

RLeXplore: Accelerating Research in Intrinsically-Motivated Reinforcement Learning

^{1*}Mingqi Yuan, ^{2*}Roger Creus Castanyer, ¹Bo Li, ³Xin Jin, ²Glen Berseth, ³Wenjun Zeng

¹Department of Computing, The Hong Kong Polytechnic University, China

²Mila Québec AI Institute & Université de Montréal, Canada

³Eastern Institute of Technology, Ningbo, China

mingqi.yuan@connect.polyu.hk

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>.

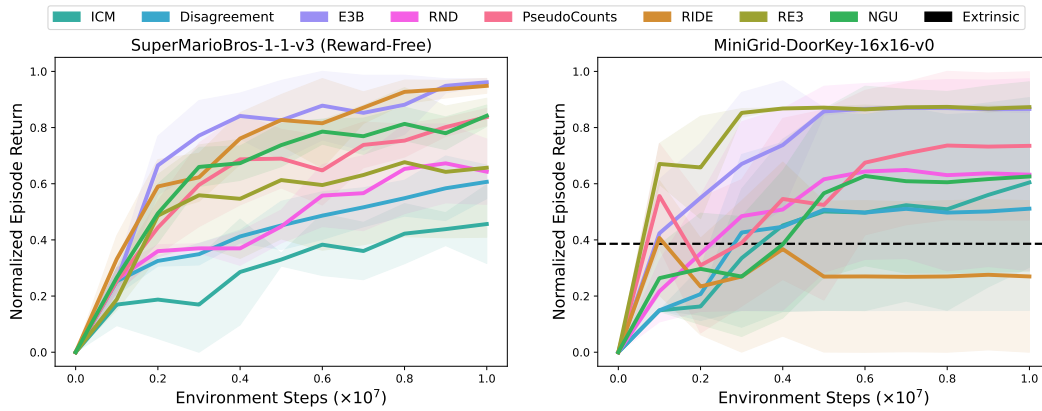


Figure 1: Episode returns achieved by the intrinsic rewards in RLeXplore. (left) *SuperMarioBros-1-1* without access to the task rewards. (right) *MiniGrid-DoorKey-16x16* with sparse rewards.

*Equal Contribution.

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 [1]. 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 [2].

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 [3]. To address this challenge, the concept of intrinsic rewards has emerged as a promising approach in the RL community [4, 5, 6, 7, 8, 9]. 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.

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 [5, 7, 10], 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. **RLeXplore** offers a plug-and-play framework for researchers working on intrinsically-motivated RL, enabling them to seamlessly integrate state-of-the-art (SOTA) intrinsic rewards into their projects. Specifically, **RLeXplore**: (1) facilitates fair comparisons across multiple baselines; (2) can be easily integrated with various RL frameworks; and (3) streamlines the development of new intrinsic reward algorithms. In Table 1 we compare the performance of the implementations in **RLeXplore** with the original results reported in previous works. In Appendix F we provide the full details on reproducibility with **RLeXplore**.

Table 1: Summary of comparative results from **RLeXplore** implementations and reported results in previous works. Please refer to Appendix F for full details on reproducibility.

Environment	Algorithm	Original	RLeXplore
SuperMarioBros	RIDE	23%	50%
SuperMarioBros	ICM	30%	30%
MiniGrid-DoorKey-16x16	ICM	0.00	0.60
MiniGrid-DoorKey-16x16	RND	0.00	0.60
MiniGrid-DoorKey-16x16	RIDE	0.25	0.12
MiniGrid-DoorKey-8x8	RE3	0.50	0.95
MiniGrid-DoorKey-8x8	RND	0.00	0.00
MiniGrid-DoorKey-8x8	ICM	0.20	0.83
Procgen - 200 Mazes	E3B	3.00	4.10
Procgen - 200 Mazes	ICM	2.50	5.90
Procgen - 200 Mazes	RND	1.70	5.00

To support these capabilities, we have provided extensive documentation* that includes detailed guides on using **RLeXplore**, along with comprehensive code tutorials†. These resources are designed to make it straightforward for users to get started with **RLeXplore**, regardless of their prior experience with intrinsic rewards in RL. In Appendix F.6 we provide an overview of the main differences and advantages of **RLeXplore** compared to existing RL libraries.

*<https://docs.rllte.dev/tutorials/mt/irs/>

†<https://github.com/RLE-Foundation/RLeXplore#tutorials>

48 We aim for the community to adopt RLeXplore as a standard tool for evaluating intrinsic reward
49 methods, reducing implementation efforts and mitigating inconsistencies in results and conclusions.

50 Our work presents a systematic study aimed at addressing gaps in understanding the critical imple-
51 mentation and optimization details of intrinsic rewards. To guide our investigation, we formulate
52 numerous questions, aiming to uncover the intricacies of intrinsic rewards and their impact on RL
53 agent performance. Our results highlight the importance of thoughtful implementation design for
54 intrinsic rewards, showing that naive implementations can lead to suboptimal performance. Through
55 carefully studied design decisions, we demonstrate significant performance gains.

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

62 2 Background

63 We frame the RL problem considering a MDP [11, 12] defined by a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, E, P, d_0, \gamma)$,
64 where \mathcal{S} is the state space, \mathcal{A} is the action space, and $E : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the extrinsic reward
65 function, $P : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$ is the transition function that defines a probability distribution over \mathcal{S} ,
66 $d_0 \in \Delta(\mathcal{S})$ is the distribution of the initial observation s_0 , and $\gamma \in [0, 1)$ is a discount factor. The
67 goal of RL is to learn a policy $\pi_\theta(a|s)$ to maximize the expected discounted return:

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

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

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

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

71 We provide a comprehensive review of related work in Section B, where we discuss the background
72 and context of intrinsic reward methods. In Section C, we present a detailed overview of the SOTA
73 intrinsic reward methods that we implement in RLeXplore.

74 3 RLeXplore

75 In this section, we present **RLeXplore**, a unified, highly-modularized and plug-and-play framework
76 that currently provides high-quality and reliable implementations of eight state-of-the-art intrinsic
77 reward algorithms[‡]. Comparing multiple intrinsic reward methods under fair conditions is challenging
78 due to various confounding factors, such as using distinct backbone RL algorithms (e.g., PPO [13],
79 DQN [14], IMPALA [15]), optimization (e.g., reward and observation normalization, network
80 architecture) and evaluation details (e.g., environment configuration, algorithm hyperparameters).
81 RLeXplore is designed to provide a unified framework with standardized procedures for implementing,
82 computing, and optimizing intrinsic rewards.

83 3.1 Architecture

84 The core design decision of RLeXplore involves decoupling the intrinsic reward modules from the RL
85 optimization algorithms, which enables our intrinsic reward implementations to be integrated with

[‡]RLeXplore complies with the MIT License.

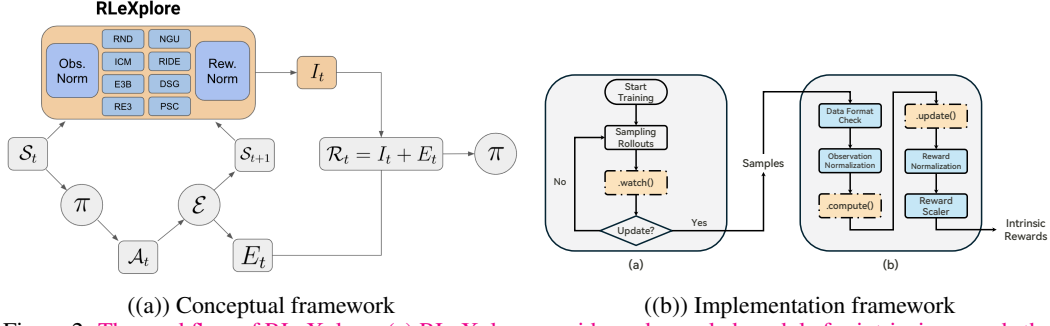


Figure 2: The workflow of RLeXplore. (a) RLeXplore provides a decoupled module for intrinsic rewards that integrates seamlessly with the RL training loop. RLeXplore implements 8 SOTA intrinsic rewards and several utilities for processing the observations and intrinsic rewards, while adapting to the unmodified RL training loop. (b) RLeXplore monitors the agent-environment interactions and gathers data samples using the `.watch()` function. (b) After experience rollouts are collected, RLeXplore computes the corresponding intrinsic rewards using the `.compute()` function and updates the auxiliary models via the `.update()` function.

any desired RL algorithm (or existing library, see Appendix E and the official integration examples[§]). Figure 2 illustrates the basic workflow of RLeXplore, which consists of two parts: data collection (i.e., policy rollout) and reward computation.

Commonly, 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 [8] 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 [9] or the predictor network in RND [4]). Section E illustrates the usage of the aforementioned functions. All operations are subject to the standard workflow of the Gymnasium API [16].

RLeXplore offers several benefits to the research community:

- For researchers seeking reliable tools for benchmarking and general applications: RLeXplore provides high-quality implementations of popular intrinsic reward algorithms, useful in both research and practical applications. It can be seamlessly integrated with existing RL libraries. We provide specific examples of integrating RLeXplore with Stable Baselines3 [17], CleanRL [18], and RLLTE [19] in Appendix E.
- For developers experimenting with new intrinsic rewards: RLeXplore offers modular components, such as various embedding networks, and a standardized workflow. This setup facilitates the creation, modification, and testing of new ideas. Detailed examples are available in the code repository and documentation.
- For promoting collaboration and accelerating progress: We have published a space using Weights & Biases (W&B) to store reusable experiment results on recognized benchmarks. This initiative aims to enhance collaboration within the research community and speed up progress by providing easy access to established benchmark results.

[§]<https://github.com/RLE-Foundation/RLeXplore#tutorials>

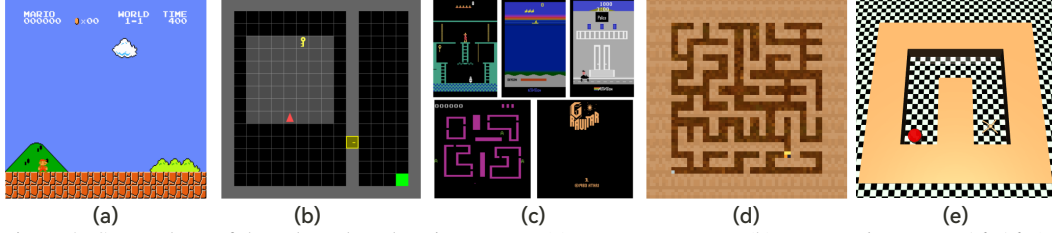


Figure 3: Screenshots of the selected exploration games. (a) *SuperMarioBros*. (b) *MiniGrid-DoorKey16x16*. (c) *ALE-5*. (d) *Progen-Maze*. (e) *Ant-UMaze*.

3.2 Algorithmic Baselines

In RLeXplore, we implement eight widely-recognized intrinsic reward algorithms spanning the different categories described in Section B, namely ICM [5], RND [4], Disagreement [9], NGU [7], PseudoCounts [7], RIDE [6], RE3 [20], and E3B [8], 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.

For detailed descriptions of each method we refer the reader to Appendix C.

4 Experiments

Our experiments aim to achieve two main objectives: (i) evaluating the effectiveness of our implementations in training exploratory agents, and (ii) assessing their performance across various sparse-reward environments to demonstrate the generality and robustness of our framework. First, we use *SuperMarioBros* without access to the environment’s rewards to study the low-level implementation details of intrinsic reward methods that drive robust exploration. We selected *SuperMarioBros* because effective exploration within this environment strongly correlates with task performance, making it an excellent benchmark for measuring the efficacy of exploration techniques. This environment has been widely used in previous studies on exploration in reinforcement learning [9, 6, 2]. To further generalize our findings, we also use the *MiniGrid-Doorkey16x16* environment, which is challenging due to the sparse rewards, making it difficult to solve with classical RL algorithms[¶]. The effectiveness of intrinsic rewards in *MiniGrid* environments has also been highlighted in prior works [6, 8, 21]. With these 2 environments we aim to study the implementation details in both reward-free and sparse-reward tasks.

Secondly, to showcase the generalizability of RLeXplore, we evaluate our implementations in additional sparse-reward environments, including Progen, MiniGrid, Ant-UMaze, and the set of five hard-exploration games in the Arcade Learning Environment (ALE) suite. These experiments are designed to test how well our methods balance the use of dense intrinsic rewards with sparse extrinsic rewards across a variety of tasks. The complete set of learning curves for all the experiments are shown in Appendix H and Appendix L.

Lastly, we explore recent advancements in using combined intrinsic rewards [21] to enhance exploration in contextual MDPs. Specifically, we use the full set of levels in *SuperMarioBros* to evaluate how well both single and combined intrinsic rewards can explore various game versions and generalize their exploration across different levels.

In the following sections, we present results from *SuperMarioBros* and *MiniGrid* for objective (i) and from *Progen-Maze* for objective (ii). Additionally, in Appendix F we show that using RLeXplore

[¶]<https://minigrid.farama.org/environments/minigrid/DoorKeyEnv/#description>

we are able to reproduce and improve the performance reported in previous works for many intrinsic rewards and across multiple environments.

The design of these experiments is driven by our primary goal: to provide a general and reliable set of intrinsic reward implementations within a user-friendly framework. Instead of attempting to benchmark all algorithms across every possible domain, we focus on verifying the generality of each method within a carefully selected subset of popular exploration tasks.

