LECO: Learnable Episodic Count for Task-Specific Intrinsic Reward

Daejin Jo* **Sungwoong Kim*** **Daniel Wontae Nam***

Taehwan Kwon **Seungeun Rho** **Jongmin Kim** **Donghoon Lee**

Kakao Brain
Seongnam, South Korea
{daejin.jo, swkim, dwtnam, taehwan.kwon, seungeun.rho, jmkim, dhlee}
@kakaobrain.com

Abstract

Episodic count has been widely used to design a simple yet effective intrinsic motivation for reinforcement learning with a sparse reward. However, the use of episodic count in a high-dimensional state space as well as over a long episode time requires a thorough state compression and fast hashing, which hinders rigorous exploitation of it in such hard and complex exploration environments. Moreover, the interference from task-irrelevant observations in the episodic count may cause its intrinsic motivation to overlook task-related important changes of states, and the novelty in an episodic manner can lead to repeatedly revisit the familiar states across episodes. In order to resolve these issues, in this paper, we propose a learnable hash-based episodic count, which we name LECO, that efficiently performs as a task-specific intrinsic reward in hard exploration problems. In particular, the proposed intrinsic reward consists of the episodic novelty and the task-specific modulation where the former employs a vector quantized variational autoencoder to automatically obtain the discrete state codes for fast counting while the latter regulates the episodic novelty by learning a modulator to optimize the task-specific extrinsic reward. The proposed LECO specifically enables the automatic transition from exploration to exploitation during reinforcement learning. We experimentally show that in contrast to the previous exploration methods LECO successfully solves hard exploration problems and also scales to large state spaces through the most difficult tasks in MiniGrid and DMLab environments.

1 Introduction

In reinforcement learning (RL), tackling the problem of sparse rewards still remains a great challenge. Many works have been proposed to develop dense synthetic reward, which is generally called intrinsic reward, to provide the motivation for exploration in the absence of extrinsic rewards from the environment [5, 26, 1, 28, 13, 38, 33, 10, 40, 29, 37, 15, 21]. Among them, count-based exploration [31, 4, 25, 24, 33, 23, 5], that encourages the RL agent to visit novel states, is one of the most popular classes for intrinsic motivation since it is simple and effective in many tasks, and more importantly it is theoretically justified [31, 20]. In general, the state novelty can be measured within a single episode or through an entire task across episodes. In recent RL problems, environments are often changed over episodes or even procedurally generated within the same task [7, 3, 22, 8, 17]. Therefore,

*Contributed equally.

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

per-episode novelty by *episodic count* has been widely adapted in modern hard exploration problems [1, 28, 38].

However, in high-dimensional and/or continuous state spaces, state counts need to be approximated in an abstract space since most states are likely to be unique and novel in the original space, leading to insufficient exploration [4, 25, 28, 1]. In addition, computation of episodic count at every time step based on the episodic memory could be burdensome if an episode lasts for a long time and therefore requires fast hashing [33]. Previous works have discretized the state space to obtain the hash code [33, 14, 9] for a fast pseudo-count, but they have not extensively applied it to recent difficult exploration problems such as procedurally generated and partially observable environments.

Moreover, existing count-based exploration methods have mostly employed task-agnostic state novelty, and thereby task-irrelevant count may disturb its intrinsic reward bonus to faithfully reflect task-related information [19, 2, 18, 40]. This problem could be severe in exploration by per-episode count. For example, task-wise familiar states across episodes can have high novelty in each episode, which confines overall exploration into smaller areas, while some states that are highly related to extrinsic rewards need to be repeatedly revisited even in a single episode for obtaining extrinsic rewards more densely.

In this work, we propose a hash-based episodic count that is learned to provide task-specific intrinsic motivation in solving hard exploration problems. The proposed **L**earnable **E**pisodic **CO**unt (LECO) produces task-specific intrinsic rewards that are efficiently and practically applicable to high-dimensional states, long-horizon episodes, procedurally generated environments, and highly sparse extrinsic rewards. In specific, the proposed intrinsic reward is composed of the episodic count-based novelty and the task-specific modulation. Our episodic count makes use of a vector quantized variational autoencoder (VQ-VAE) [34] to automatically obtain the discrete state codes for a fast counting. In addition, we exploit a learnable modulator to control the episodic novelty in relation to task-specific extrinsic rewards. The proposed LECO also adaptively shifts the RL phase from exploration to exploitation. Experimental results on the most difficult exploration tasks in procedurally generated environments of MiniGrid [7] and DMLab [3] show that LECO efficiently solves such tasks and outperforms previous state-of-the-art exploration methods.

Our main contributions can be summarized as follows:

- A novel learnable episodic count is proposed to efficiently provide task-specific intrinsic motivation to the RL agent under complex environments with sparse rewards.
- A complementary synthesis of learnable hash code based on VQ-VAE and learnable intrinsic reward modulator enables task-specific transition from exploration to exploitation.
- The proposed LECO demonstrates consistent performance improvements over previous exploration methods in solving various hard exploration benchmark tasks.

2 Related Work

Count-based exploration. Count-based exploration was first proposed in tabular tasks [31], which promotes the agent to reduce its uncertainty by visiting unfamiliar states corresponding to low counts. However, it is basically ineffective in high-dimensional state spaces since every state would be different. Therefore, several recent works have utilized count-based exploration for high-dimensional environments. For instance, pseudo counts are derived from density models using context tree switching [4] or PixelCNN [25]. More recently, RND [5] has approximated the state visitation count by the difference between a random fixed target network and a trained predictor network. RND has also been used in other methods for measuring the state novelty [1, 38, 37]. On the other hand, a number of algorithms have transformed the high-dimensional state to the discretized hash code for fast visitation counting by a hash table [33, 10]. Similarly, we obtain the episodic count by the discretized hash code. However, we exploit learnable hash codes by VQ-VAE that is trained to approximate the state density. In addition, different from [14] that directly produces a global code by VQ-VAE, we concatenate local codes as in the original VQ-VAE for generating a single state code.

Task-specific intrinsic motivation. While count-based exploration methods have typically utilized the state novelty in a task-agnostic manner, curiosity-based methods have tried to consider task-related information for intrinsic motivation by learning the environment dynamics [26, 29, 19, 28]. They

have developed intrinsic rewards based on the prediction error upon the agent's actions. Among them, recently, a number of approaches have exploited the task-dependent state novelty using the information bottleneck in order to ignore task-irrelevant distractions [19, 2, 18]. Meanwhile, there have been a number of attempts to directly learn the intrinsic reward function to ultimately optimize the task-specific extrinsic rewards [40, 39, 36]. Their algorithms are generally performed under the meta-gradient framework. LECO also adopts the idea of learnable rewards and the gradient-based bi-level optimization [40] to construct a task-specific modulator for intrinsic rewards. However, unlike those works that do not work well with sparse extrinsic rewards, LECO takes a full advantage of intrinsic motivation by the task-agnostic state novelty based on the episodic count before obtaining the extrinsic rewards.

Exploration for procedurally generated environments. One of the important challenging RL problems is robust exploration in a procedurally generated environment where the environment (partially) changes between episodes, and hence the agent needs to explore a very large state space and have strong generalization ability. Many recent RL benchmarks deal with procedurally generated environments [7, 3, 22, 8, 17], and accordingly state-of-the-art exploration algorithms have tried to improve sample efficiency in such environments. RIDE [28] has used a change between successive states on a latent space as an intrinsic reward while AGAC [13] has encouraged action diversity by an adversarial policy. NovelD [38] has exploited regulated difference of the novelty between consecutive states, and MADE [37] has devised exploration by maximizing the deviation from previously explored regions. AMIGo [6] has proposed a teacher that generates count-based intrinsic goals. In contrast to these handcrafted or task-agnostic intrinsic rewards, LECO learns to leverage task-level information to guide the count-based exploration especially at the episode level due to the procedurally generated environment. Moreover, LECO also shows improved performances on DMLab tasks with higher-dimensional partial observations.

3 Learnable Episodic Count via Task-Specific Modulation

3.1 Notation

Our problem setting follows a typical RL problem described using Markov Decision Process defined by a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, and a transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$. An RL agent interacting with the environment samples its actions from a policy, which is an action distribution function $\pi \rightarrow \mathcal{P}(\mathcal{A})$, and receives a reward, $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, upon state transition. The objective of the agent is to learn a policy that maximizes the expected sum of rewards $G_t = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}\right]$, where $\gamma \in [0, 1]$ is a discount factor.

Among various sub-problems that RL faces, balancing the trade-off between exploration and exploitation still remains a fundamental problem [32]. Especially, efficient exploration is important when dealing with sparse rewards. One approach to solving such problem is to design an algorithm that directly generates exploring behavior. More specifically, a widely adopted method is to design an intrinsic motivation through a form of a reward such that the reward which the agent receives at time t is given by $r_t = r_t^e + \alpha r_t^i$, where r^e is the extrinsic reward from the environment, r^i is the intrinsic reward designed for exploration, and α is a hyperparameter for scaling [28].

