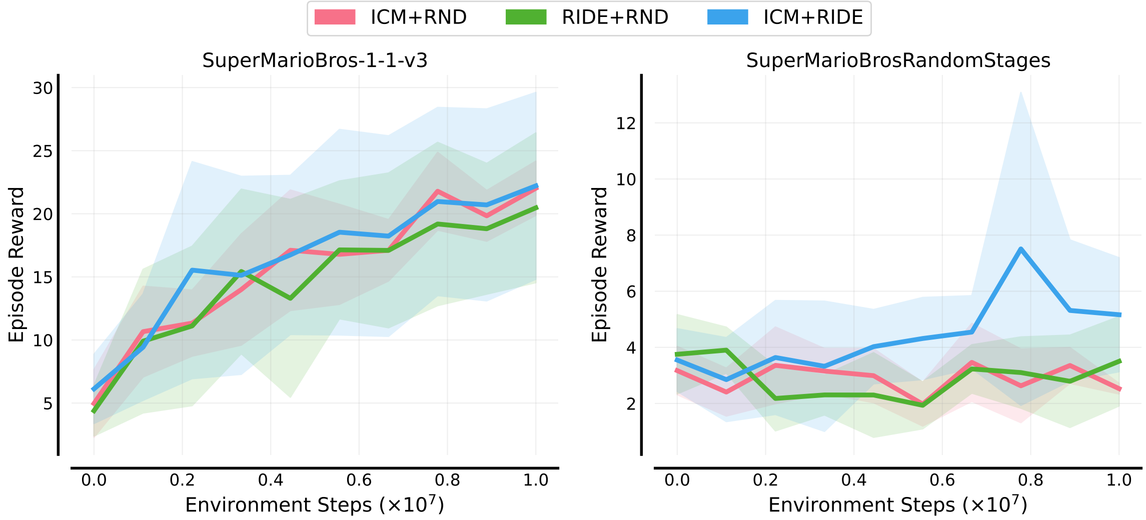

Figure 17: Learning curves of RQ8 (global+global exploration) on *SuperMarioBros-1-1-v3* and *SuperMarioBrosRandomStages-v3*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

F On-Policy RL Algorithms and Discrete Control Tasks

In this section, we demonstrate the combination of RLeXplore and on-policy RL algorithms and its effectiveness on discrete control tasks. Specifically, we couple the PPO algorithm and intrinsic rewards, and evaluate their performance on *Montezuma's Revenge*, a hard exploration task from the Atari benchmark (Bellemare et al., 2013). Meanwhile, we utilize the PPO implementation of CleanRL Huang et al. (2022) to show the adaptability of RLeXplore. Table 5 illustrates the training hyperparameters used for the experiments.

Part	Hyperparameter	Value
PPO	Observation downsampling	(84, 84)
	Stacked frames	4
	Environment steps	1e+8
	Episode steps	128
	Number of workers	1
	Environments per worker	8
	Optimizer	Adam
	Learning rate	1e-4
	GAE coefficient	0.95
	Action entropy coefficient	0.01
	Value loss coefficient	0.5
	Value clip range	0.1
	Max gradient norm	0.5
	Epochs per rollout	4
	Batch size	256
	Discount factor	0.99
Intrinsic reward	Observation normalization	RMS
	Reward normalization	RMS
	Weight initialization	Orthogonal
	Update proportion	0.25
	with LSTM	False

Table 5: Training hyperparameters for *Montezuma's Revenge*.