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 details impact the training of intrinsically-motivated agents. We first define an initial baseline configuration for optimizing the intrinsic rewards, shown in Table 2. 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.

Table 2: Details of baseline settings.

Hyperparameter	Value
Observation norm.	$\neq 255.0$
Reward norm.	RMS
Weight init.	Orthogonal
Update proportion with LSTM	1.0
	False

In this section, we conduct reward-free experiments (i.e., without access to extrinsic rewards) using the *SuperMarioBros* environment [22]. *SuperMarioBros* is a widely used benchmark for evaluating exploration in RL [5, 6], 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 D.

Importantly, as shown in Figure 2, we keep the PPO hyperparameters fixed and the overall RL training loop unmodified throughout all the experiments in the paper in order to isolate the effect of the RQs on the intrinsic reward components. Previous work has shown that PPO has many implementation details that are key for it to achieve great performance [23, 24]. In the following, we study implementation details for the intrinsic reward components. The fixed PPO hyperparameters are shown in Table 3.

RQ 1: The impact of observation normalization.

Observation normalization is crucial in deep learning to avoid numerical instabilities during optimization. Image observations, where each pixel value typically ranges from 0 to 255 per color channel, are commonly normalized to a range of 0 to 1 using Min-Max normalization by dividing each pixel value by 255. However, previous studies suggest that Min-Max normalization may not be ideal for all representation learning algorithms [4].

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) for the inputs to the intrinsic reward modules. RMS normalizes observations by subtracting the running mean and dividing by the running standard deviation of all observations collected by the agent thus far. 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, benefit from RMS normalization. Critically, RND achieves 0 rewards if observations are not normalized with RMS. These results indicate that RMS normalization is important 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 [4].

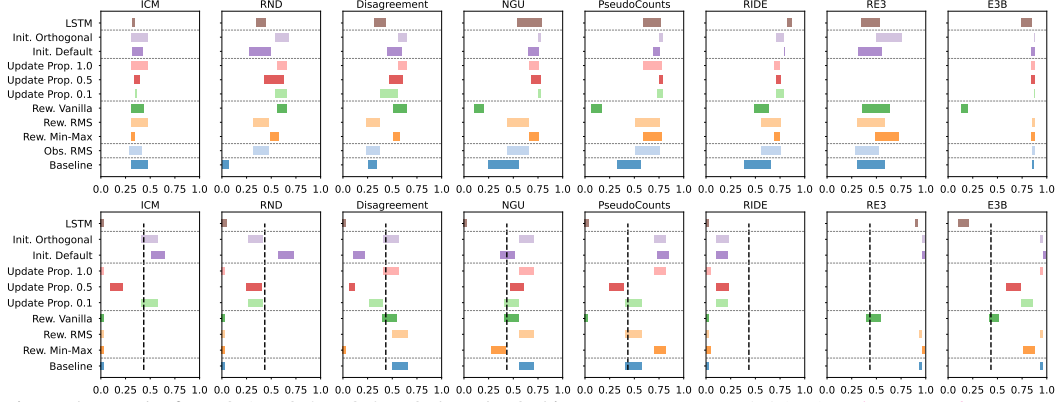


Figure 4: Results for RQ 1, RQ 2, RQ 3, RQ 4, and RQ 5 in *SuperMarioBros-1-1* (top) and *MiniGrid-DoorKey-16x16* (bottom). The x-axis represents the episode return normalized by maximum score possibly achieved in the task. The vertical dashed line represents the performance of the extrinsic agent, which only has access to the task rewards. Each bar represents the mean and standard error (i.e. standard deviation normalized by the number of seeds) of the normalized episode returns. Results are aggregated over 10 seeds, and each run uses 10M environment interactions.

RQ 2: The impact of reward normalization.

Similarly to RQ 1, reward normalization can have a large impact 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 for the reward outputs of the intrinsic reward modules: (1) Min-Max normalization, (2) using an RMS of the standard deviation, and (3) no reward normalization.

Reward normalization smooths the optimization process, which can be beneficial for stability but can lead to slower convergence [4]. Our findings show that almost all intrinsic rewards critically require some form of reward normalization, as agents fail to explore without normalized rewards. Importantly, the latter applies to all the environments that we experiment with. Additionally, while RMS is generally the default strategy for reward normalization, our results show that Min-Max normalization is a more robust option in *SuperMarioBros*, improving the performance and reducing the variance of the majority of the methods. Interestingly, in *MiniGrid-DoorKey16x16*, Min-Max normalization seems to decrease the performance of Disagreement and NGU, rendering this implementation detail worth tuning for specific environments.

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

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

Our findings indicate that the auxiliary networks generally perform robustly across a range of update frequencies from 10% to 100% in *SuperMarioBros*. However, `update_proportion=1.0` is generally the most robust setting in *MiniGrid-DoorKey16x16*. Importantly, for RND and ICM, having more delayed updates for the auxiliary networks (i.e. `update_proportion=0.1`) achieves the best

226 performance in *MiniGrid-DoorKey16x16*. The latter was also noted in the official implementations of
227 these algorithms [4, 18], where a value of `update_proportion=0.25` was used.

228 The results in *SuperMarioBros* indicate that the synchronization of training between intrinsic rewards
229 and RL agents does not need to be complete. To further explore the impact of the update frequency,
230 we conducted additional experiments with extreme values of `update_proportion=0.001` and
231 `update_proportion=10.0`, as shown in Figure 8. These results reveal that very low update
232 frequencies can lead to substantial performance drops if the auxiliary networks fall too far behind.
233 Thus, the optimal update proportion can significantly influence performance and depends on the
234 specific algorithm and environment used.

RQ 4: The impact of weight initialization.

235

236 Weight initialization plays a crucial role in optimizing deep neural networks, enabling faster conver-
237 gence. In RQ 4, we compare two approaches for weight initialization in the auxiliary networks **of the**
238 **intrinsic reward modules**: (1) orthogonal weight initialization and (2) uniform weight initialization
239 (PyTorch’s default). **Note that again, the policy and value networks remain unchanged.**

240 Our results highlight the importance of weight initialization in intrinsically-motivated RL. Specifically,
241 we found that orthogonal weight initialization is beneficial for most intrinsic rewards, regardless of
242 their specific optimization tasks (e.g., inverse dynamics, forward dynamics), and even in random
243 networks (e.g., RND and RE3). This benefit is evidenced by reduced variance in episode returns and
244 generally higher mean returns. This observation aligns with previous research indicating that orthog-
245 onal weight initialization can improve performance stability in deep RL agents [23, 24]. Importantly,
246 RND is the intrinsic reward method which shows the highest variability for this implementation
247 detail, where orthogonal weight initialization works better in *SuperMarioBros* but worse than uniform
248 initialization in *MiniGrid-DoorKey16x16*.

RQ 5: Is memory required to optimize intrinsic rewards?

249

250 In RQ 5, we investigate whether the intrinsic rewards included in RLeXplore benefit from memory-
251 enabled architectures. We compare the optimization of intrinsic rewards using a vanilla policy
252 network and one equipped with a long-short term memory (LSTM) [25] module, while keeping PPO
253 as the RL backbone algorithm.

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

259 4.2 Combination of Intrinsic and Extrinsic Rewards

RQ 6: Joint Optimization of Intrinsic and Extrinsic Rewards

260

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

For this analysis, we used the *Procgen-Maze* task [26] 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. We note that in our framework, we do not provide different context information to the agents for singleton versus contextual MDPs (e.g. the context ID). We refer to these frameworks to formalize the agent-environment interaction when the environment remains static throughout training (i.e. singleton - 1 maze) versus when it varies at each episode (i.e. contextual - a different maze at each episode).

Figure 5 demonstrates that learning two separate value functions [18], 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.

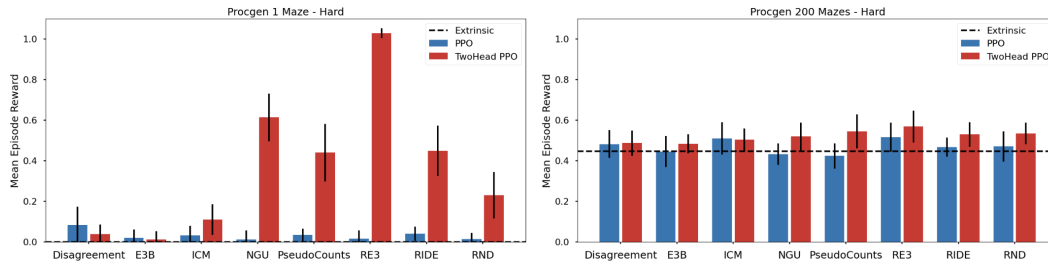


Figure 5: (Left) During training, the extrinsic agent struggles to find the goal in the selected Maze, resulting in a reward of 0. While some intrinsic reward methods yield occasional non-zero rewards, the algorithms perform significantly better when intrinsic and extrinsic value estimation are decoupled 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, 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: The performance of mixed intrinsic rewards.

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.

References

- [1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- [2] 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*, pages 1–17, 2019.
- [3] Minqi Jiang, Tim Rocktäschel, and Edward Grefenstette. General intelligence requires rethinking exploration. *Royal Society Open Science*, 10(6):230539, 2023.
- [4] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *Proceedings of the 7th International Conference on Learning Representations*, pages 1–17, 2019.
- [5] 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*, pages 16–17, 2017.
- [6] Roberta Raileanu and Tim Rocktäschel. Ride: Rewarding impact-driven exploration for procedurally-generated environments. In *International Conference on Learning Representations*, 2020.
- [7] 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 *International Conference on Learning Representations*, 2020.
- [8] Mikael Henaff, Roberta Raileanu, Minqi Jiang, and Tim Rocktäschel. Exploration via elliptical episodic bonuses. *Advances in Neural Information Processing Systems*, 35:37631–37646, 2022.
- [9] Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-supervised exploration via disagreement. In *International conference on machine learning*, pages 5062–5071. PMLR, 2019.
- [10] Misha Laskin, Denis Yarats, Hao Liu, Kimin Lee, Albert Zhan, Kevin Lu, Catherine Cang, Lerrel Pinto, and Pieter Abbeel. Urlb: Unsupervised reinforcement learning benchmark. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.
- [11] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.
- [12] 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.
- [13] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [14] 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.
- [15] 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*, pages 1407–1416. PMLR, 2018.

- [16] 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.
- [17] 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.
- [18] 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.
- [19] Mingqi Yuan, Zequn Zhang, Yang Xu, Shihao Luo, Bo Li, Xin Jin, and Wenjun Zeng. Rllite: Long-term evolution project of reinforcement learning. *arXiv preprint arXiv:2309.16382*, 2023.
- [20] 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*, pages 9443–9454, 2021.
- [21] 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.
- [22] Christian Kauten. Super Mario Bros for OpenAI Gym. GitHub, 2018.
- [23] Shengyi Huang, Rousslan Fernand Julien Dossa, Antonin Raffin, Anssi Kanervisto, and Weixun Wang. The 37 implementation details of proximal policy optimization. *The ICLR Blog Track 2023*, 2022.
- [24] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep policy gradients: A case study on ppo and trpo. *arXiv preprint arXiv:2005.12729*, 2020.
- [25] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [26] Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020.
- [27] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2018.
- [28] Michael Laskin, Hao Liu, Xue Bin Peng, Denis Yarats, Aravind Rajeswaran, and Pieter Abbeel. Cic: Contrastive intrinsic control for unsupervised skill discovery. In *Deep RL Workshop NeurIPS 2021*, 2021.
- [29] Víctor Campos, Alexander Trott, Caiming Xiong, Richard Socher, Xavier Giró-i Nieto, and Jordi Torres. Explore, discover and learn: Unsupervised discovery of state-covering skills. In *International Conference on Machine Learning*, pages 1317–1327. PMLR, 2020.
- [30] Zhaohan Guo, Shantanu Thakoor, Miruna Pîslar, Bernardo Avila Pires, Florent Altché, Corentin Tallec, Alaa Saade, Daniele Calandriello, Jean-Bastien Grill, Yunhao Tang, et al. Byol-explore: Exploration by bootstrapped prediction. *Advances in neural information processing systems*, 35:31855–31870, 2022.