3.2 LECO Intrinsic Reward

The use of state visitation count for intrinsic rewards based on the state novelty has been a core concept in several previous studies [28, 1, 38, 13, 30, 16] due to its simplicity, task-agnostic property, and theoretical basis. However, such state-novelty measured from visitation counts monotonically decreases as more experiences accumulate, equally for both important and unimportant states. Thus, an agent would be discouraged to visit even the task-important states after a certain period of time, and the exploration towards those states vanishes [1]. Furthermore, an intrinsic bonus based on the state visitation could lead the agent to only pursue any state novelty, especially in the partially-observable and sparse reward environments. For example, imagine a task in which the goal is to find a colored key hidden in a box and unlock a door that has the same color as the key. If the box changes its color to a random color every time when it is opened, the agent would repeatedly open and close the box just to obtain the intrinsic bonus. In this task, the agent should be encouraged to find the door after obtaining the key and discouraged to repeat task-irrelevant actions. We directly build upon this idea

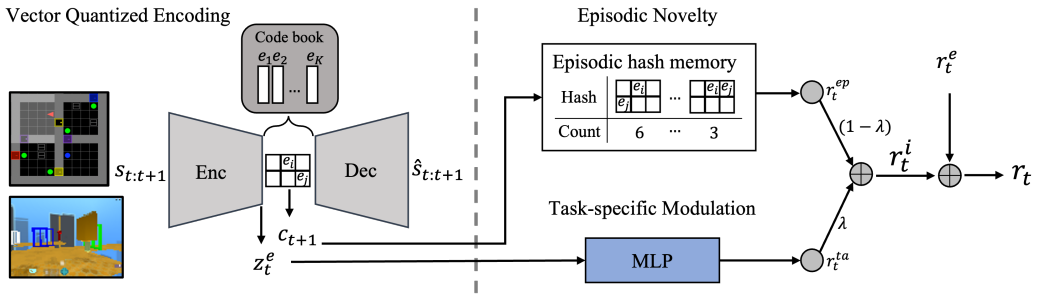


Figure 1: The schematic diagram of the intrinsic reward in LECO. The states at time t and $t+1$ are encoded and quantized to the embedding vector z_t^e and hash c_{t+1}. Then, each is fed into the task-specific modulator and episodic counter, respectively, to produce the task-specific modulation r_t^{ta} and episodic intrinsic reward r_t^{ep}, which are then combined to produce LECO's intrinsic reward r_t^i.

by decomposing the intrinsic reward into the state-novelty and task-specific modulation, to balance exploration and exploitation, respectively.

We design our intrinsic motivation to encourage the discovery of state novelty that is adaptively controlled to maximize the extrinsic return, hence the name **L**earnable **E**pisodic **CO**unt for task-specific motivation. Specifically, we define the general form of intrinsic reward as a sum of two parts,

$$r_t^i(a_{t-1}, s_t, a_t, s_{t+1}) = (1-\lambda) r_t^{\text{ep}}(s_{t+1}) + \lambda r_t^{\text{ta}}(a_{t-1}, s_t, a_t), \tag{1}$$

where $r_t^{\text{ep}}(s_{t+1}) = 1/\sqrt{N_{\text{ep}}(s_{t+1})}$ is the intrinsic reward from the episodic state-novelty, $N_{\text{ep}}(s)$ is the episodic visitation count of state s, and r^{ta} is the task-specific modulation with $\lambda \in [0, 1]$ as its weighting factor. The first term measures the episodic novelty by state visitation count within the current episode that is initialized to zero at the beginning of each episode. The second term learns to modulate the episodic novelty in the first term according to whether the state is beneficial or harmful in achieving the task-specific objective. Note that the previous action a_{t-1} is included as the input to reward calculation to provide extra transition information. A schematic diagram of the proposed intrinsic reward can be found in Figure 1.

LECO comprises of two submodules: 1) a learnable hash that encodes input observations to discrete hash codes for episodic counts and 2) a modulator that maximizes the task-specific extrinsic return by adaptively scaling the episodic novelty. Here, the modulator additively scales the episodic novelty. Compared to multiplicative scaling, the additive modulation allows the episodic novelty to be retained solely when the modulator does not generate enough signals due to the scarcity of extrinsic rewards.

3.3 Episodic Count via Vector Quantized Hashing

Previous researches have extended classic state visitation counts [31] to high-dimensional state spaces [4, 33]. A generalization of the classic count-based approach by locality-sensitive hashing (LSH) on the continuous embedding space was proposed [33] where the latent space is obtained by an autoencoder (AE). The use of a hash table in their approach enables counting the occurrences in a constant time, taking the advantage of scalability, especially in modern distributed RL systems [12, 11, 27]. However, such LSH does not take into account the spatial information due to its data independency [35]. As an alternative to AE, we propose to utilize vector quantized variational autoencoder (VQ-VAE) [34] that directly optimizes the discrete compressed representation of a state, producing a hash code without LSH.

In LECO, a state s_t is mapped to a hash code c_t by vector quantized hashing based on VQ-VAE comprised of an encoder, a decoder, and a codebook with θ_{enc}, θ_{dec}, and $\theta_e = \{e_k \in \mathbb{R}^D\}_{k=1}^K$ as their respective parameters. Here, K is the number of codes, and e_k is the D-dimensional embedding of the kth code. In detail, first, the input state s_t is encoded to a spatial representation $z_t^e = f_{\text{enc}}(s_t; \theta_{\text{enc}})$, where $z_t^e \in \mathbb{R}^{w \times h \times D}$, and $f_{\text{enc}}(\cdot; \theta_{\text{enc}})$ is an encoder of VQ-VAE. Then, the representation z_t^e is mapped to z_t^q by finding the nearest vector in the codebook θ_e for each vector in the spatial position $i \in \{1, 2, ..., w \times h\}$. To be more specific, let us denote the vector in position i of z_t^e and z_t^q as $z_t^e(i)$ and $z_t^q(i)$, respectively. Then, $z_t^q(i) = \arg\min_{e_k \in \theta_e} \|z_t^e(i) - e_k\|_2$, and the hash code is defined as

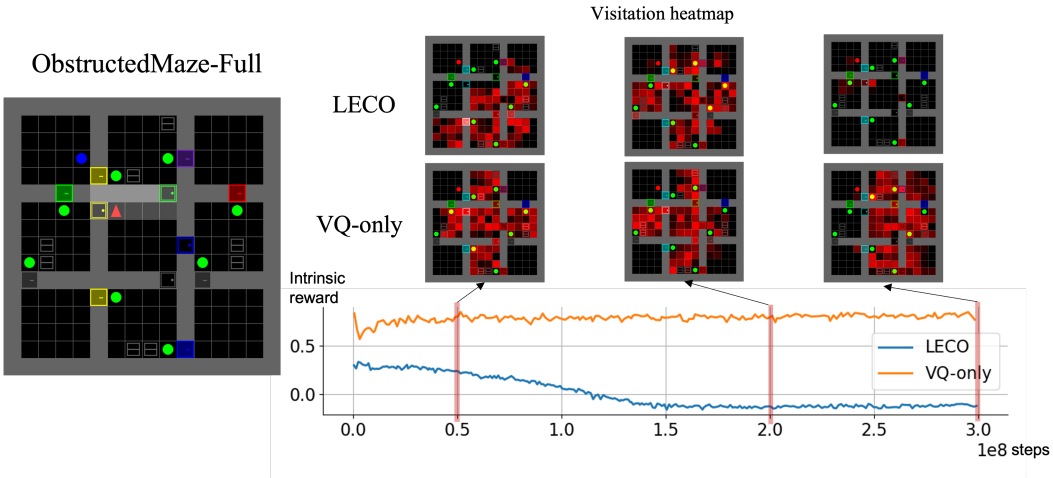


Figure 2: `ObstructedMaze-Full` Environment (**Left**). The changes in the coverage of visitations (**Right Top**) as the training progresses show that LECO modulates the episodic reward (**Right Bottom**) and so the agent focuses on the task-specific objective. On the other hand, the agent with intrinsic motivation from the episodic count only, labeled as VQ-only, remains distracted by the episodic state-novelty even in the late training phase. We provide more examples with further discussion in subsection D.2 in Appendix.

the sequence of indices such that $c_t = [k_i]_{i=1}^{w \times h}$ where $k_i = \arg\min_k \|z_t^e(i) - e_k\|_2$. Finally, the decoder then predicts the mean $\mu = f_{\text{dec}}(z_t^q; \theta_{\text{dec}})$ of an isotropic Gaussian distribution $p(s|z_t^q)$ with $\sigma^2 = 1$, where $f_{\text{dec}}(\cdot; \theta_{\text{dec}})$ is the decoder of the VQ-VAE.

Following the conventional VQ-VAE training, the hash function is updated according to the gradient

$$\nabla_{\theta_{\text{vq}}} \mathcal{J}^{\text{vq}} = -\sum_{t=0}^{T} \nabla \Big(\log p(s_t|z_t^q) + \|\mathbf{sg}[z_t^e] - z_t^q\|_2^2 + \|z_t^e - \mathbf{sg}[z_t^q]\|_2^2 \Big), \tag{2}$$

where $\mathbf{sg}$ stands for stopgradient operator, T is an unroll length of the agent, and $\theta_{\text{vq}} = \{\theta_{\text{enc}}, \theta_{\text{dec}}, \theta_e\}$.

3.4 Task-Specific Intrinsic Modulation

In a sparse reward environment, when an extrinsic reward signal has little or no presence, having a task-agnostic intrinsic motivation can be significantly beneficial in guiding the agent to efficiently explore the environment until the discovery of more extrinsic signals. However, a task-general intrinsic reward such as the inverse of the episodic count may become a distracting signal in the later exploitation since it is not reduced with the progression of learning. We thereby propose to include task-specific modulation that aims to ultimately maximize the extrinsic return in the intrinsic reward formulation of LECO. The task-specific modulation can be seen as a dynamic factor for controlling the exploration and exploitation, based on the extrinsic rewards earned from the environment.

For this, we leverage the bi-level optimization proposed in LIRPG [40]. First, let policy π parameterized by θ be updated using the policy gradient on the sum of intrinsic and extrinsic rewards,

$$\theta' = \theta + \eta \nabla_\theta \mathcal{J}^{e+i} = \theta + \eta \nabla_\theta \Big(\mathbb{E}_{\pi_\theta} [\sum_{t=0}^{\infty} \gamma^t (r_t^e + \alpha r_t^i)] \Big), \tag{3}$$

where η is the step size. We then construct our task-specific modulator for the intrinsic reward in Equation 1 as a parametric model such that $r_t^i = (1-\lambda) r_t^{\text{ep}} + \lambda r_t^{\text{ta}}(a_{t-1}, s_t, a_t; \theta_{\text{ta}})$ where θ_{ta} is the modulator parameters. Now, the task-specific modulator is trained to maximize only the extrinsic rewards using the corresponding policy gradient such that

$$\theta'_{\text{ta}} = \theta_{\text{ta}} + \eta_{\text{ta}} \nabla_{\theta_{\text{ta}}} \mathcal{J}^e = \theta_{\text{ta}} + \eta_{\text{ta}} \nabla_{\theta_{\text{ta}}} \Big(\mathbb{E}_{\pi_{\theta'}} [\sum_{t=0}^{\infty} \gamma^t r_t^e] \Big), \tag{4}$$

where η_{ta} is the step size for updating θ_{ta}. Here, it is noted that the expectation is performed on the updated parameters θ' that is affected by our intrinsic reward and accordingly θ_{ta}. Therefore, θ_{ta} is updated by the chain rule such that $\nabla_{\theta_{\text{ta}}}\mathcal{J}^e = \nabla_{\theta'}\mathcal{J}^e\nabla_{\theta_{\text{ta}}}\theta'$. This can be considered a well-known bi-level optimization framework popularly used in meta-learning, where the inner-loop and the outer-loop are corresponding to Equation 3 and Equation 4, respectively.