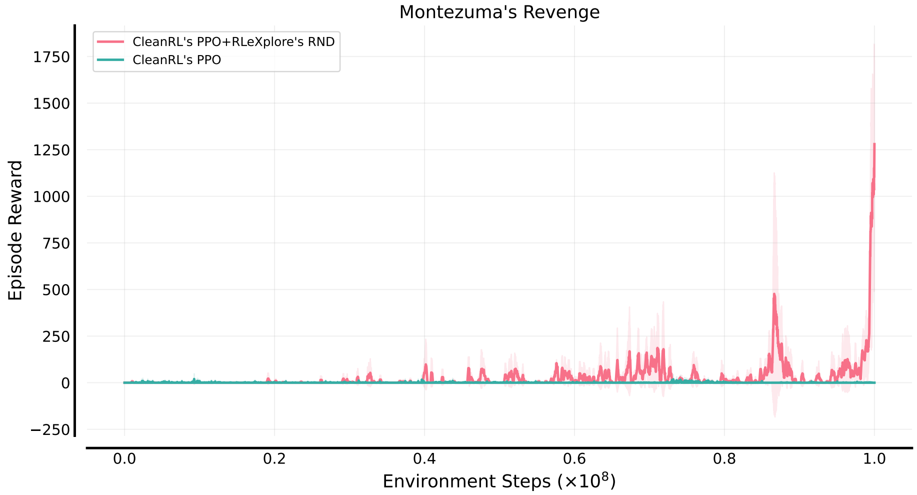

Figure 18: Since only RND can achieve significant results in this task among the eight intrinsic rewards, we only show the results of RND. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

G Off-Policy RL Algorithms and Continuous Control Tasks

To showcase the generality of RLeXplore, we run additional experiments in settings different from the ones in the main paper. Concretely, we couple intrinsic rewards with soft actor-critic (SAC) Haarnoja et al. (2018), an off-policy RL algorithm, and test their performance in *Ant-UMaze*, a continuous control task with sparse rewards. Table 6 illustrates the training hyperparameters used for the experiments. We show the performance of Disagreement, RND, ICM and vanilla SAC in Figure 19. The results indicate that intrinsically-motivated agents are able to navigate the maze more efficiently, finding the goals more often than the vanilla agents that can only learn from the sparse task rewards.

We only use 3 intrinsic rewards with SAC because of the episodic nature of the other intrinsic reward methods. For example, the episodic memory in RIDE, PseudoCounts, NGU; and the episodic ellipsoid in E3B require the replay buffer to sample entire episodes instead of random rollouts. We aim to implement this logic in the future in our RLeXplore codebase.

Part	**Parameter**	**Value**
	Total timesteps	$1 \cdot 10^6$
	Buffer size	$1 \cdot 10^6$
	Discount (γ)	0.99
	Target smoothing coefficient (τ)	0.005
	Batch size	256
	Learning starts	5000
	Policy learning rate	$3 \cdot 10^{-4}$
	Q function learning rate	$1 \cdot 10^{-3}$
	Policy frequency	2
	Target network frequency	1
	Noise clip	0.5
	Entropy coefficient (α)	0.2
	Auto-tune entropy coefficient	True
Intrinsic reward	Observation normalization	RMS
	Reward normalization	RMS
	Weight initialization	Orthogonal
	Update proportion	0.25
	with LSTM	False

Table 6: Training hyperparameters for *Ant-Umaze*.

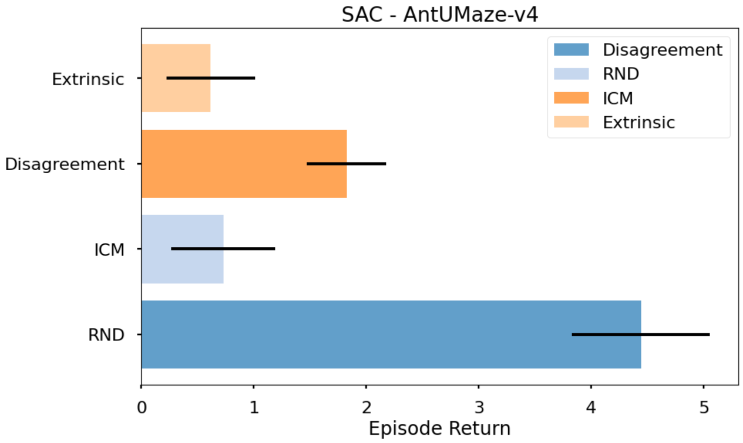

Figure 19: Performance comparison between the three selected intrinsic rewards and the extrinsic reward.