- [31] Steven Kapturowski, Alaa Saade, Daniele Calandriello, Charles Blundell, Pablo Sprechmann, Leopoldo Sarra, Oliver Groth, Michal Valko, and Bilal Piot. Unlocking the power of representations in long-term novelty-based exploration. In *Second Agent Learning in Open-Endedness Workshop*. 400
- [32] 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. 401 402 403 404 405 406
- [33] 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. 407 408 409
- [34] 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. 410 411
- [35] 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. 412 413 414
- [36] 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, pages 5125–5133, 2020. 415 416 417
- [37] 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. 418 419 420
- [38] 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. 421 422 423
- [39] Sam Lobel, Akhil Bagaria, and George Konidaris. Flipping coins to estimate pseudocounts for exploration in reinforcement learning. *arXiv preprint arXiv:2306.03186*, 2023. 424 425
- [40] 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*, pages 2721–2730, 2017. 426 427 428
- [41] Jarryd Martin, Suraj Narayanan Sasikumar, Tom Everitt, and Marcus Hutter. Count-based exploration in feature space for reinforcement learning. In *IJCAI*, 2017. 429 430
- [42] Arthur Aubret, Laetitia Matignon, and Salima Hassas. An information-theoretic perspective on intrinsic motivation in reinforcement learning: A survey. *Entropy*, 25(2):327, 2023. 431 432
- [43] Bradley C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015. 433 434
- [44] Xingrui Yu, Yueming Lyu, and Ivor Tsang. Intrinsic reward driven imitation learning via generative model. In *Proceedings of the International Conference on Machine Learning*, pages 10925–10935, 2020. 435 436 437
- [45] 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. 438 439 440
- [46] 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. 441 442 443

- [47] Hao Liu and Pieter Abbeel. Aps: Active pretraining with successor features. In *International Conference on Machine Learning*, pages 6736–6747. PMLR, 2021.
- [48] Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Reinforcement learning with prototypical representations. In *International Conference on Machine Learning*, pages 11920–11931. PMLR, 2021.
- [49] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670, 2010.
- [50] Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.
- [51] Varsha Dani, Thomas P Hayes, and Sham M Kakade. Stochastic linear optimization under bandit feedback. In *COLT*, volume 2, page 3, 2008.
- [52] Tianjun Zhang, Huazhe Xu, Xiaolong Wang, Yi Wu, Kurt Keutzer, Joseph E Gonzalez, and Yuandong Tian. Bebold: Exploration beyond the boundary of explored regions. *arXiv preprint arXiv:2012.08621*, 2020.
- [53] Roger Creus Castanyer, Joshua Romoff, and Glen Berseth. Improving intrinsic exploration by creating stationary objectives. *arXiv preprint arXiv:2310.18144*, 2023.
- [54] 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.
- [55] 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*, pages 1861–1870. PMLR, 2018.

Checklist

1. For all authors...
 - (a) Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope? [\[Yes\]](#)
 - (b) Did you describe the limitations of your work? [\[Yes\]](#)
 - (c) Did you discuss any potential negative societal impacts of your work? [\[N/A\]](#) This work will not have a negative social impact.
 - (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [\[Yes\]](#)
2. If you are including theoretical results...
 - (a) Did you state the full set of assumptions of all theoretical results? [\[Yes\]](#)
 - (b) Did you include complete proofs of all theoretical results? [\[Yes\]](#)
3. If you ran experiments (e.g., for benchmarks)...
 - (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [\[Yes\]](#)
 - (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [\[Yes\]](#)
 - (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [\[Yes\]](#)
 - (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [\[Yes\]](#)

- 488 4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
- 489 (a) If your work uses existing assets, did you cite the creators? [\[Yes\]](#)
- 490 (b) Did you mention the license of the assets? [\[Yes\]](#)
- 491 (c) Did you include any new assets either in the supplemental material or as a URL? [\[Yes\]](#)
- 492 (d) Did you discuss whether and how consent was obtained from people whose data you're
- 493 using/curating? [\[Yes\]](#)
- 494 (e) Did you discuss whether the data you are using/curating contains personally identifiable
- 495 information or offensive content? [\[Yes\]](#)
- 496 5. If you used crowdsourcing or conducted research with human subjects...
- 497 (a) Did you include the full text of instructions given to participants and screenshots, if
- 498 applicable? [\[N/A\]](#)
- 499 (b) Did you describe any potential participant risks, with links to Institutional Review
- 500 Board (IRB) approvals, if applicable? [\[N/A\]](#)
- 501 (c) Did you include the estimated hourly wage paid to participants and the total amount
- 502 spent on participant compensation? [\[N/A\]](#)

A Limitations

In RLeXplore, we selected eight widely-used intrinsic reward methods that align with state-of-the-art exploration objectives. However, RLeXplore does not encompass the entire spectrum of exploration algorithms, as there is a vast array of exploration objectives with different characteristics that are not currently implemented in the framework.

For instance, skill-based algorithms, which typically involve separate phases for skill discovery and skill learning, are not included in RLeXplore. These algorithms, such as DIAYN [27], CIC [28], and EDL [29], have a fundamentally different structure and application compared to immediate intrinsic rewards, making them less suitable for integration into the current version of RLeXplore.

Additionally, RLeXplore was designed with accessibility in mind, ensuring that the implemented algorithms can be run on standard computational resources by any researcher. To maintain this accessibility, we have not included more complex and potentially powerful algorithms like BYOL-Explore [30] or RECODE [31]. These algorithms are not open-source and have been optimized exclusively with non-open-source RL algorithms, which further limits their integration into RLeXplore.

B Related Work

Intrinsic rewards often depend on heavily engineered implementations to stabilize their optimization [4]. This complexity can hinder reproducibility and lead to varying performance reports across different studies in the literature.

For example, even though RND is straightforward in concept, its effective implementation involves several crucial details: (1) observation normalization using running statistics with a warm-up period before training; (2) two-head value approximation to separately regress the streams of extrinsic and intrinsic rewards; (3) combining extrinsic and intrinsic advantages rather than rewards; and (4) propagating intrinsic rewards across episodes, using non-truncating value estimation when terminal signals are received from the environment. An RND implementation that omits any of these details risks producing inconsistent or misleading results, failing to reflect the true potential of the original algorithm. Similar "tricks" are necessary for many other intrinsic reward algorithms. In this work, we meticulously implement these details to ensure rigorous and reproducible results. Furthermore, we conduct experiments to identify which implementation details have the most significant impact on policy performance.

Some works have benchmarked intrinsic rewards in specific environments [32, 33, 10], 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.

B.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 [34, 35, 36, 37]. In finite state spaces, count-based methods perform near optimally [34]. 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 [38, 39]. Pseudo-counts provide a framework to generalize count-based methods to high-dimensional and partially observed environments [38, 40, 41]. [4] proposed random network distillation (RND), which uses the prediction error against a fixed network as a learning signal that is correlated to counts. Recently, [8] proposed E3B and showed that the intrinsic 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.

550 B.2 Curiosity-Driven Exploration

551 Curiosity-based objectives train agents to interact with the environment seeking to experience
552 outcomes that are not aligned with the agents’ predictions [42]. Hence, curiosity-driven exploration
553 usually involves training an agent to increase its knowledge about the environment (e.g., environment
554 dynamics) [43, 5, 44]. The intrinsic curiosity module (ICM) [5, 2] learns a joint embedding space
555 with inverse and forward dynamics losses and was the first curiosity-based method successfully
556 applied to deep RL settings. Disagreement [9] further extended ICM by using the variance over an
557 ensemble of forward-dynamics models to compute curiosity. However, curiosity-driven methods are
558 consistently found to be unsuccessful when the environment has irreducible noise [45]. To address the
559 problem, [6] proposed RIDE, which uses the difference between two consecutive state embeddings as
560 the intrinsic reward and encourages the agent to choose actions that result in significant state changes.
561 In general, curiosity-based objectives remain amongst the most popular intrinsic rewards in deep RL
562 applications to this day. In RLeXplore, we include ICM, Disagreement, and RIDE as representatives
563 of the state-of-the-art curiosity-driven methods.

564 B.3 Global and Episodic Exploration

565 Towards more general and adaptive agents, recent works have studied decision-making problems
566 in contextual Markov decision processes (MDPs) (e.g., procedurally-generated environments) [6,
567 8, 46]. Contextual MDPs require episodic-level exploration, where novelty estimates are reset at
568 the beginning of each episode. [21] showed that both global and episodic exploration modalities
569 have unique benefits and proposed combined objectives that achieve remarkable performance across
570 many MDPs of different structures. NGU [7] and RIDE [6] also instantiate both global and episodic
571 bonuses. Inspired by this recent line of works, in this paper, we study novel combinations of objectives
572 for exploration that achieve impressive results in contextual MDPs.

573 B.4 Unsupervised RL

574 Unsupervised Reinforcement Learning (URL) is a developing area of research focused on training
575 decision-making agents without relying on explicit supervision. This approach draws inspiration
576 from human learning, which often relies on intrinsic motivation. The goal of URL is to pre-train
577 agents in a way that allows them to quickly and effectively adapt to new tasks with minimal external
578 guidance.

579 A common approach in URL involves skill-based methods, where the process is divided into two
580 distinct phases: skill discovery and skill learning [47, 48, 29]. In this framework, agents first learn a
581 variety of skills through exploration and then use these skills to maximize performance on a given
582 task with an external reward function [27].

583 The URL benchmark (URLB) [10] provides implementations of eight different URL algorithms
584 and evaluates their performance using a modified version of the DeepMind Control Suite. However,
585 URLB has limitations: its implementations are not modular or easily integrated with other RL
586 libraries, which hinders its broader adoption in research.

587 To address this issue, we introduce RLeXplore. Unlike URLB, RLeXplore is designed to be highly
588 modular and easily integrable with existing RL libraries. This modularity allows researchers to seam-
589 lessly incorporate RLeXplore into their workflows. Additionally, RLeXplore focuses on immediate
590 intrinsic reward methods (i.e., non-skill-based approaches) that are straightforward to combine with
591 task rewards. These methods do not require explicit separation of the RL training into distinct phases.

592 The primary contribution of RLeXplore is not just its benchmark but its set of reliable, easy-to-use
593 implementations. This design facilitates research and experimentation by providing practical tools
594 rather than focusing solely on ranking algorithms across a specific set of tasks.

C Algorithmic Baselines

ICM [5]. 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 infers the current action \mathbf{a}_t based on the encoded states \mathbf{e}_t and \mathbf{e}_{t+1} , where $\mathbf{e} = \psi(\mathbf{s})$ and $\psi(\cdot)$ is an embedding network. Meanwhile, the forward model f predicts the encoded next-state \mathbf{e}_t based on $(\mathbf{e}_t, \mathbf{a}_t)$. Finally, the intrinsic reward is defined as

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

RND [4]. 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 \tilde{f} . As a result, the agent is motivated to explore unseen parts of the state space. The intrinsic reward is defined as

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

Disagreement [9]. 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

$$I_t = \text{Var}\{f_i(\mathbf{e}_t, \mathbf{a}_t)\}, i = 0, \dots, N \quad (5)$$

NGU [7]. 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

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

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 [7]. Pseudo-counts has been widely used in count-based exploration approaches [38, 40] with diverse implementations like neural density models. In this paper, we follow NGU [7] that computes pseudo-counts via k -nearest neighbor estimation, which is highly efficient and can be applied to arbitrary task. Given the encoded observations $\{\mathbf{e}_0, \dots, \mathbf{e}_{T-1}\}$ visited in the an episode, we have

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

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

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

RIDE [6]. 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

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

where $N_{\text{ep}}(\mathbf{s}_{t+1})$ 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 [20]. 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

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

E3B [8]. E3B provides a generalization of count-based rewards to continuous spaces. 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 [49, 50, 51] 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 [8]. Given a feature encoding f , at each time step t of the episode the elliptical bonus I_t is defined as follows:

$$I_t = f(s_t)^T C_{t-1} f(s_t) \quad (11)$$

$$C_{t-1} = \sum_{i=1}^{t-1} f(s_i) f(s_i)^T + \lambda I \quad (12)$$

where f is the learned representation mapping and C_{t-1} is the episodic ellipsoid [8].

D Experimental Settings

D.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) [13] implementation from RLLTE [19].

Table 3: PPO hyperparameters for *SuperMarioBros* and *Procgen* games. These remain fixed for all experiments.

Hyperparameter	SuperMarioBros	MiniGrid	Procgen
Observation downsampling	(84, 84)	(7, 7)	(64, 64)
Observation normalization	/ 255.	No	/ 255.
Reward normalization	No	No	No
Weight initialization	Orthogonal	Orthogonal	Orthogonal
LSTM	No	No	No
Stacked frames	No	No	No
Environment steps	10000000	10000000	25000000
Episode steps	128	32	256
Number of workers	1	1	1
Environments per worker	8	256	64
Optimizer	Adam	Adam	Adam
Learning rate	2.5e-4	2.5e-4	5e-4
GAE coefficient	0.95	0.95	0.95
Action entropy coefficient	0.01	0.01	0.01
Value loss coefficient	0.5	0.5	0.5
Value clip range	0.1	0.1	0.2
Max gradient norm	0.5	0.5	0.5
Epochs per rollout	4	4	3
Batch size	256	1024	2048
Discount factor	0.99	0.99	0.999

D.2 Details of RQs

Table 4 illustrates the details of the candidates for all RQs. For RQ 1-5, 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 RQ 1 only change the technique of observation normalization, and RQ 2 will use the best observation normalization method for each reward module obtained in RQ 1. Likewise, RQ 3 will follow the best results obtained in RQ 1-2 and only change the proportion of samples used for model update. However, we kept using the baselines settings for each reward in RQ 8 to explore the most original performance of the mixed intrinsic rewards.

Table 4: Details of candidates for all RQs, where \mathbf{I} is a batch of intrinsic rewards.