Model for task-specific modulation. After executing an action a_t, a task-specific modulation $r_t^{\text{ta}} \in (-1, 1)$ is modeled as

$$r_t^{\text{ta}}(a_{t-1}, s_t, a_t; \theta_{\text{ta}}) = \tanh\left({\mathbb{1}_{a_{t-1}}}^\top f_{\text{ta}}\big(z_t^e, a_t; \theta_{\text{ta}}\big)\right), \tag{5}$$

where $\mathbb{1}_a$ is the one-hot vector for action a, $f_{\text{ta}} : \mathbb{R}^D \to \mathbb{R}^{|\mathcal{A}|}$ is a MLP network, and z_t^e is shared with VQ-VAE based encoding to make the modulator efficient in training. We choose $\tanh$, which generates both positive and negative modulation, so that even in the absence of meaningful signals from episodic counts, the modulator can act according to LIRPG and converge to a value that maximizes the extrinsic objective.

4 Experiments

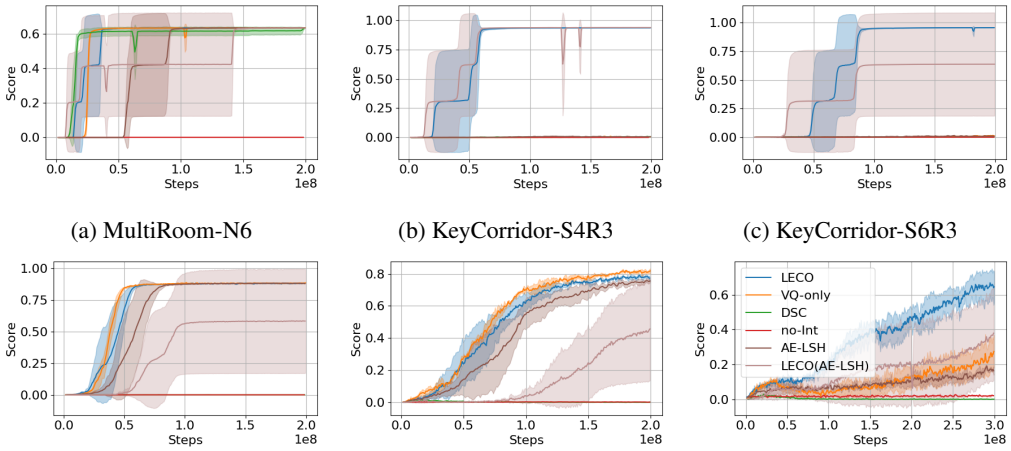
The key concept of LECO is the learnable and modulated episodic count. In other words, compared to a naive episodic count that assigns the state novelty proportional to its visitation count within an episode, LECO aims to produce low intrinsic rewards to task-irrelevant states regardless of its episodic count. We demonstrate such behavior by comparing the mean intrinsic rewards obtained by LECO to those from exploration with just the episodic count-based intrinsic reward in Figure 2. The results show that while the episodic novelty remains high, modulation in LECO decreases the intrinsic reward such that the explorative behavior has vanished towards the end of training and the agent focuses on the task-specific exploitation. Apart from the novelty based on the agent's location, in an environments where agent can pickup and drop objects, such actions can create task-irrelevant novel states, further complicating the search space. The task-specific modulation of LECO can mitigate those task-irrelevant behaviors as shown in Figure 10 and Figure 12 in Appendix.

Environments. As LECO operates on multiple components, we design the experiments to thoroughly cover various aspects, especially the adaptiveness and the scalability. Since the real world as an environment is nonstationary and requires high-dimensional perception, it is crucial for an agent to efficiently explore the constantly changing environment, even at a large scale. Thus, we first demonstrate the explorative performance of LECO in procedurally generated environments of MiniGrid [7]. Then, we further extend the experiments to large scale environments of DMLab tasks.

Baselines. In order to present the contribution of each component of our proposed method, we compare LECO to its variants that incrementally add each component to a baseline without the intrinsic motivation which we denote as *no-Int* for the rest of the section. Similarly, a variant with only the vector quantization episodic counts is denoted as *VQ-only*. Additionally, a variant that uses AE-LSH [33] as the hash function is denoted as *LECO(AE-LSH)*. Furthermore, in order to show that our proposed algorithm is beyond a mere combination of VQ-VAE and LIRPG, we compare the LECO to a straightforward combination of the two, denoted as *LECO-naive*.

The no-Int baseline is an IMPALA-based agent [12], with minor architectural modifications according to the environments. Details on the base model and the architectures can be found in Appendix A. Other baselines include *DSC* (Down Sampled Cell) [10], *AE-LSH* [33], *NovelD* [38], and *AGAC* [13], where DSC uses a non-learnable hash for state compression and AE-LSH uses a learnable hash. NovelD and AGAC are state-of-the-art exploration methods for procedurally generated environments. Here, for a fair assessment of LECO, we implement the episodic count of NovelD and AGAC using the VQ hashing as done in LECO for both of MiniGrid and DMLab tasks.

Experimental Setup. In MiniGrid, LECO was trained using two A100 GPUs with a batch size of 768 for 18 hours. In DMLab, we used eight V100 GPUs with a batch size of 576 for 8 hours. The unroll length was $T = 96$ for all tasks and same LSTM-based policy network architecture was used for LECO and all other baselines. Details on hyperparameters, model architectures, and training



(a) MultiRoom-N6 (b) KeyCorridor-S4R3 (c) KeyCorridor-S6R3

(d) ObstructedMaze-1Q (e) ObstructedMaze-2Q (f) ObstructedMaze-Full

Figure 3: Comparison of LECO and its variations on six selected MiniGrid tasks. The final choice of design (LECO) shows superior performance compared to its variants. The shaded area represents a range of standard deviation over 3 runs with different random seeds.

settings are provided in Appendix B. Official codes to run the algorithm and the experiments will be made public soon after the publication of this paper.

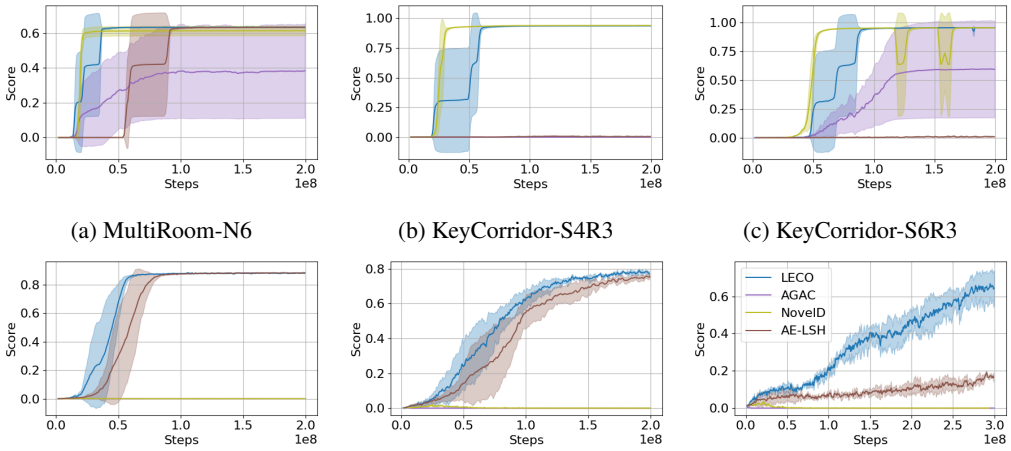
4.1 MiniGrid

MiniGrid [7] is a procedurally generated gridworld environment specifically designed to test and develop exploration algorithms under sparse rewards and partial observations. A typical world in MiniGrid consists of $M \times N$ tiles, where each tile is assigned to be objects such as floor, wall, lava, door, key, ball, box, and goal. The agent deployed navigates through the world, collecting objects and solving sub-tasks such as opening a door using a key, to reach a goal placed somewhere in the world. Of the many types of environments included, we test the performance of LECO in three types of MultiRoom, KeyCorridor, and ObstructedMaze environments. More specifically, we use total of six configurations of those environments namely `MultiRoom-N6(MRN6)`, `KeyCorridorS4R3(KCS4R3)`, `KeyCorridorS6R3(KCS6R3)`, `ObstructedMaze-1Q(OM1Q)`, `ObstructedMaze-2Q(OM2Q)`, and `ObstructedMaze-Full(OMFull)`, roughly ordered in increasing level of difficulty. The choices were made to reflect a wide range of difficulty including the most difficult task, while evaluating on similar tasks used by the previous baselines.

We emphasize that for all tasks the baselines and LECO only use the original partial input observations, in which a 7x7x3 tensor given as the partially-observable grid cell is the input observation for both the policy network and intrinsic module as in [6]. This setting is more challenging than the other settings such as the partial observation for the policy and full observation for the intrinsic module as in [28, 38, 30], and the partial observation for the policy and meta information such as the agent's position for the intrinsic module as in [13].

We first compare LECO to its variations as shown in Figure 3. The variations include no-Int, DSC, AE-LSH, VQ-only, and LECO(AE-LSH). The results show that LECO solved all six tasks while VQ-only solved three, `MRN6`, `OM1Q`, `OM2Q`, and show slow learning trend in `OMFull`. On the other hand, compared to DSC, learnable hashing methods for using episodic count as an intrinsic reward but with heuristic quantization, VQ-only outperforms DSC as DSC solved only one out of the six tasks. While AE-LSH solved the same number of tasks as VQ-only, the scores of VQ-only are higher in all tasks.