#	Candidate	Detail
RQ 1	Min-Max	$\text{image} = \text{image} / 255.0$
	RMS	$\text{image} = \text{Clip} \left(\frac{\text{image} - \text{running mean}}{\text{running std.}}, -5.0, 5.0 \right)$
RQ 2	Vanilla	$\mathbf{I} = \mathbf{I}$
	RMS	$\mathbf{I} = \frac{\mathbf{I}}{\text{running std}}$
	Min-Max	$\mathbf{I} = \frac{\mathbf{I} - \min(\mathbf{I})}{\max(\mathbf{I}) - \min(\mathbf{I})}$
RQ 3	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.
RQ 4	Vanilla	Fill the input tensor with values drawn from the uniform distribution.
	Orthogonal	Fill the input tensor with a (semi) orthogonal matrix.
RQ 5	Vanilla	Policy network with only convolutional and linear layers.
	LSTM	Policy network that includes an LSTM layer.
RQ 6	Vanilla	$R = E + I$
	Two-head	Value network uses two separate branches for E and I .
RQ 7	N/A	N/A
RQ 8	Global+Episodic	E3B+RND, E3B+ICM, E3B+RIDE, RE3+RND, RE3+ICM, RE3+RIDE
	Global+Global	RND+ICM, RND+RIDE, ICM+RIDE

648 D.3 Best Configurations

Table 5: The best configurations for each intrinsic reward on *SuperMarioBros-1-1-v3* games.

Reward	Obs. Norm.	Reward Norm.	Update Prop.	Weight Init.	Memory Required
PseudoCounts	✓	Min-Max	0.5	Orthogonal	✗
ICM	✗	RMS	1.0	Orthogonal	✗
RND	✓	Vanilla	1.0	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	✓	Vanilla	1.0	Orthogonal	✗

E Usage Examples

E.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 be 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, rlds, terms, trunks, infos = envs.step(actions)
        # get data from the transitions
        irs.watch(obs, actions, rlds, next_obs, terms, trunks, 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 be 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, rlds, terms, trunks, infos = envs.step(actions)
    # get data from the transitions
    irs.watch(obs, actions, rlds, next_obs, terms, trunks, 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,
```

```

710         rewards=rwds, terminateds=terms,
711         truncateds=terms, next_observations=next_obs),
712         sync=False)
713     ...
714     # update the reward module
715     batch = replay_storage.sample()
716     irs.update(samples=dict(observations=batch.obs,
717                             actions=batch.actions,
718                             rewards=batch.rewards,
719                             terminateds=batch.terminateds,
720                             truncateds=batch.truncateds,
721                             next_observations=batch.next_obs)
722     )
723     ...
724 
```

725 E.2 RLeXplore with Stable-Baselines3

726 Stable-Baselines3 (SB3) [17] is one of the most successful and popular RL frameworks that provides
727 a set of reliable implementations of RL algorithms in Python. SB3 provides a convenient callback
728 function that can be called at given stages of the training procedure, the following code example
729 demonstrates how to use RLeXplore in SB3 for on-policy RL algorithms:

```

730 class RLeXploreWithOnPolicyRL(BaseCallback):
731     """
732     Combining RLeXplore and on-policy algorithms from SB3.
733     """
734     def __init__(self, irs, verbose=0):
735         super(RLeXploreWithOnPolicyRL, self).__init__(verbose)
736         self.irs = irs
737         self.buffer = None
738
739     def init_callback(self, model: BaseAlgorithm) -> None:
740         super().init_callback(model)
741         self.buffer = self.model.rollout_buffer
742
743     def _on_step(self) -> bool:
744         """
745         This method will be called by the model after each call to `
746         env.step()`.
747
748         :return: (bool) If the callback returns False, training is
749                 aborted early.
750         """
751         observations = self.locals["obs_tensor"]
752         device = observations.device
753         actions = th.as_tensor(self.locals["actions"], device=device)
754         rewards = th.as_tensor(self.locals["rewards"], device=device)
755         dones = th.as_tensor(self.locals["dones"], device=device)
756         next_observations = th.as_tensor(self.locals["new_obs"],
757                                         device=device)
758
759         # get data from the transitions
760         self.irs.watch(observations, actions, rewards, dones, dones,
761                       next_observations)
762
763         return True
764
765     def _on_rollout_end(self) -> None:
766         # prepare the data samples
767         obs = th.as_tensor(self.buffer.observations)
768         # get the new observations
769         new_obs = obs.clone()
770         new_obs[:-1] = obs[1:]
771 
```

```

772     new_obs[-1] = th.as_tensor(self.locals["new_obs"])
773     actions = th.as_tensor(self.buffer.actions)
774     rewards = th.as_tensor(self.buffer.rewards)
775     dones = th.as_tensor(self.buffer.episode_starts)
776     print(obs.shape, actions.shape, rewards.shape, dones.shape,
777           obs.shape)
778     # compute the intrinsic rewards
779     intrinsic_rewards = irs.compute(
780         samples=dict(observations=obs, actions=actions,
781                     rewards=rewards, terminateds=dones,
782                     truncateds=dones, next_observations=new_obs),
783

```

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

785 E.3 RLeXplore with CleanRL

786 CleanRL [18] is an open-source project focused on implementing RL algorithms with clean, under-
787 standable, and reproducible code. It aims to make RL more accessible by providing implementations
788 that are simpler and more transparent than those typically found in research papers or larger libraries.
789 The following code example demonstrates how to use RLeXplore in CleanRL for on-policy RL
790 algorithms:

```

791 # load the library
792 from rllte.xplore.reward import RE3
793 # create the reward module
794 irs = RE3(envs=envs, device=device)
795 ...
796 # get data from the transitions
797 irs.watch(observations=obs[step], actions=actions[step],
798           rewards=rewards[step], terminateds=dones[step],
799           truncateds=dones[step], next_observations=next_obs
800           )
801 ...
802 next_obs = obs.clone()
803 next_obs[:-1] = obs[1:]
804 next_obs[-1] = next_obs
805 # compute the intrinsic rewards
806 intrinsic_rewards = irs.compute(
807     samples=dict(observations=obs, actions=actions,
808                 rewards=rewards, terminateds=dones,
809                 truncateds=dones, next_observations=next_obs),
810     sync=True)
811 # add the intrinsic rewards to the rewards
812 rewards += intrinsic_rewards
813

```

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

F Comparative Analysis of Intrinsic Reward Implementations

This section provides a detailed comparative analysis of our intrinsic reward implementations in the RLeXplore framework against other publicly available implementations. The results are compiled in tables for different environments to demonstrate the performance of each algorithm. We cite the works from which we obtain the original results in each of the tables and we provide our results by averaging the performance of the last 100 training episodes over 3 seeds.

F.1 SuperMarioBros without Task Rewards

Table 6: Comparison of % of level completed in SuperMarioBros without task rewards.

Algorithm	% of Level Completed (10M Steps)	% of Level Completed (1M Steps)
(Original) RIDE [6]	-	23%
(Original) ICM [5]	30%	-
(RLeXplore) RIDE	100%	50%
(RLeXplore) ICM	30%	2%

The percentage of the level completed is computed dividing the episode return by 3,000, which corresponds as the maximum reward that can be obtained in *SuperMarioBros-1-1* (if the agent solves the level without wasting time). Note that in Figure 1 we divide this quantity by 100 and show a maximum reward of 30.

Note that our implementation of ICM reproduces the results reported in the original paper in Mario [5] and our implementation of RIDE further outperforms the original implementation.

F.2 MiniGrid-DoorKey-16x16 (Extrinsic + Intrinsic Rewards)

Table 7: Episode returns in MiniGrid-DoorKey-16x16 with extrinsic and intrinsic rewards.

Algorithm	Episode Return (10M Steps)
(Original) RIDE [52]	0.25
(Original) ICM [52]	0.0
(Original) RND [52]	0.0
(Original) IMPALA [52]	0.0
(RLeXplore) PPO	0.37
(RLeXplore) ICM	0.6
(RLeXplore) RND	0.6
(RLeXplore) RIDE	0.12

Using the implementations in RLeXplore we obtain significantly better performance in the same tasks and with the same algorithms.

F.3 MiniGrid-DoorKey-8x8 (1M Environment Steps)

We also evaluate our implementations in *MiniGrid-DoorKey-8x8* with a budget of 1M environment steps to be able to compare to the original results reported in [20].

Table 8: Episode returns in MiniGrid-DoorKey-8x8 with 1M environment steps.

Algorithm	Episode Return (1M Steps)
(Original) RE3 [20]	0.5
(Original) RND [20]	0.0
(Original) ICM [20]	0.2
(Original) A2C [20]	0.0
(RLeXplore) RE3	0.95
(RLeXplore) RND	0.0
(RLeXplore) ICM	0.83
(RLeXplore) PPO	0.22

Importantly, we reproduce the results reported in [20] very accurately, showing that RE3 can provide more sample-efficient exploration in this domain, compared to RND and ICM. Still, our implementations of RE3 and ICM achieve even better performance than the original ones.

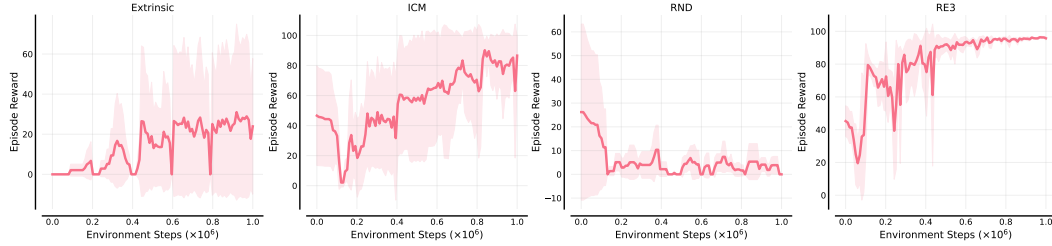


Figure 7: Using RLeXplore in *MiniGrid-DoorKey-8x8*, we are able to not only reproduce the conclusions obtained in previous work [20] regarding the capabilities of RE3 compared to ICM and RND, but we also generally achieve better performance, hence providing stronger baselines to the RL community.

F.4 Procgen - 200 Mazes (25M Training Steps)

Table 9: Performance comparison in Procgen - 200 Mazes with 25M training steps.

Algorithm	Procgen - 200 Mazes (25M Steps)
(Original) E3B [53]	3.0
(Original) ICM [53]	2.5
(Original) RND [53]	1.7
(RLeXplore) E3B	4.1
(RLeXplore) ICM	5.9
(RLeXplore) RND	5.0

F.5 ALE-5 (25M Training Steps)

In this section, we present the evaluation results of the intrinsic reward algorithms on a set of ALE games known for their challenging exploration requirements. These "hard-exploration" games, including Gravitar, Montezuma's Revenge, Private Eye, Seaquest, and Venture, serve as a benchmark for testing the effectiveness of intrinsic rewards in aiding exploration and improving agent performance.

We observe that while intrinsic rewards lead to a decline in performance in Gravitar, they generally provide substantial benefits, particularly in environments where exploration is difficult. For example, in Seaquest, the use of intrinsic rewards enables algorithms to significantly outperform the extrinsic agent, which ranks among the lowest.

Table 10: Mean performance across different environments for each algorithm, averaged over 3 seeds after 25M environment steps. Results are averaged over the last 100 episodes of training. In Gravitar, intrinsic rewards appear to hinder the performance of the extrinsic agent, whereas in the other environments, they significantly enhance performance. Notably, in Seaquest, the extrinsic agent ranks among the lowest, highlighting the benefit of intrinsic rewards. All experiments were conducted using sticky actions with a repeat probability of 0.25.

Algorithm	Gravitar	MontezumaRevenge	PrivateEye	Seaquest	Venture
Extrinsic	1060.19	42.83	88.37	942.37	391.73
Disagreement	689.12	0.00	33.23	6577.03	468.43
E3B	503.43	0.50	66.23	8690.65	0.80
ICM	194.71	31.14	-27.50	2626.13	0.54
PseudoCounts	295.49	0.00	1076.74	668.96	1.03
RE3	130.00	2.68	312.72	864.60	0.06
RIDE	452.53	0.00	-1.40	1024.39	404.81
RND	835.57	160.22	45.85	5989.06	544.73

Note that we do not compare these results to other works because evaluation settings differ significantly between papers. For instance, in our case, we used sticky actions with a probability of 0.25%, which makes the exploration problem more difficult and it is not always used. Also, we trained our agents for 25M steps instead of the standard 200M due to computational constraints. Still, our results provide evidence that intrinsic rewards are generally helpful to achieve better episode returns in hard-exploration environments.

F.6 Comparison with Other Projects

Table 11: Details on official implementations of the included intrinsic rewards. **Decoupled**: Did the code decouple the intrinsic reward modules from the RL optimization algorithms, which can be directly reused in other projects?

Reward	Official Repository	ML framework	Backbone RL algorithm	Supported Tasks	Decoupled
ICM	Repository	Tensorflow	A3C	SuperMarioBros, VizDoom	✗
RND	Repository	Tensorflow	PPO	ALE	✗
Disagreement	Repository	Tensorflow	PPO	SuperMarioBros, ALE, Maze	✗
NGU	N/A	N/A	N/A	N/A	N/A
PseudoCounts	from NGU	N/A	N/A	N/A	N/A
RIDE	Repository	PyTorch	IMPALA	MiniGrid	✗
RE3	Repository	PyTorch	A2C, Dreamer, RAD	DMControl, MiniGrid	✗
E3B	Repository	PyTorch	IMPALA	MiniHack, VizDoom	✗

Table 11 illustrates the details on official implementations of the included intrinsic rewards in RLeXplore. It is natural to find that they are implemented (1) in different codebases with (2) different libraries (e.g. PyTorch vs Tensorflow), (3) using different RL algorithms (PPO, IMPALA, A3C, A2C), and (4) supporting different environments (ALE, Mario, MiniGrid, DMC). These details further motivate the development of a unified framework for training RL agents with intrinsic rewards under standardized conditions and reinforce our motivation to develop RLeXplore.

Furthermore, we provide a comparison of the advantages of other popular codebases for training RL agents with intrinsic rewards in terms of the number of intrinsic reward algorithms implemented, their modularity and ability to reuse components between RL libraries easily, their documentation, and the number of experiments provided. As compared to other existing projects, RLeXplore offers a distinctive advantage by providing a more unified and standardized approach to training RL agents with intrinsic rewards. It allows users to easily swap intrinsic reward modules regardless of RL libraries, which promotes reproducibility and consistency across different research works. Finally,

868 RLeXplore is evaluated on a wide range of benchmarks with over 1,000 experiments, ensuring its
 869 reliability and robustness across various scenarios.

Table 12: Comparison between RLeXplore and other reported libraries of intrinsic rewards. Note hat we focus on the intrinsic reward methods implemented. For instance, CleanRL has many implementations of different RL algorithms, but RND is the only supported intrinsic reward.

Framework	ML Framework	Number of Algorithms	Plug & Play	Documentation	Benchmark Results
CleanRL	PyTorch	1	✗	✓	1 task, 1 experiments
DI-Engine	PyTorch	3	✗	✓	5 tasks, 19 experiments
rllib	TensorFlow	2	✗	✓	N/A
RLeXplore	PyTorch	8	✓	✓	9 tasks, over 1000 experiments

870 G Ablation of the Update Proportion

871 In this section, we study the effects of extreme update proportion values on performance in the game
 872 Super Mario Bros. Our earlier experiments showed minimal performance differences when varying