Without the task-specific modulators, the episodic count-only methods such as VQ-only and AE-LSH, tend to perform worse in KeyCorridor tasks than in ObstructedMaze tasks due to the task-irrelevant state novelties. More concretely, in `OM1Q` and `OM2Q`, the locked doors are generally located near the



(a) MultiRoom-N6 (b) KeyCorridor-S4R3 (c) KeyCorridor-S6R3

(d) ObstructedMaze-1Q (e) ObstructedMaze-2Q (f) ObstructedMaze-Full

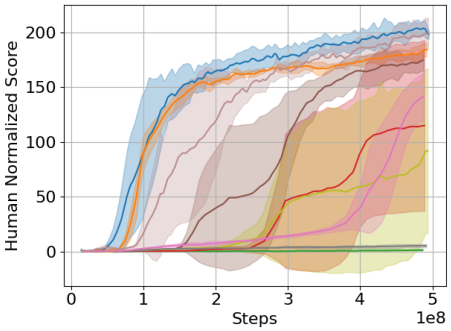
Figure 4: Comparison of LECO to baselines on six selected MiniGrid tasks. LECO shows the most promising performance compared to related existing methods in the field. The shaded area represents a range of standard deviation over 3 runs with different random seeds.

keys making it easier for the agent to discover the sequence of opening the locked door by exploration, while in KeyCorridor, even though the map is more simple, the doors are generally placed further away from the keys and thus increased search depth hinders the episodic count-only methods from opening the locked door and eventually solving tasks. We visualize the agent's behavior with and without the modulation in subsection D.1 of Appendix which depicts the episodic count-only agent choosing task-irrelevant actions along the way in KeyCorridor task.

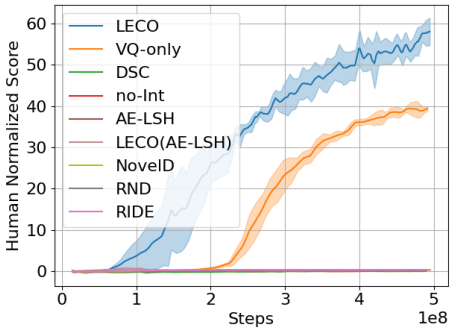
As shown in Figure 3, LECO(AE-LSH) generally performs better than AE-LSH except `OM1Q` and `OM2Q`. However, the performances of LECO(AE-LSH) are lower than those of LECO on all tasks, which shows the benefits from VQ-VAE for the episodic counts. We analyze the benefits of LECO compared to LECO(AE-LSH) over the hash space (see subsection D.3). As shown by the results of no-Int, a policy without any intrinsic reward is unable to solve any of the six tasks.

Combining all comparisons gives indications that 1) episodic count enables training in sparse reward exploration tasks with procedurally generated environments, 2) vector quantization-based counting is more effective than heuristic cell discretization (DSC) as well as AE-LSH, and 3) the task-specific modulation of LECO further improves the performances of the episodic count-based approach.

Moreover, we compare the performance of LECO with other baselines of AE-LSH, AGAC, and NovelD on the same six MiniGrid tasks. As shown in Figure 4, only LECO solves all six tasks including `OMFull`. Here, it is noted that the original implementations of AGAC and NovelD used the agent's position and the full observations, respectively, for the episodic counts in their methods while we used the original partial observations in our implementations of AGAC and NovelD. In fact, the original performances of NovelD and AGAC are better in their papers, and they have also shown to solve OMFull in their papers. However, we observe that, when we correct their official codes to utilize only the partial observations in obtaining the episodic counts, their performances are dropped as in our results. Recently, MADE [37] has shown to produce state-of-the-art performances on MiniGrid including `OMFull`, and MADE seems to be a bit more sample efficient than LECO according to Figure 7 in [37]. However, in this paper, it is somewhat difficult for us to compare the learning progress of LECO with that of MADE since we failed to reproduce their performances given rough descriptions on MiniGrid implementations in their paper and the absence of support for MiniGrid experiments in their official codes. In MultiRoom and KeyCorridor environments, the increasing rates in scores of LECO is slightly slower than those of NovelD since the modulator in LECO requires a bit more experiences to train, however, the advantage of having the modulator is clearly shown through significant performance improvement in the harder exploration tasks. In addition, LECO is more stable in the later phase of KeyCorridor training.



(a) lasertag_three_opponents_small



(b) lasertag_three_opponents_large

Figure 5: Learning curves on two selected DMLab tasks. Compared to all of its variants and baselines LECO shows the highest performance. Especially in `lasertag_three_opponents_large`, only LECO and VQ-only is able to solve the task. The y-axis represents the human normalized score. The shaded area represents the range of a standard deviation over 3 runs of random seeds.

4.2 DMLab

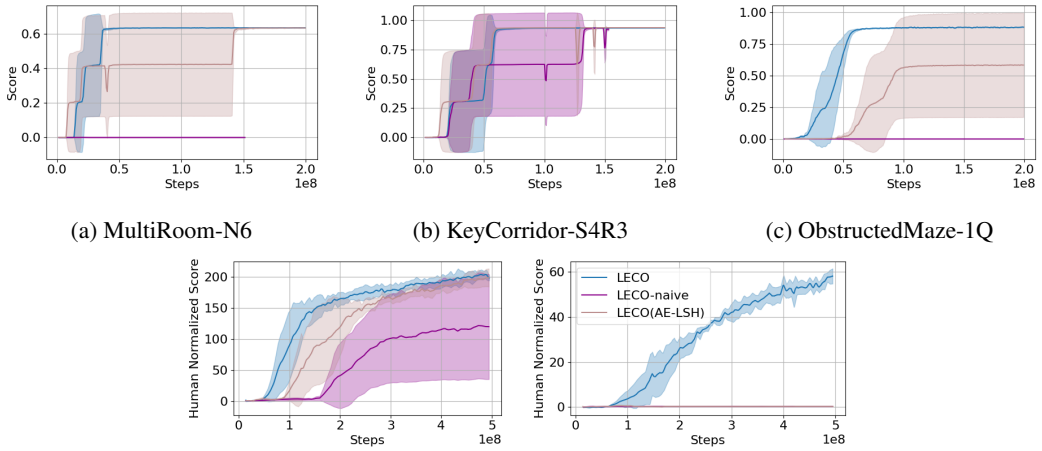
DMLab [3] is a learning environment consisting of various tasks, each involving different visuals and objectives in a simulated 3D world. We consider DMLab for its partial-observability that naturally arises from first-person perspective and its complexity from image-based observation with fine details and randomly generated levels to demonstrate the scalability of LECO. We compare LECO and LECO(AE-LSH) to no-Int, episodic count methods based on AE-LSH, DSC, and VQ-only, and state-of-the-art exploration methods including NovelD, RND [5], and RIDE [28] on two individual exploration-heavy tasks. For RND and RIDE, the architectures of neural networks are based on the RND network used in NovelD for MiniGrid and the forward/inverse networks used in RIDE for MiniGrid [2]. We use the same RND networks for RND and NovelD and the same architecture of VQ for LECO, NovelD, and RIDE in their episodic count module.

We select two specific tasks from DMLab, `lasertag_three_opponents_small` and `lasertag_three_opponents_large`, in order to show LECO's capability to solve exploration tasks at a large scale. In both tasks, an agent is deployed in a procedurally generated 3-D environment where it has to search and shoot three opponents moving around the maze-like map (see Figures 16 to 18 for example screenshots). Since the reward is only given when the agent successfully tags the opponents by shooting a laser, these tasks are known to be difficult to solve without an advanced exploration strategy and are characterized as sparse reward problems. The `small` and `large` stand for the size of the map, which determines the reward sparsity. In Figure 5, we present the learning curves of LECO and baselines on these tasks. Each task has its own independent agent, but the architectures and hyperparameters remain consistent across these tasks. To the best of our knowledge, experiments for RND, RIDE, and NovelD on DMLab-lasertags has not been published so far. As shown in Figure 5, these models show decent performances on `lasertag_three_opponents_small`, but perform worse than LECO and moreover do not solve the task of `lasertag_three_opponents_large` at all.

4.3 LECO vs LECO-naive.

One of the key contributions of the LECO is the formulation of task-specific modulation of the task-agnostic episodic count in the additive form. In order to show that LECO is more than a simple combination of VQ-VAE and LIRPG, we experimentally compare LECO to a naive combination of VQ-VAE and LIRPG where we directly optimize the episodic counter using the meta-gradient of extrinsic rewards. More concretely, we define the intrinsic reward $r_t^i(a_{t-1}, s_t, a_t, s_{t+1}) = r_t^{\text{ta}}(a_{t-1}, s_t, a_t, r_t^{\text{ep}}(s_{t+1}))$ and denote as *LECO-naive*. Just from the formulation we can speculate that it would be insufficient to solve sparse reward problems since it cannot generate dense signals from sparse extrinsic rewards, especially in the early stages of RL. We

[2] `https://github.com/tianjunz/NovelD`



(a) MultiRoom-N6 (b) KeyCorridor-S4R3 (c) ObstructedMaze-1Q

(d) lasertag_three_opponents_small (e) lasertag_three_opponents_large

Figure 6: Comparison between different variants of LECO on three selected MiniGrid tasks (a)-(c) and the two DMLab tasks (d)-(e). LECO clearly outperforms LECO-naive in all tasks. The shaded area represents a range of standard deviation over 3 runs with different random seeds.

compare the performances of LECO to LECO-naive along with LECO(AE-LSH) on Minigrid(`MRN6`, `KCS4R3`, and `OM1Q`), and DMLab(`lasertag-three_oppenents_small` and `lasertag-three_oppenents_large`). As shown in Figure 6, the performances of LECO-naive are lower than those of both LECO and LECO(AE-LSH) on all tasks, showing that the proposed additive formulation enables taking the full advantages of both parts and accordingly to solve very hard exploration problems with automatic transition from exploration to exploitation.