 873 the update proportion between 10% and 100% in this environment. To draw more robust conclusions,
 874 we explore how the frequency of gradient updates for auxiliary networks impacts overall performance
 875 in detail by using more extreme values.

876 Our results indicate that insufficiently trained auxiliary networks (e.g., `update_proportion=0.001`)
 877 significantly degrade performance. This suggests that auxiliary networks must be reasonably aligned
 878 with the policy updates to maintain effective performance. While increasing the update frequency
 879 can benefit some algorithms, the marginal gains do not justify the additional computational cost in
 880 most cases.

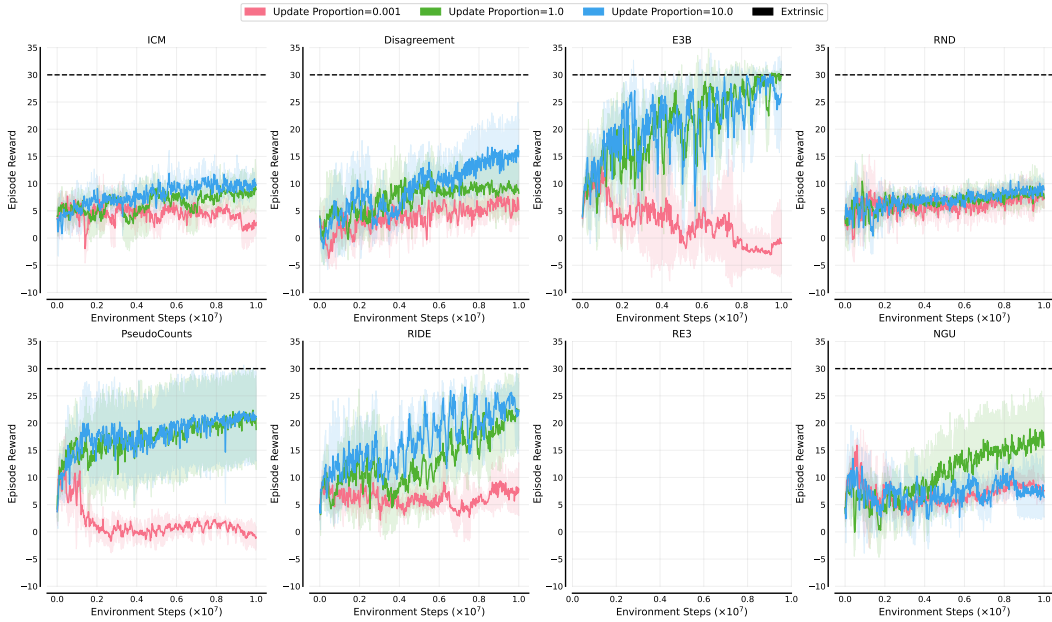


Figure 8: Performance impact of different update proportions on SuperMarioBros-1-1-v3. The learning curves show the average performance (solid line) and variability (shaded regions) across five random seeds. The results highlight the effect of extreme update frequencies on the efficacy of auxiliary networks and overall exploration performance.

881 It’s important to highlight that the experiments depicted in Figure 8 were conducted under
 882 different conditions than those in Figure 4. In the earlier experiments (Figure 4), we used
 883 the optimal configuration for each algorithm, determined through incremental ablations of the
 884 RQs. In contrast, for the results in Figure 8, we standardized the conditions by setting both
 885 `observation_normalization=RMS` and `reward_normalization=RMS`. This approach allowed
 886 us to better isolate and assess the impact of the update proportion variable, which accounts for the
 887 differences in variability compared to Figure 4.

888 H Learning Curves

889 H.1 RQ1

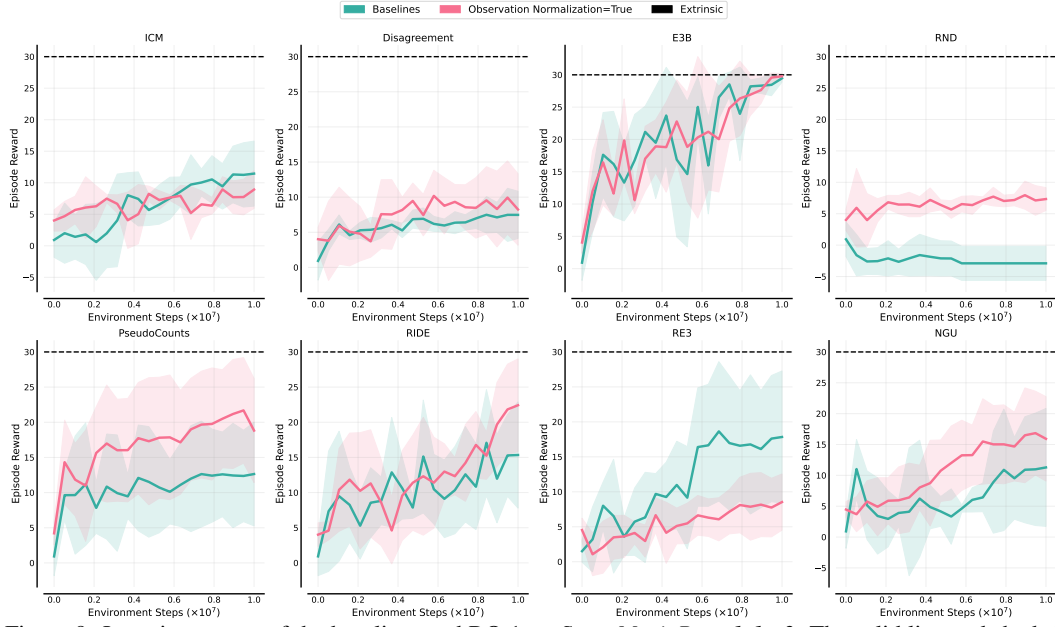


Figure 9: Learning curves of the baselines and RQ 1 on *SuperMarioBros-1-1-v3*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

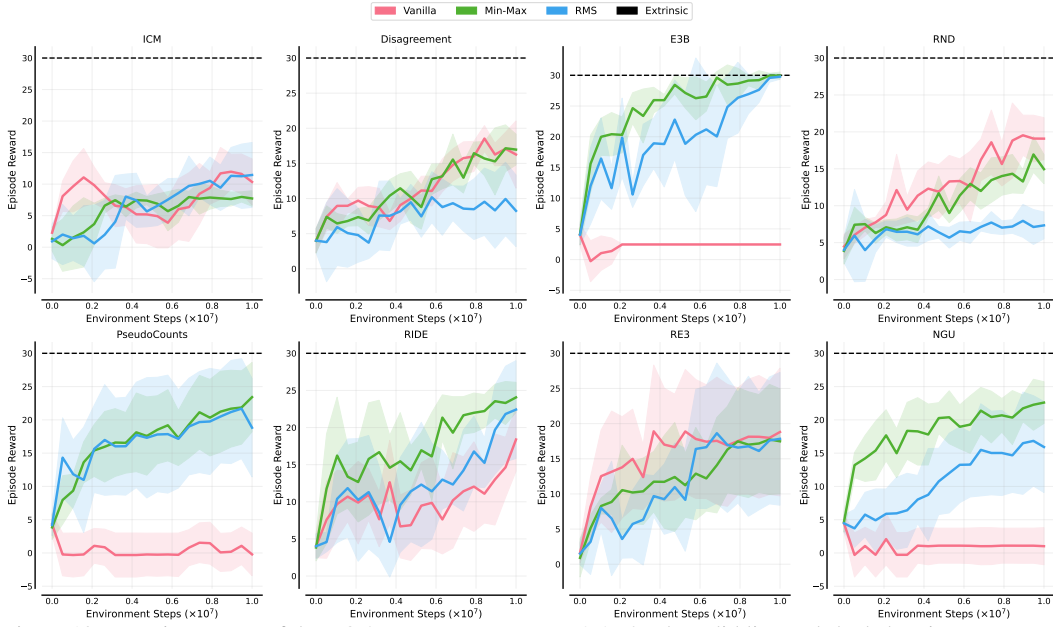


Figure 10: Learning curves of the RQ 2 on *SuperMarioBros-I-v3*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

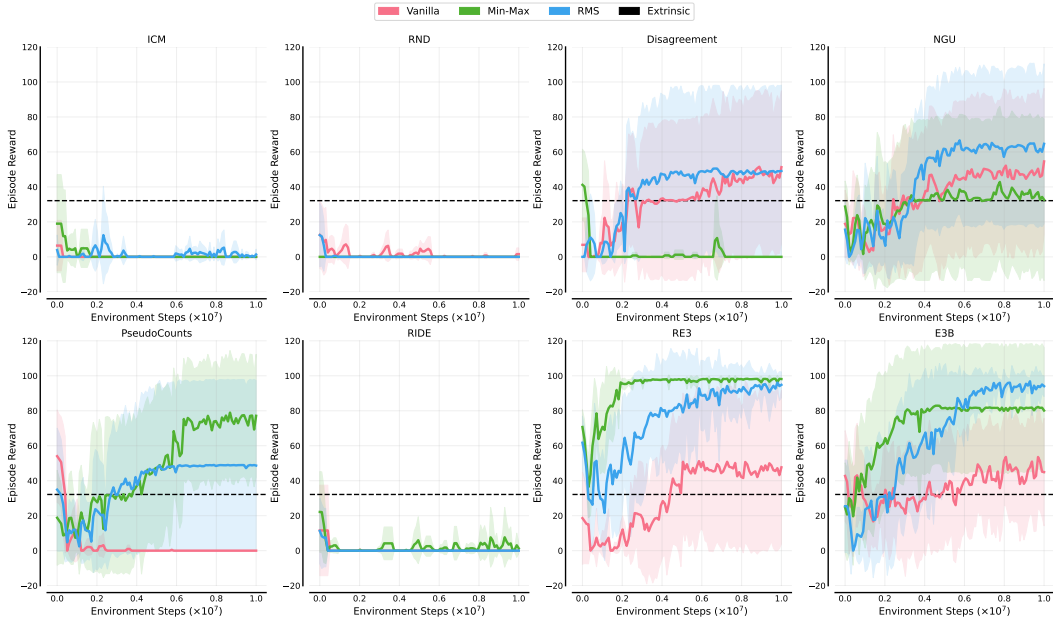


Figure 11: Learning curves of the RQ 2 on *MiniGrid-DoorKey-16x16-v0*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

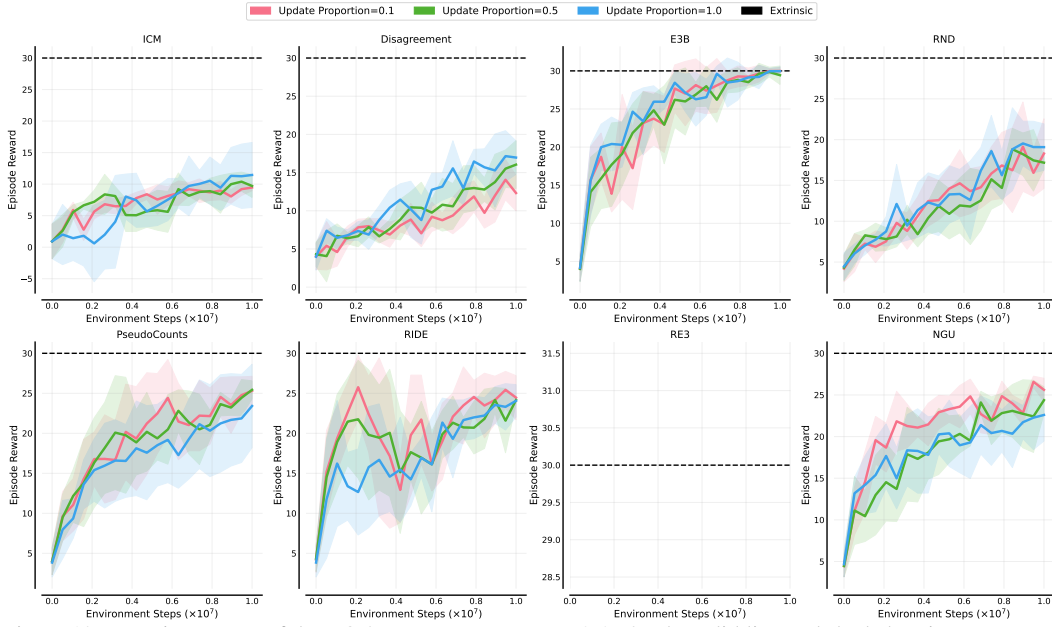


Figure 12: Learning curves of the RQ 3 on *SuperMarioBros-I-v3*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

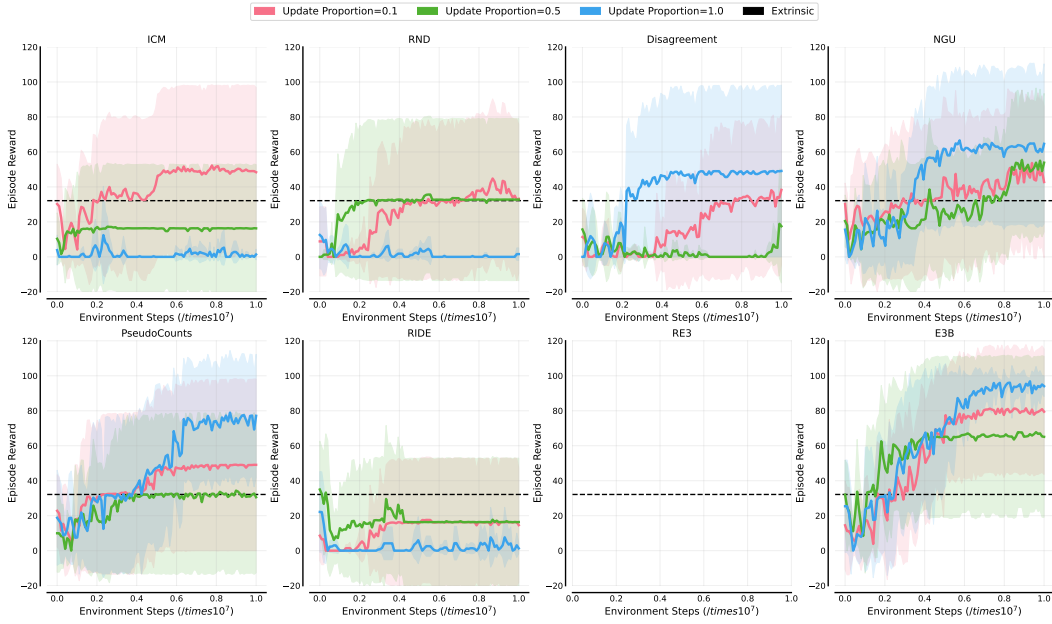


Figure 13: Learning curves of the RQ 3 on *MiniGrid-DoorKey-16x16-v0*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

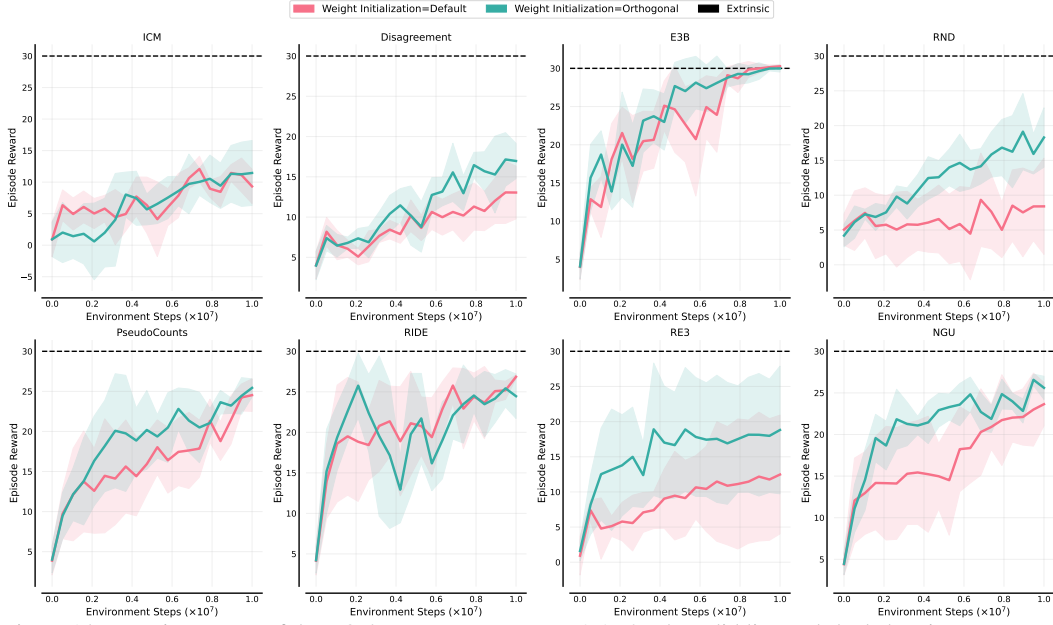


Figure 14: Learning curves of the RQ 4 on *SuperMarioBros-I-v3*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

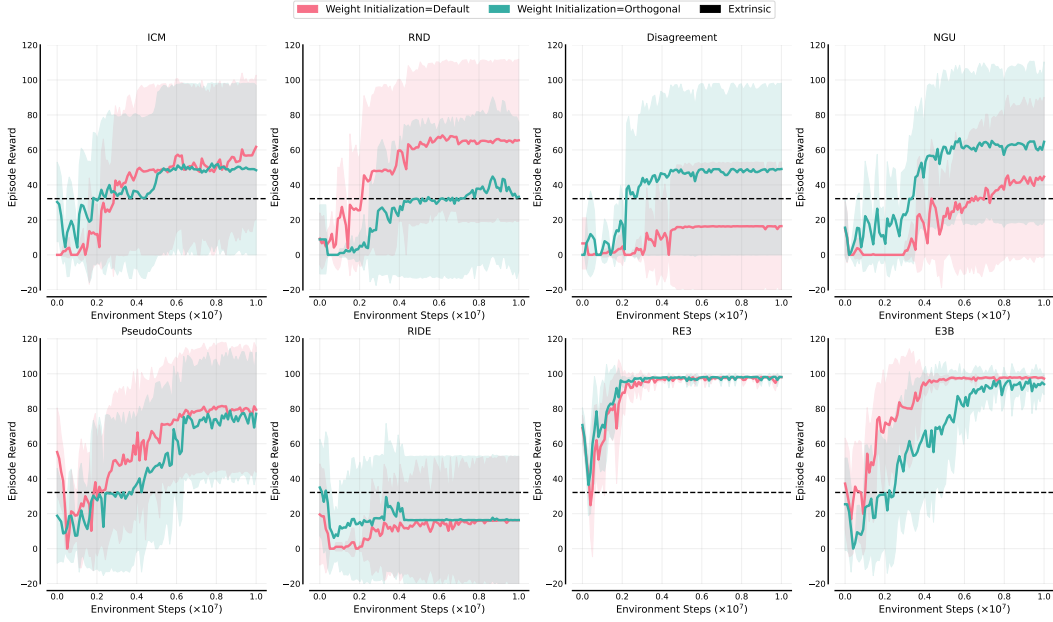


Figure 15: Learning curves of the RQ 4 on *MiniGrid-DoorKey-16x16-v0*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

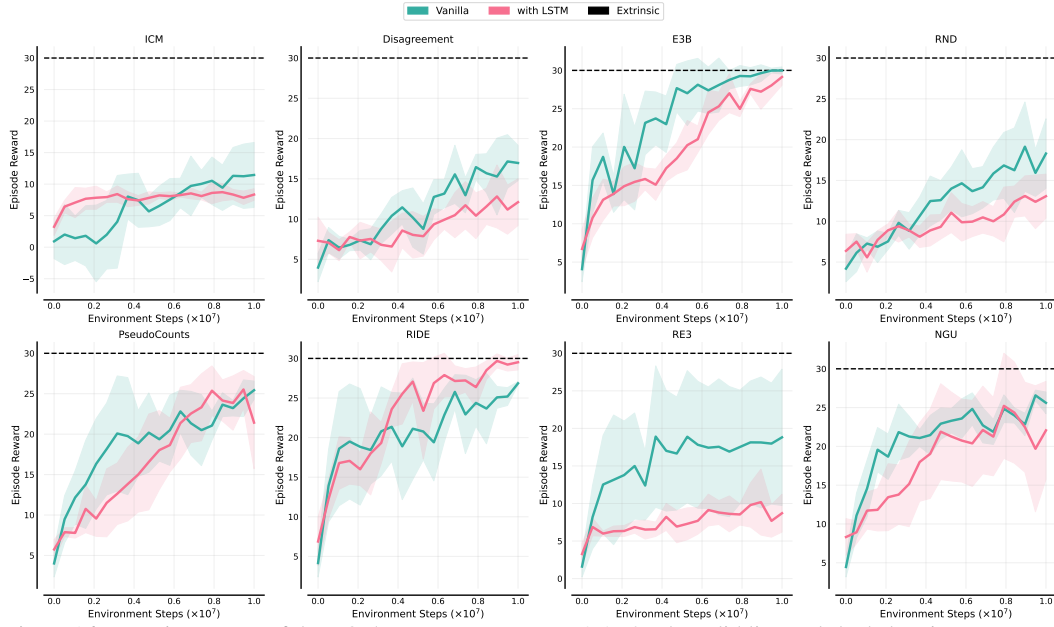


Figure 16: Learning curves of the RQ 5 on *SuperMarioBros-I-v3*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

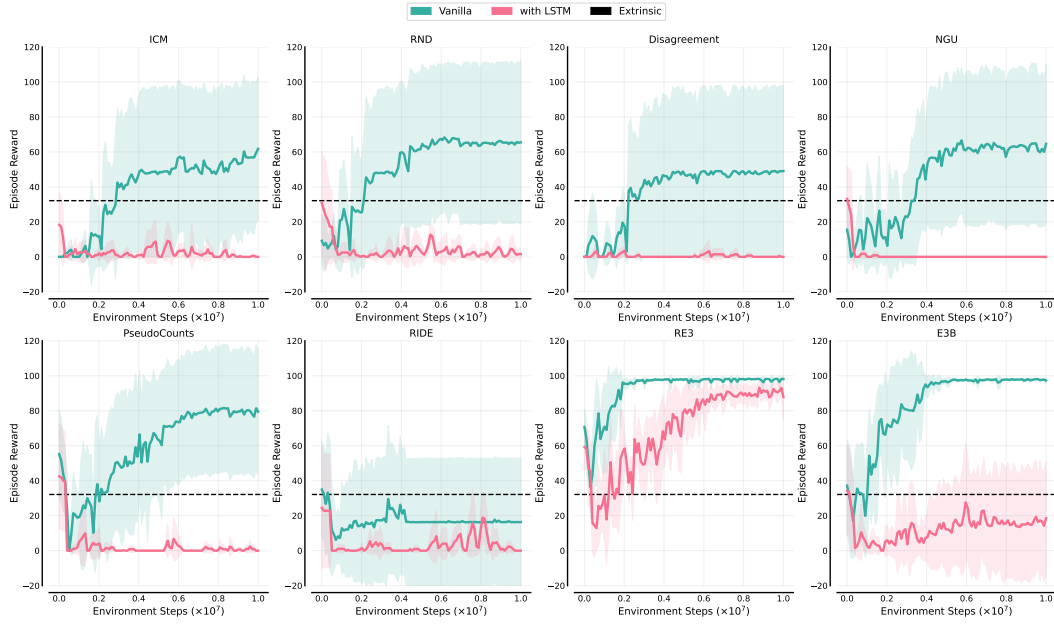


Figure 17: Learning curves of the RQ 5 on *MiniGrid-DoorKey-16x16-v0*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

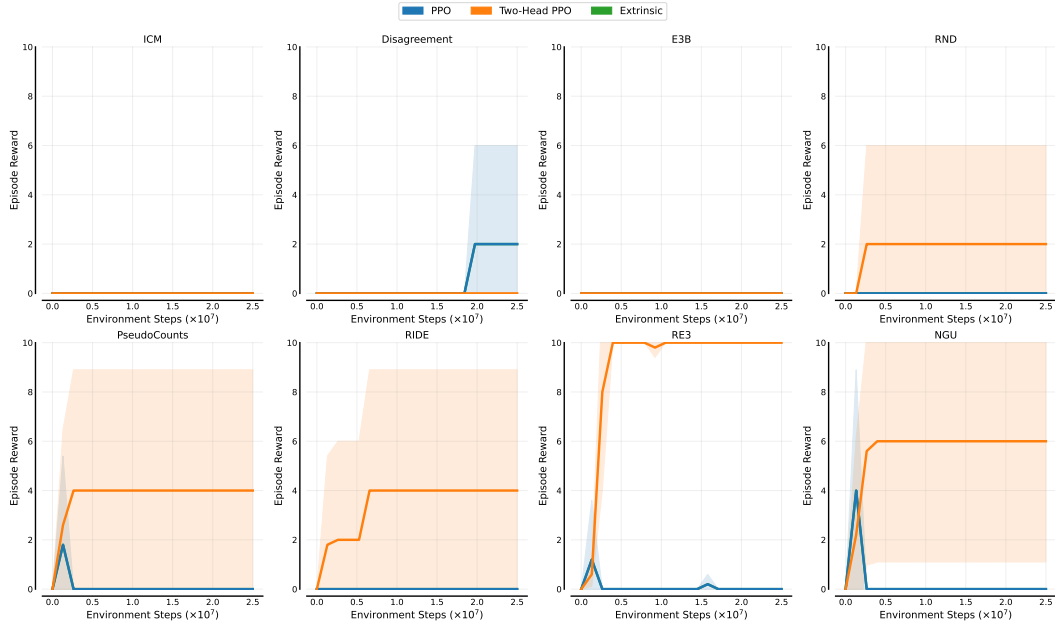


Figure 18: Learning curves of RQ 6 on *Procgen-IMazeHard*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

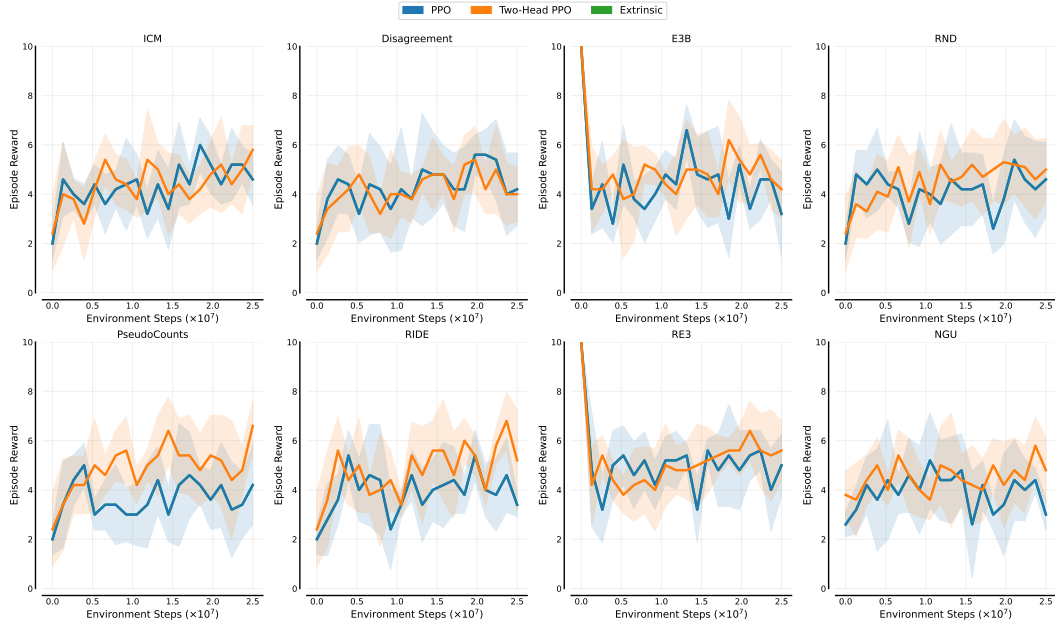


Figure 19: Learning curves of RQ 6 on *Procgen-AllMazeHard*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

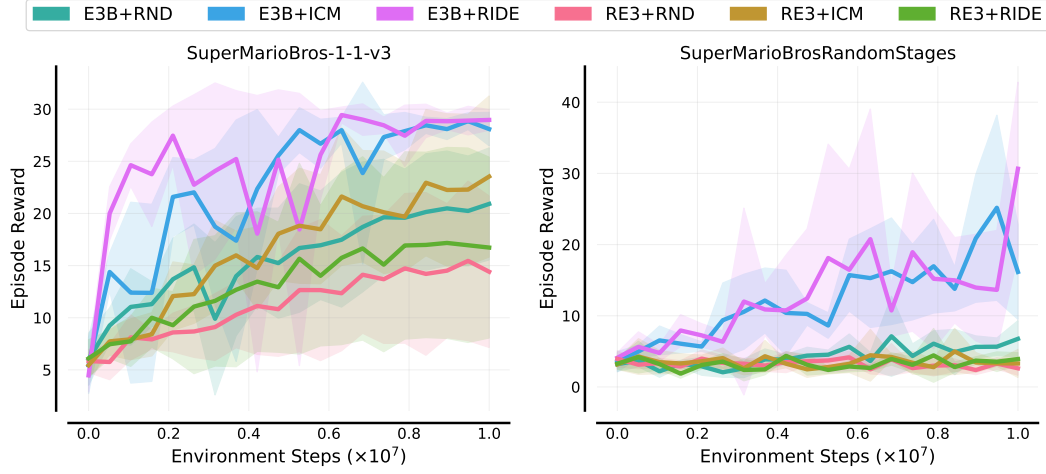


Figure 20: Learning curves of RQ 8 (global+episodic exploration) on *SuperMarioBros-1-1-v3* and *SuperMarioBrosRandomStages-v3*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

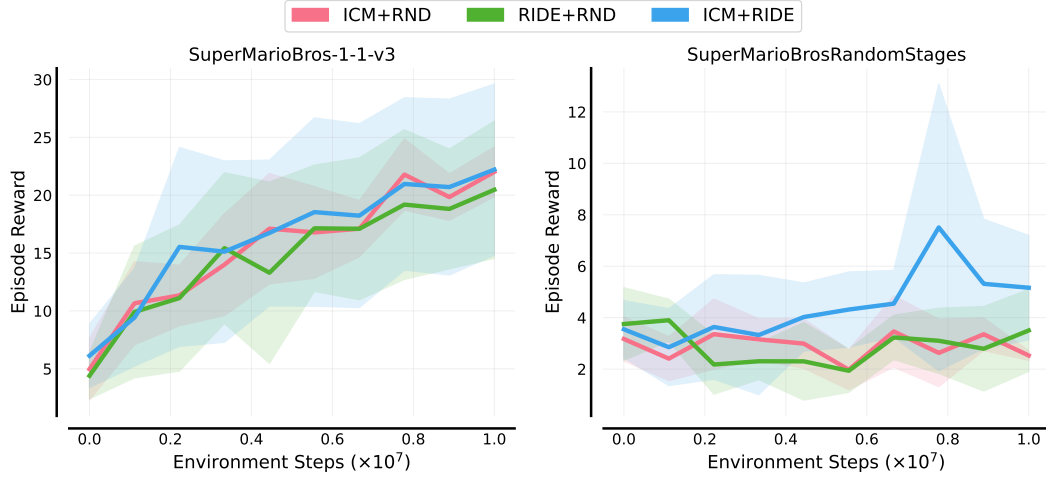


Figure 21: Learning curves of RQ 8 (global+global exploration) on *SuperMarioBros-1-1-v3* and *SuperMarioBrosRandomStages-v3*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

896 I A study of intrinsic rewards in Contextual MDPs

897 In contextual MDPs, there is little shared structure across episodes, since the episodic context can
 898 vary the environment significantly. In this settings, global intrinsic rewards, which re-use experi-
 899 ence from past episodes to compute novelty in the current episode can provide wrong estimations.
 900 Conversely, episodic intrinsic rewards, such as E3B and PseudoCounts, are specifically designed
 901 to estimate novelty within each new episode, aligning better with the dynamic nature of contextual
 902 MDPs. To instantiate the contextual MDP settings, we train the intrinsic reward methods using the