5 Conclusion

This paper proposes a learnable episodic count to produce task-specific intrinsic rewards for hard exploration problems including high-dimensional state spaces and procedurally generated environments. In specific, the hash code based on VQ-VAE is used for computing the episodic state count and the corresponding state novelty, and this episodic novelty is regulated by the task-specific modulator that is trained to maximize the extrinsic rewards. Experimental results demonstrate that the proposed task-specific exploration based on the episodic count significantly outperforms the previous state-of-the-art exploration methods on the most difficult tasks of MiniGrid and DMLab. However, the proposed intrinsic reward also has a number of limitations. First limitation comes from solving tasks that require intensive leveraging of the inter-episode novelty. For example, objects of negative reward could be common among different episodes, and in that case, the inter-episode novelty can lead an agent to avoid those objects. Second limitation is dealing with somewhat difficult bi-level optimization. The use of implicit gradients and the reformulated single-level problem can be explored to efficiently solve it. For future works, we will extend our work to combine the inter-episode novelty and devise a more efficient training of task-specific intrinsic reward to further improve the scalability. Moreover, we will perform experiments on all DMLab tasks with large-scale transformer-based networks and also apply to the other hard-exploration environments.

Acknowledgments and Disclosure of Funding

We would like to thank Brain Cloud Team at Kakao Brain for their support.

References

[1] 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.

[2] Chenjia Bai, Lingxiao Wang, Lei Han, Animesh Garg, Jianye Hao, Peng Liu, and Zhaoran Wang. Dynamic bottleneck for robust self-supervised exploration. In *International Conference on Neural Information Processing Systems*, 2021.

[3] Charles Beattie, Joel Z. Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, Julian Schrittwieser, Keith Anderson, Sarah York, Max Cant, Adam Cain, Adrian Bolton, Stephen Gaffney, Helen King, Demis Hassabis, Shane Legg, and Stig Petersen. Deepmind lab. *CoRR*, abs/1612.03801, 2016.

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

[5] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *International Conference on Learning Representations*, 2019.

[6] Andres Campero, Roberta Raileanu, Heinrich Kuttler, Joshua B. Tenenbaum, Tim Rocktäschel, and Edward Grefenstette. Learning with {amig}o: Adversarially motivated intrinsic goals. In *International Conference on Learning Representations*, 2021.

[7] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

[8] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International Conference on Machine Learning*, 2020.

[9] Dane Corneil, Wulfram Gerstner, and Johanni Brea. Efficient model–based deep reinforcement learning with variational state tabulation. In *International Conference on Machine Learning*, 2018.

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

[11] Lasse Espeholt, Raphaël Marinier, Piotr Stanczyk, Ke Wang, and Marcin Michalski. Seed rl: Scalable and efficient deep-rl with accelerated central inference. In *International Conference on Learning Representations*, 2020.

[12] 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.

[13] Yannis Flet-Berliac, Johan Ferret, Olivier Pietquin, Philippe Preux, and Matthieu Geist. Adversarially guided actor-critic. In *International Conference on Learning Representations*, 2021.

[14] Petros Giannakopoulos, Aggelos Pikrakis, and Yannis Cotronis. Neural discrete abstraction of high-dimensional spaces: A case study in reinforcement learning. In *2020 28th European Signal Processing Conference (EUSIPCO)*, pages 1517–1521, 2021.

[15] Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017.

[16] Xiao Jing, Zhenwei Zhu, Hongliang Li, Xin Pei, Yoshua Bengio, Tong Che, and Hongyong Song. Divide and explore: Multi-agent separate exploration with shared intrinsic motivations, 2022.

[17] Arthur Juliani, Ahmed Khalifa, Vincent-Pierre Berges, Jonathan Harper, Hunter Henry, Adam Crespi, Julian Togelius, and Danny Lange. Obstacle tower: A generalization challenge in vision, control, and planning. *CoRR*, abs/1902.01378, 2019.

[18] Jaekyeom Kim, Minjung Kim, Dongyeon Woo, and Gunhee Kim. Drop-bottleneck: Learning discrete compressed representation for noise-robust exploration. In *International Conference on Learning Representations*, 2021.

[19] Youngjin Kim, Wontae Nam, Hyunwoo Kim, Ji-Hoon Kim, and Gunhee Kim. Curiosity-bottleneck: Exploration by distilling task-specific novelty. In *International Conference on Machine Learning*, 2019.

[20] J. Zico Kolter and Andrew Y. Ng. Near-bayesian exploration in polynomial time. In *International Conference on Machine Learning*, 2009.

[21] Taehwan Kwon. Variational intrinsic control revisited. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[22] Heinrich Küttler, Nantas Nardelli, Alexander H. Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment. In *International Conference on Neural Information Processing Systems*, 2020.

[23] Marlos C. Machado, Marc G. Bellemare, and Michael Bowling. Count-based exploration with the successor representation. In *Thirty-fourth AAAI conference on artificial intelligence*, 2020.

[24] Jarryd Martin, Suraj Narayanan Sasikumar, Tom Everitt, and Marcus Hutter. Count-based exploration in feature space for reinforcement learning. In *Twenty-sixth International Joint Conference on Artificial Intelligence*, 2017.

[25] Georg Ostrovski, Marc G. Bellemare, Aaron van den Oord, and Remi Munos. Count-based exploration with neural density models. In *International Conference on Machine Learning*, 2017.

[26] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*, 2017.

[27] Aleksei Petrenko, Zhehui Huang, Tushar Kumar, Gaurav Sukhatme, and Vladlen Koltun. Sample factory: Egocentric 3d control from pixels at 100000 fps with asynchronous reinforcement learning. In *International Conference on Machine Learning*, 2020.

[28] Roberta Raileanu and Tim Rocktäschel. Ride: Rewarding impact-driven exploration for procedurally-generated environments. In *International Conference on Learning Representations*, 2020.

[29] Nikolay Savinov, Anton Raichuk, Raphaël Marinier, Damien Vincent, Marc Pollefeys, Timothy Lillicrap, and Sylvain Gelly. Episodic curiosity through reachability. In *International Conference on Learning Representations*, 2019.

[30] Mathieu Seurin, Florian Strub, Philippe Preux, and Olivier Pietquin. Don't do what doesn't matter: Intrinsic motivation with action usefulness. In *Proc. of IJCAI 2021*, 2021.

[31] 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.

[32] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[33] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. #exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2753–2762, 2017.

[34] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *International Conference on Neural Information Processing Systems*, 2017.

[35] Rongkai Xia, Yan Pan, Hanjiang Lai, Cong Liu, and Shuicheng Yan. Supervised hashing for image retrieval via image representation learning. In *Twenty-eighth AAAI conference on artificial intelligence*, 2014.

[36] Lunjun Zhang, Bradly C. Stadie, and Jimmy Ba. Learning intrinsic rewards as a bi-level optimization problem. In *Thirty-sixth Conference on Uncertainty in Artificial Intelligence (UAI)*, 2020.

[37] Tianjun Zhang, Paria Rashidinejad, Jiantao Jiao, Yuandong Tian, Joseph E. Gonzalez, and Stuart Russell. Made: Exploration via maximizing deviation from explored regions. In *International Conference on Neural Information Processing Systems*, 2021.

[38] Tianjun Zhang, Huazhe Xu, Xiaolong Wang, Yi Wu, Kurt Keutzer, Joseph E Gonzalez, and Yuandong Tian. Noveld: A simple yet effective exploration criterion. In *Advances in Neural Information Processing Systems*, 2021.

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

[40] Zeyu Zheng, Junhyuk Oh, and Satinder Singh. On learning intrinsic rewards for policy gradient methods. In *International Conference on Neural Information Processing Systems*, 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] We have discussed the limitations and corresponding future works in the conclusion section.
   (c) Did you discuss any potential negative societal impacts of your work? [No] This paper will not create any 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? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (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] We will release the code and data soon.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] More details including hyperparameters are included in appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We include the standard deviation of the results with respect to multiple runs.
   (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]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] We used only the benchmark simulators.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] There is no personal data in our experiments.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

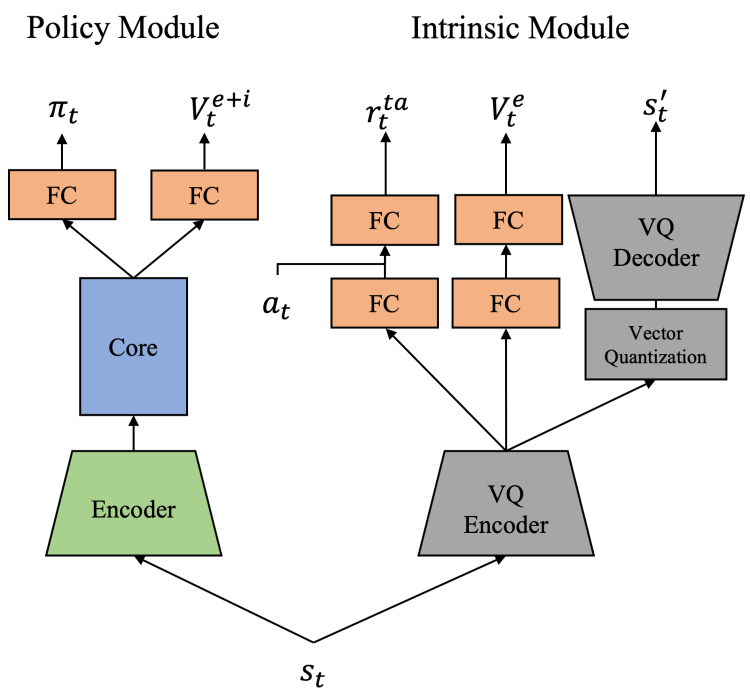


Figure 7: Diagram of network architecture for the policy module and the intrinsic module.

A Architecture Details

We employ two modules, the policy module and the intrinsic module. The overall network architectures for these modules are described in Figure 7 and their details are presented in Table 1 and Table 2. The base architectures of the policy network for MiniGrid and DMLab are based on that of AGAC and IMPALA, respectively. One difference is that the value head of the policy module, which originally outputs the value estimation of extrinsic rewards V^e, estimates the value function for the sum of extrinsic and intrinsic rewards, denoted V^{e+i}. The value estimation for extrinsic rewards is now done in the intrinsic module, as shown in the right hand side of Figure 7.

B Hyperparameters and Training Details

All baseline models as well as LECO are trained using the V-trace actor-critic framework, IMPALA [12], where the policy and the value networks are learned from trajectories collected from multiple actors running asynchronously in a distributed system. The experimentation code for all methods are implemented on top of Sample Factory [27], a code base designed for high throughtput rate during IMPALA-based training. The total number of actors used in MiniGrid tasks and DMLab tasks are 20 and 240, respectively, spread across multiple nodes. Table 3 presents our common hyperparameter values for all models on MiniGrid and DMLab tasks. It is noted that the ObstructedMaze tasks in MiniGrid used $\gamma = 0.8$ while all other tasks, including the DMLab tasks, used $\gamma = 0.99$. We observe that ObstructedMaze tasks, especially the ObstructedMaze-Full is solved with a lower γ value.

For RND, NovelD, and RIDE on DMLab tasks, we search over learning-rate $\in \{0.0001, 0.0002, 0.0004\}$, and the intrinsic reward coefficient $\in \{0.1, 0.01, 0.005\}$ and select the learning rate as 0.0001 and the intrinsic reward coefficient as 0.01.

Table 1: Policy Module

Module	**Layers**	
	MiniGrid	DMLab
Preprocessing	Normalization $\in (-1, 1)$	
ConvBlock(k, c, s)	[Conv2d($k \times k, c$, stride=s)]	
Residual Block(k, c)	[ConvBlock($k \times k, c$, stride=1), ReLU, ConvBlock($k \times k, c$, stride=1), ReLU]	
Encoder	ConvBlock(3,32,2) ELU ConvBlock(3,32,2) ELU ConvBlock(3,32,2) ELU FC $\times$ 1	ConvBlock(3,16,1), MaxPool(3×3,stride=2) ReLU Residual Block(3,16) $\times$ 2 ConvBlock(3,32,1), MaxPool(3×3,stride=2) ReLU Residual Block(3,32) $\times$ 2 ConvBlock(3,32,1), MaxPool(3×3,stride=2) ReLU Residual Block(3,32) $\times$ 2 FC $\times$ 1
Core	LSTM(256) $\times$ 1	LSTM(256+$\lvert\mathcal{A}\rvert$+1) $\times$ 2
Head	Policy: FC(256,$\lvert\mathcal{A}\rvert$) Value: FC(256, 1)	Policy: FC(256+$\lvert\mathcal{A}\rvert$+1,$\lvert\mathcal{A}\rvert$) Value: FC(256+$\lvert\mathcal{A}\rvert$+1, 1)

Table 2: Intrinsic Module

Module	**Layers**	
	MiniGrid	DMLab
Preprocessing	Upsample ($7 \times 7 \rightarrow 12 \times 12$) Downsample ($96 \times 72 \rightarrow 96 \times 64$) Normalization $\in (-1, 1)$	
ConvBlock(k,s)	[Conv2d($k \times k, 64$, stride=s), BatchNorm2d]	
Residual Block	[ReLU, ConvBlock(3, 1), ReLU, ConvBlock(1,1)]	
VQ-Encoder	ConvBlock(3, 1) [ConvBlock(4, 2), ReLU, AvgPool, Conv2d($1 \times 1, 64$, stride=1)] $\times$ 3 ConvBlock(4, 2) Residual Block $\times$ 2	
VQ-Decoder	Symmetric architecture of Encoder	
Modulator	FC(576, 512)	FC(384, 512)
	ReLU FC(512 + $\lvert\mathcal{A}\rvert$, $\lvert\mathcal{A}\rvert$) $\tanh$	
Value Head	FC(576, 512)	FC(384, 512)
	ReLU FC(512, 1)	

C Ablation Study

C.1 Hash size.

In our VQ-based hashing, used in LECO and the baselines, the hash size is defined as the spatial size $w \times h$ of the output of the encoder of VQ-VAE. Accordingly, the capacity of the hash is defined as $K^{w \times h}$ where K is the codebook size. This capacity can be critical in measuring the state novelty based on the episodic count, as it determines the level of the state compression and the corresponding

Table 3: Hyperparameters

Parameter	Value	
	MiniGrid	DMLab
Learning rate (η)	0.0003	0.0001
Learning rate (η_{ta})	0.0003	0.3η
Batch Size	16	48
Entropy Coefficient	0.0005	0.003
Unroll Length	96	
γ	0.99^*	
α	0.01	
λ	0.5	
V-trace ρ	1.0	
V-trace c	1.0	
Optimizer	Adam	
Adam ϵ	1e-6	
Adam β_1	0.9	
Adam β_2	0.999	

Table 4: Hashing parameters

Method	Parameter	Value	
		MiniGrid	DMLab
VQ	spatial size ($w \times h$)	3×3	3×2
	codebook size (K)	8	24
AE-LSH [33]	SimHash dimension	25	62
	$b(s)$ size (bits)	256	512
DSC [10]	spatial size ($w \times h$)	3×3	3×2
	intensity (K)	11	4

state clustering. If the capacity is too large then most states would be mapped to different hashes and if it is too small then most states would be mapped to the same hash.

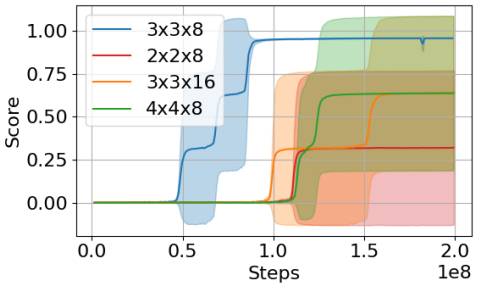
In MiniGrid, we search over the spatial size $\in \{2 \times 2, 3 \times 3, 4 \times 4\}$ and the codebook size $\in \{8, 16\}$. In DMLab, we search over the spatial size $\in \{3 \times 2, 6 \times 4\}$ and the codebook size $\in \{12, 24, 36\}$. The key results of this grid search for LECO are shown in Figure 8. As shows in the figure, $3 \times 3 \times 8$ performs the best in MiniGrid tasks and $3 \times 2 \times 24$ performs the best in the DMLab tasks and thus the two parameter values are selected as the default values for our hashing as in Table 4.

For AE-LSH, we search over the hash size $\in \{9, 15, 25, 35\}$ in MiniGrid, and $\in \{24, 42, 62, 82\}$ in DMLab and select 25 and 62 as the hash size of each task. For a given state, DSC hash is defined as follows: 1) downsample it to $w \times h$ by nearest neighbor algorithm, 2) rescale the intensity so that they are integers between 0 and K for each channel of the state, and 3) serialize it. For DSC in MiniGrid, we search over the downsampled size $\in \{2 \times 2, 3 \times 3, 4 \times 4\}$, and in DMLab we search over the downsampled size $\in \{3 \times 2, 6 \times 4\}$ and the intensity $\in \{4, 8\}$, and select $3 \times 3 \times 11$ and $3 \times 2 \times 4$ as the hash size for MiniGrid and DMLab, respectively.

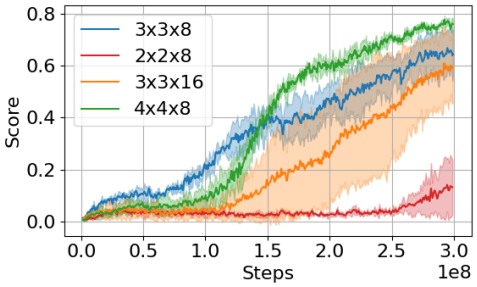
C.2 Study of λ.

Equation 1 indicates the possibility of adjusting the weight to the task-specific modulation. In order to show the effect of this weight, we run an ablation on Equation 1 where the equation is restated here for convenience:

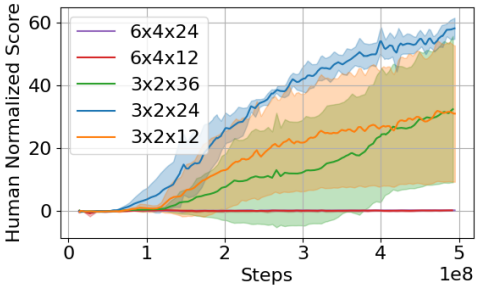
$$r_t^i(a_{t-1}, s_t, a_t, s_{t+1}) = (1 - \lambda) r_t^{\text{ep}}(s_{t+1}) + \lambda r_t^{\text{ta}}(a_{t-1}, s_t, a_t).$$



(a) MiniGrid-KeyCorridor-S6R3



(b) MiniGrid-ObstructedMaze-Full



(c) DMLab-lasertag_three_opponents_large

Figure 8: Performance comparison across different VQ hashing parameters ($w \times h \times K$).

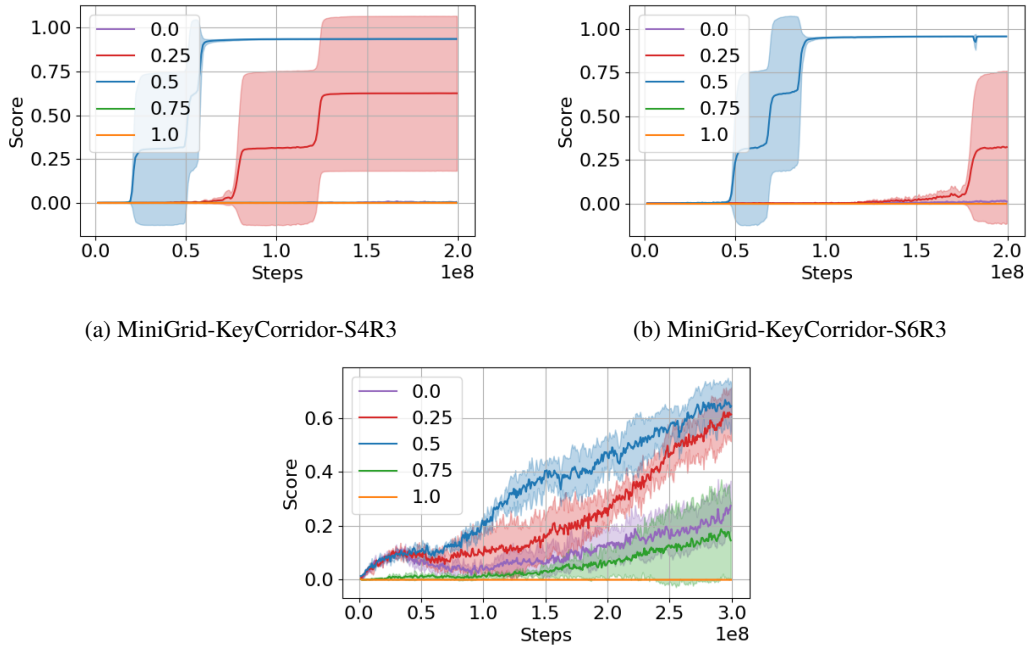
We fix λ as a constant hyperparameter and its value is determined by grid search on three MiniGrid tasks as shown in Figure 9 of Appendix. On the other hand, lambda can be adaptively calculated relative to the magnitude of the novelty from episodic counts, however, we leave it as the future work.