 903 *SuperMarioBrosRandomStages-v3* variant, where at each episode a different level of the game is
 904 sampled uniformly. Within this settings, the RL agents need to learn to explore differently at each
 905 episode, and it has been shown that global bonuses can fail to generalize their exploration capabilities
 906 [21].

907 As shown in Figure 22, E3B achieves the highest performance among all the intrinsic rewards, while
 908 other intrinsic rewards struggle to adapt and nearly fail to learn. These results provide further evidence
 909 about the limitations of global intrinsic rewards described in [21]. Together with our results shown
 910 in Figure 1, we show that almost all the methods successfully explore Level 1 of *SuperMarioBros*
 911 without task rewards but struggle to make progress when trained using all the levels simultaneously.
 912 Figure 22 shows that E3B is the best-performing algorithm in this specific contextual MDP. The latter
 913 makes sense as E3B is purposely designed to handle the contextual exploration settings [8]. This
 914 distinct advantage underscores the importance of designing intrinsic rewards that are context-sensitive
 915 and capable of updating their novelty detection mechanisms based on the specific characteristics
 916 of each episode. Crucially, as shown in RQ7, using mixed objectives (i.e. particularly combining
 917 E3B with other objectives) yields even better performance in *SuperMarioBrosRandomStages-v3*,
 918 demonstrating that the advantages of global and episodic rewards can be smoothly combined and
 919 optimized together.

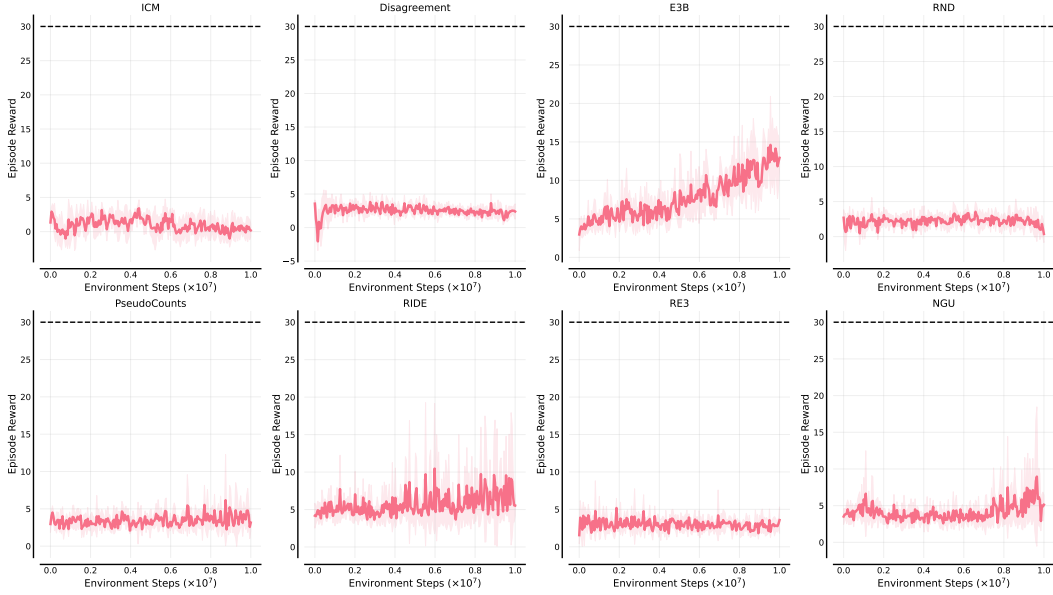


Figure 22: Learning curves of RQ 7 on *SuperMarioBrosRandomStages-v3*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

920 J On-Policy RL Algorithms and Discrete Control Tasks

921 In this section, we demonstrate the combination of RLeXplore and on-policy RL algorithms and
 922 its effectiveness on discrete control tasks. Specifically, we couple the PPO algorithm and intrinsic
 923 rewards, and evaluate their performance on *Montezuma Revenge*, a hard exploration task from the
 924 ALE benchmark [54]. We use the PPO implementation of CleanRL [18] to show the adaptability of
 925 RLeXplore. Table 13 illustrates the training hyperparameters used for the experiments.

Table 13: Training hyperparameters for *Montezuma Revenge*.

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

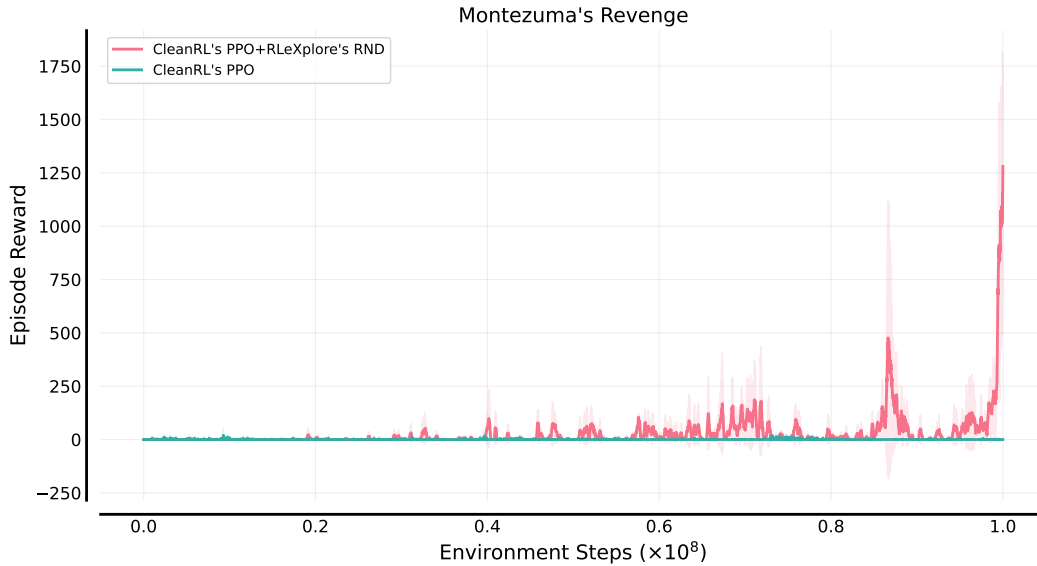


Figure 23: Since only RND can achieve significant results in this task among the eight intrinsic rewards, we only show the results of RND. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

926 K Off-Policy RL Algorithms and Continuous Control Tasks

927 To showcase the generality of RLeXplore, we run additional experiments in settings different from
 928 the ones in the main paper. Concretely, we couple intrinsic rewards with soft actor-critic (SAC) [55],
 929 an off-policy RL algorithm, and test their performance in *Ant-UMaze*, a continuous control task with
 930 sparse rewards. Table 14 illustrates the training hyperparameters used for the experiments. We show
 931 the performance of Disagreement, RND, ICM and vanilla SAC in Figure 24. The results indicate that
 932 intrinsically-motivated agents are able to navigate the maze more efficiently, finding the goals more
 933 often than the vanilla agents that can only learn from the sparse task rewards.

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

Table 14: Training hyperparameters for *Ant-UMaze*.

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

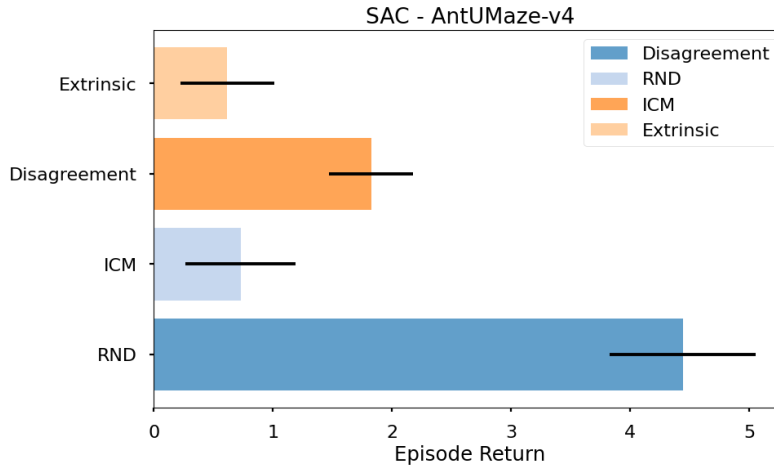


Figure 24: Performance comparison between the three selected intrinsic rewards and the extrinsic reward.

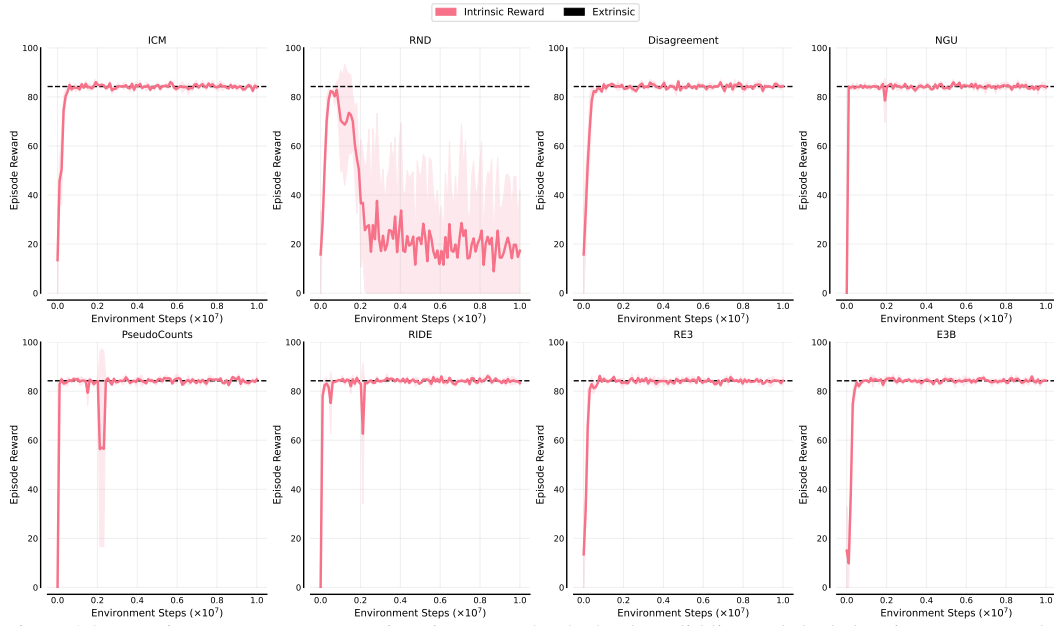


Figure 25: Learning curves on *MiniGrid-MultiRoom-N2-S4-v0*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

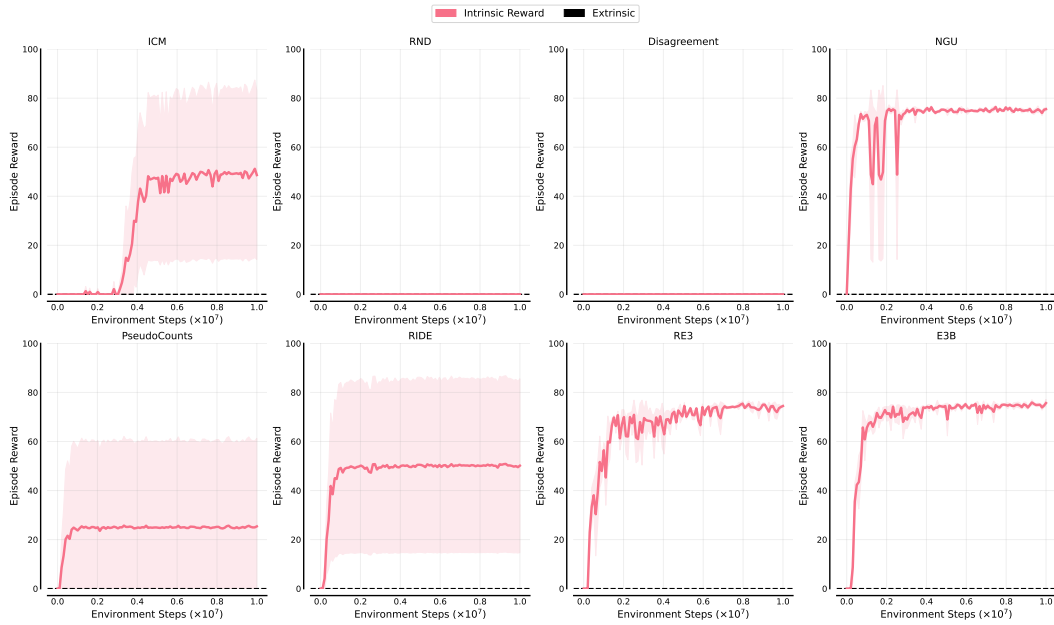


Figure 26: Learning curves on *MiniGrid-MultiRoom-N4-S5-v0*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

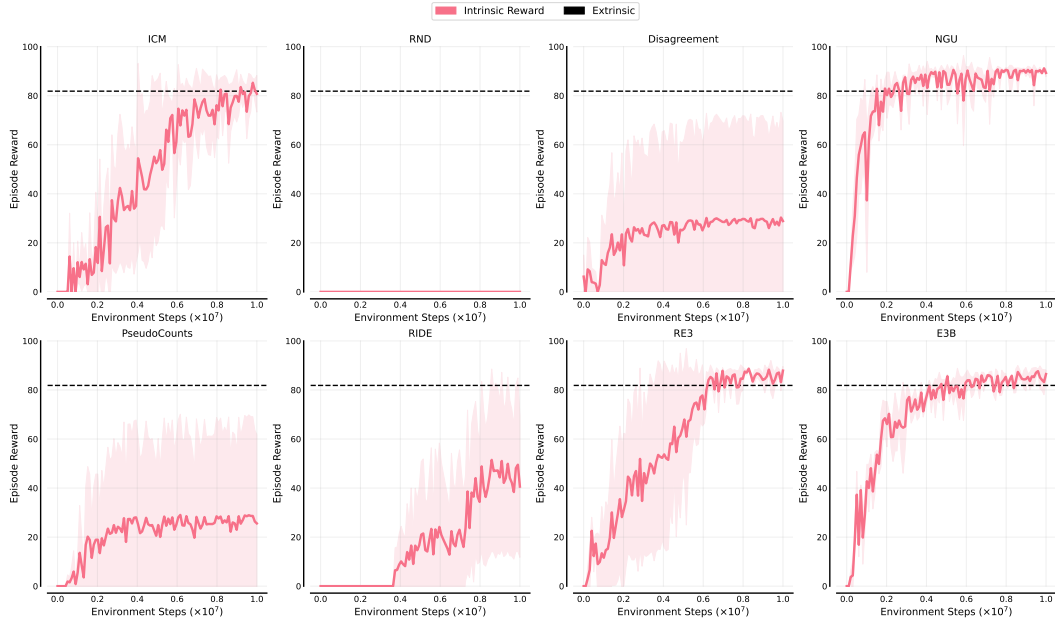


Figure 27: Learning curves on *MiniGrid-KeyCorridorS3R3-v0*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

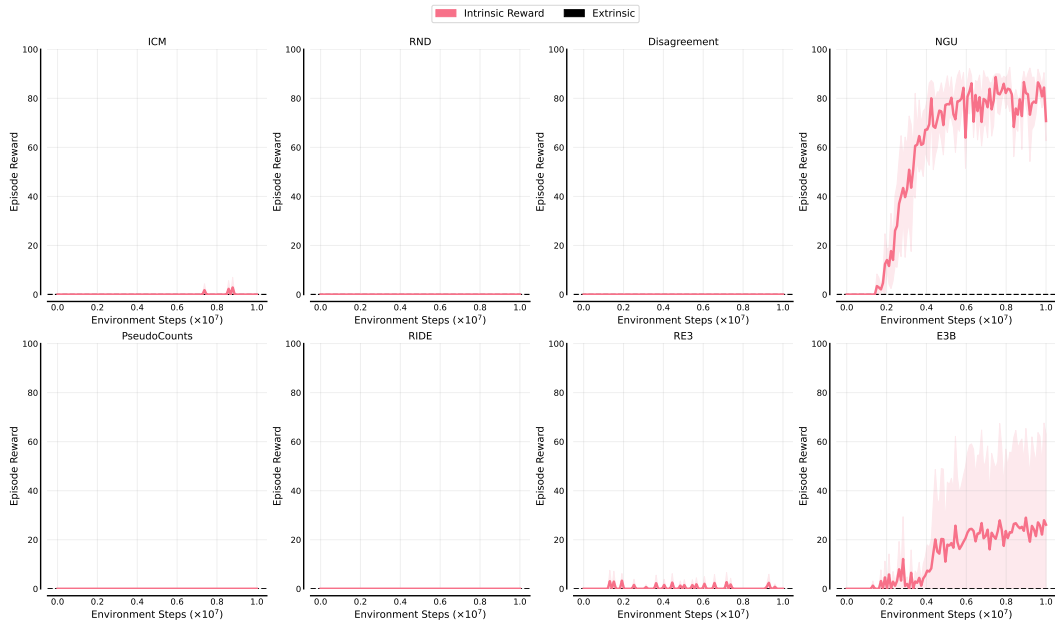


Figure 28: Learning curves on *MiniGrid-KeyCorridorS5R3-v0*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.

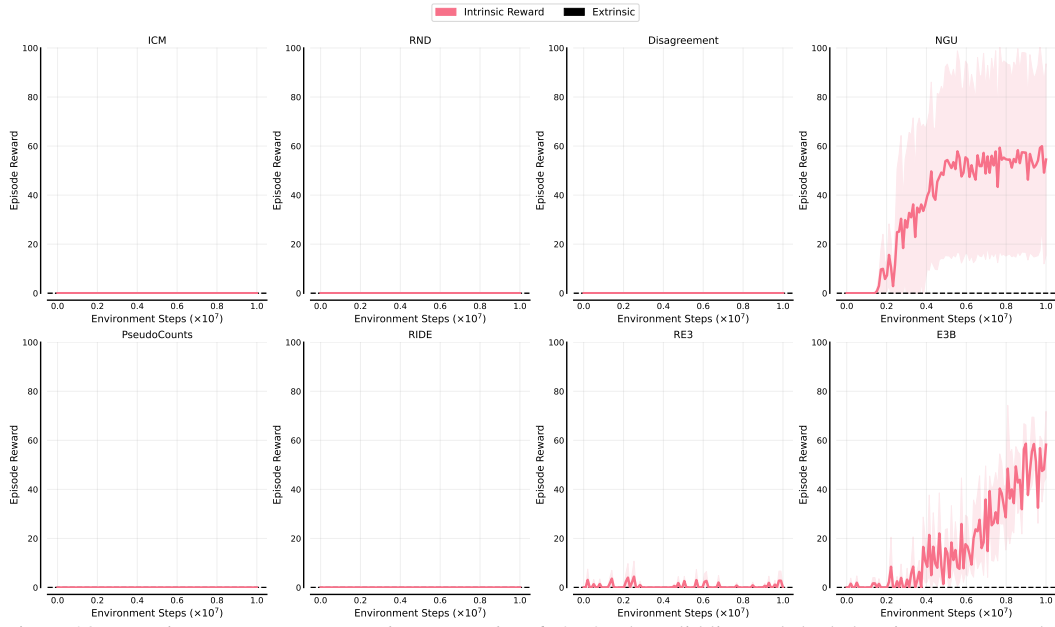


Figure 29: Learning curves on *MiniGrid-KeyCorridorS6R3-v0*. The solid line and shaded regions represent the mean and standard deviation computed with five random seeds, respectively.