When $\lambda = 0$, the algorithm is equivalent to using episodic count only, which is denoted as VQ-only in the main text. When $\lambda = 1.0$, the algorithm is similar to using LIRPG [40] only, however, note that the architecture of the task-specific modulator are fairly different from the intrinsic module in LIRPG, and the final optimization loss of the intrinsic module in LECO is also different in that it includes a VQ-VAE loss as a part through the shared encoder. When $\lambda = 0.5$, it is equivalent to LECO. As shown in Figure 9, having only either the episodic state novelty or the task-specific modulation results in failure in solving the task. On the other hand, putting more weight on the episodic state novelty ($\lambda = 0.25$) relatively performs better than small weight ($\lambda = 0.75$), however the performance is best when the two terms are balanced ($\lambda = 0.5$).

D Discussions

D.1 Task-irrelevant actions in count-only methods.

In order to solve the KeyCorridor tasks, an agent is required to obtain a key, open a locked door, and obtain the target object, which is a colored ball. Without task-specific modulation the agent can easily get stuck on repeating meaningless actions such as dropping the key or opening and closing the door, to obtain task-irrelevant state novelties. Figure 10 visually depicts such phenomenon. In this figure we provide action heat maps on three distinct episodes. Each map displays the accumulated counts on three specific actions ($pickup, drop, toggle$) from different episodes which are sampled by the final checkpoint of each model. Red colored grids represent the agent's execution of specific action for more than once in that position. For example, in *episode-1* of the top-row, the agent performs the specific action twice below of the gray key. In top-row of the figure we can observe the agent of VQ-only performing the $toggle$ action multiple times in front of the doors and the $pickup$ & $drop$ action repeatedly in empty spaces. Moreover, the agent opens the door to the room with the target object but does not proceed to picking it up (see *Episode-3*). On the other hand, the bottom row of



(a) MiniGrid-KeyCorridor-S4R3

(b) MiniGrid-KeyCorridor-S6R3

(c) MiniGrid-ObstructedMaze-Full

Figure 9: Performances according to different λ's. The labels indicate the normalized λ such that the sum of the weights of the episodic state novelty and task-specific modulation is equal to 1.

the same figure shows that LECO focuses on achieving the task objective by faithfully conducting task-related actions.

D.2 Task-specific modulation in r^i.

The task-specific modulation of LECO is designed to dynamically control exploration and exploitation during training. Here, we conceptually illustrate the trends of LECO's intrinsic reward r^i as the learning progresses. In the beginning of training, before collecting enough extrinsic rewards, the agent would not have any information about task-relevant and -irrelevant behaviors. In hard exploration problems, the agent cannot obtain extrinsic rewards without utilizing the state novelty (i.e. exploration) in the environment. Assuming that exploration by episodic state novelty is enough to guide the agent to achieve some extrinsic rewards, those extrinsic rewards would cause r^{ta} to be positive to encourage the agent to explore more in that direction. As the agent learns explorative behaviors and collects some extrinsic rewards, task-specific modulation will start to distinguish task-relevant from the irrelevant amongst those learned behaviors. Then, the agent transitions to focusing on the task-relevant behaviors and eventually ignore the task-irrelevant behaviors by decaying the intrinsic reward as a whole. The results in Figure 13 depicts the described decaying trend. We can also observe that the dynamics of LECO's intrinsic reward r^i is different for each task; r^i decays more moderately as the relative difficulty and extrinsic reward sparsity increases. The difficulty and sparsity roughly increases from (a)→(c), (d)→(f) and from (g)→(h).

D.3 Comparisons of hashing results.

First, We qualitatively show some weaknesses of AE-LSH compared to VQ. In general, both hash methods map similar states into the same hash, however, some differences can be observed in the obtained hashing results. As shown in Figure 14 and Figure 15, in Minigrid, AE-LSH maps important states (i.e. states in which the agent realizes a target object or states just before obtaining a target object) and relatively less important states into the same hash. This obstructs correctly distinguishing task-specific state novelty for efficient exploration. On the other hand, VQ more clearly clusters states to the hash according to the relative importance.

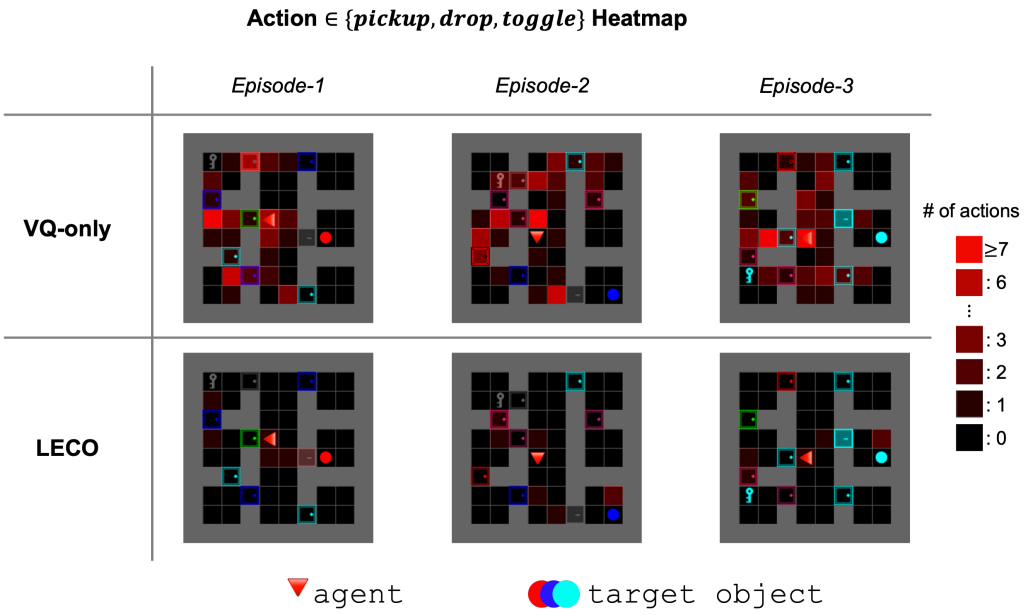


Figure 10: Action heat maps of three different episodes on the task of `KeyCorridor-S4R3`, Count only based, VQ-only, (**Top-Row**) and LECO (**Bottom-Row**). The red triangle is added to indicate the initial position of the agent. The ball, whose color is randomly chosen at the start of each episode, is the target object that the agent has to pickup. Unlike KeyCorridor, for the tasks of ObstructedMaze family, the goal object is the blue ball, whose color stays consistent.

In DMLab-lasertags, the difference in performance between AE-LSH and VQ is also reflected in the hashing results. As shown in Figure 16, Figure 17, and Figure 18, AE-LSH maps states to hashes according to visual similarities whereas VQ learns to map according to the importance represented via the presence of the opponents in sight.

Table 5: Mean *new-hash rates* on five episodes sampled on DMLab-lasertag.

`three_opponents_small`		`three_opponents_large`	
LECO(AE-LSH)	LECO(VQ)	LECO(AE-LSH)	LECO(VQ)
86.8%	71.6%	68.2%	58.6%

We quantitatively show the effectiveness of VQ compared to AE-LSH. As shown in Table 5, on the DMLab-Lasertags, the new hash rates, which measure how often the states are mapped into a new hash within an episode, obtained by VQ are smaller than those by AE-LSH even though the entire hash space of VQ is larger than AE-LSH (i.e. $6^{24} > 2^{62}$). This means that VQ makes similar states to be more grouped into the same hash.

D.4 Noisy-TV on MiniGrid.

Provided by [28] is a variation of MultiRoom task that brings the concept of Noisy-TV problem into the MiniGrid environment setting, named `MiniGrid-MultiRoomNoisyTV-N7-S4`. In this version of the task, the color of the ball changes to a random color when the agent performs a specific action, creating a visual novelty irrelevant to the objective of the task, which is to reach the goal [28]. In Figure 11, we present the learning curves of LECO and baselines on this Noisy-TV environment. The result shows that all methods that use the episodic intrinsic reward, including LECO, are able to successfully solve the noisy environments. In this experiment, We conjecture that the task-specific modulator in LECO requires a little more experiences than NovelD since it is updated upon the extrinsic rewards that are sparse in the early RL phase. In a similar argument, DSC, which requires

no training, is most efficient in the noisy TV problem. Moreover, in fact NovelD converges stably in about 3e7 frames (vs. 5e7 frames for LECO).

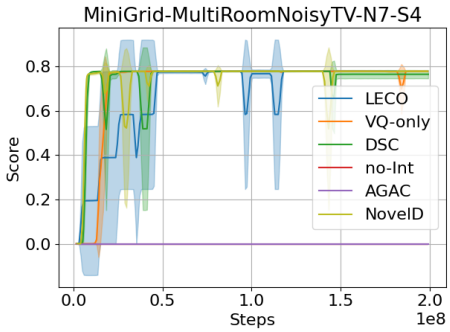


Figure 11: Performance comparison on Noisy-TV environment. Along with its variants, LECO is able to bypass the Noisy-TV problem. The shaded area represents a range of a standard deviation over 3 runs of different random seeds.

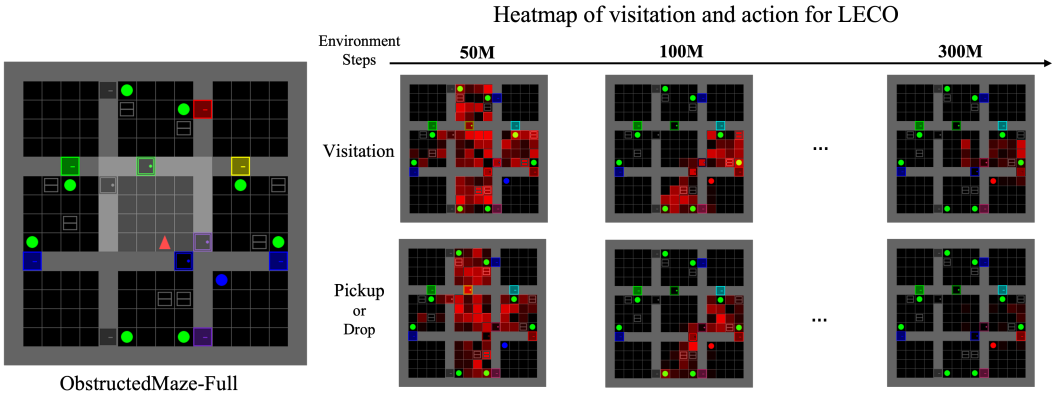
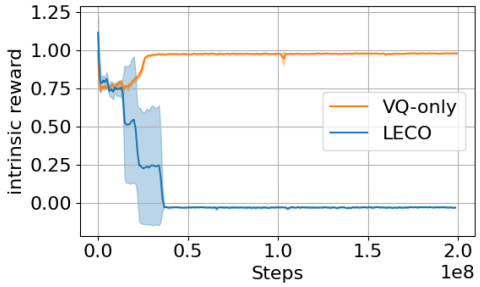
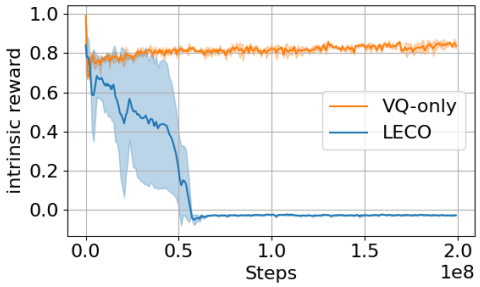


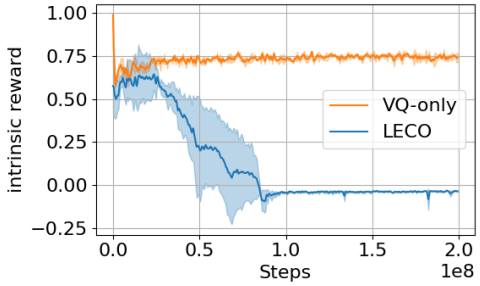
Figure 12: `ObstructedMaze-Full` Environment (**Left**), Visitation heatmaps (**Right Top**) and Action heatmaps (**Right Bottom**): LECO successfully solves this task from 100M steps, and does not repeat meaningless pickup & drop actions to obtain the state novelty at 300M steps.



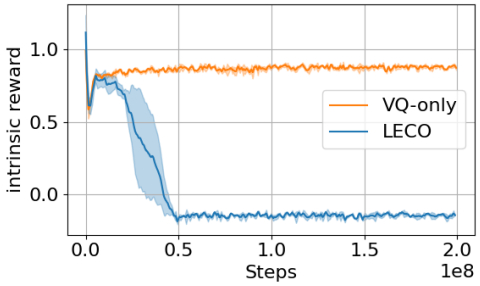
(a) MiniGrid-MultiRoom-N6



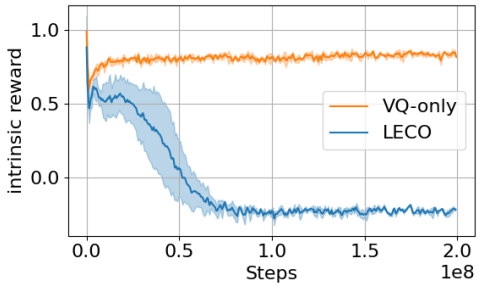
(b) MiniGrid-KeyCorridor-S4R3



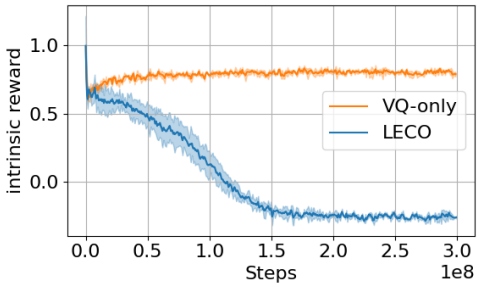
(c) MiniGrid-KeyCorridor-S6R3



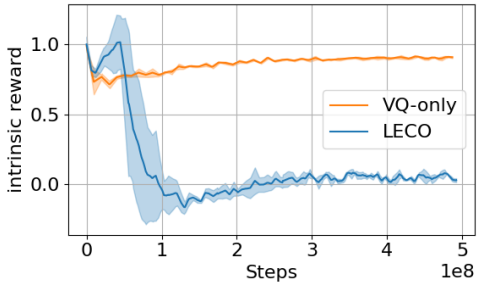
(d) MiniGrid-ObstructedMaze-1Q



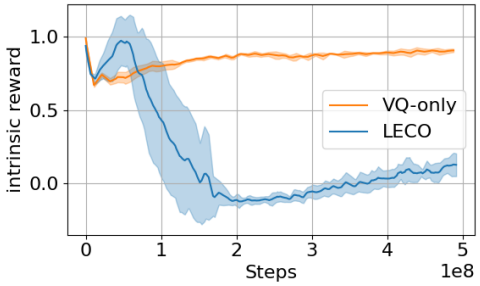
(e) MiniGrid-ObstructedMaze-2Q



(f) MiniGrid-ObstructedMaze-Full



(g) DMLab-lasertag_three_opponents_small



(h) DMLab-lasertag_three_opponents_large

Figure 13: Change in r^i for different tasks during training.

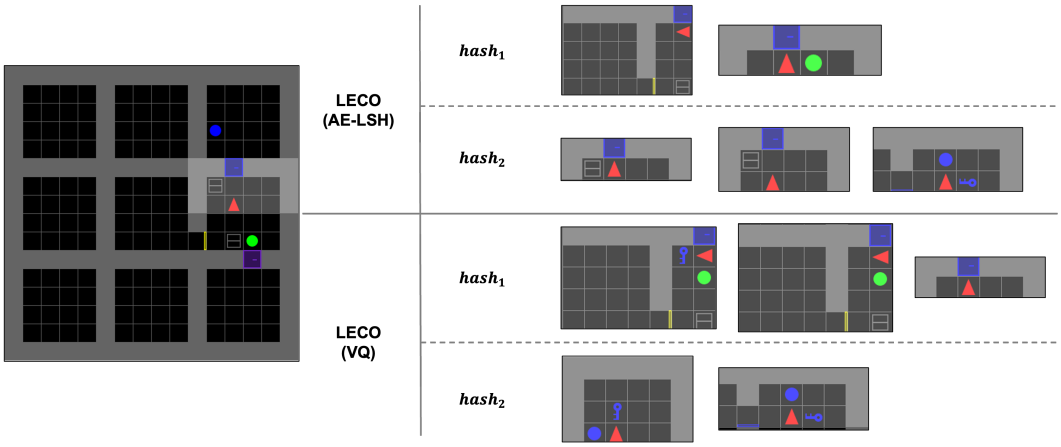


Figure 14: (**Left:**) An episode of `ObstructedMaze-1Q` Environment. The bright area is the partial observation. In this episode, the agent must find a blue key, open the locked blue door, and obtain the blue ball. (**Right:**) Hash samples from the episodic hash memory of LECOs. Each row represents a hash index and the partially observed states that are mapped into this hash.

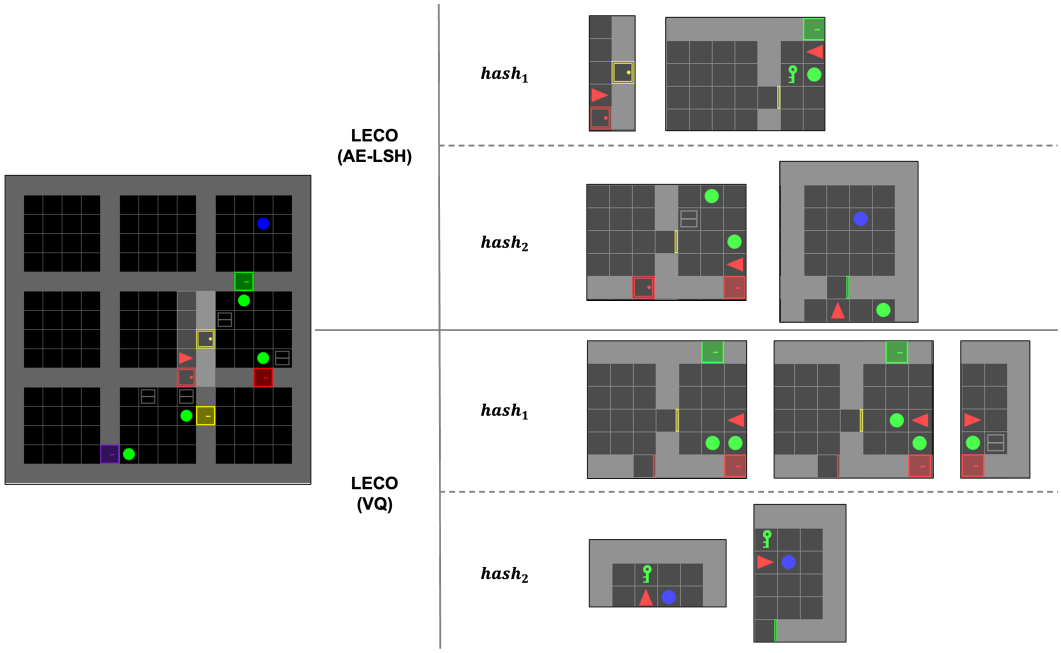


Figure 15: (**Left:**) An episode of `ObstructedMaze-2Q` Environment. The bright area is the partial observation. In this episode, the agent must find a green key, open the locked green door, and obtain the blue ball. (**Right:**) Hash samples from the episodic hash memory of LECOs. Each row represents a hash index and the partially observed states that are mapped into this hash.

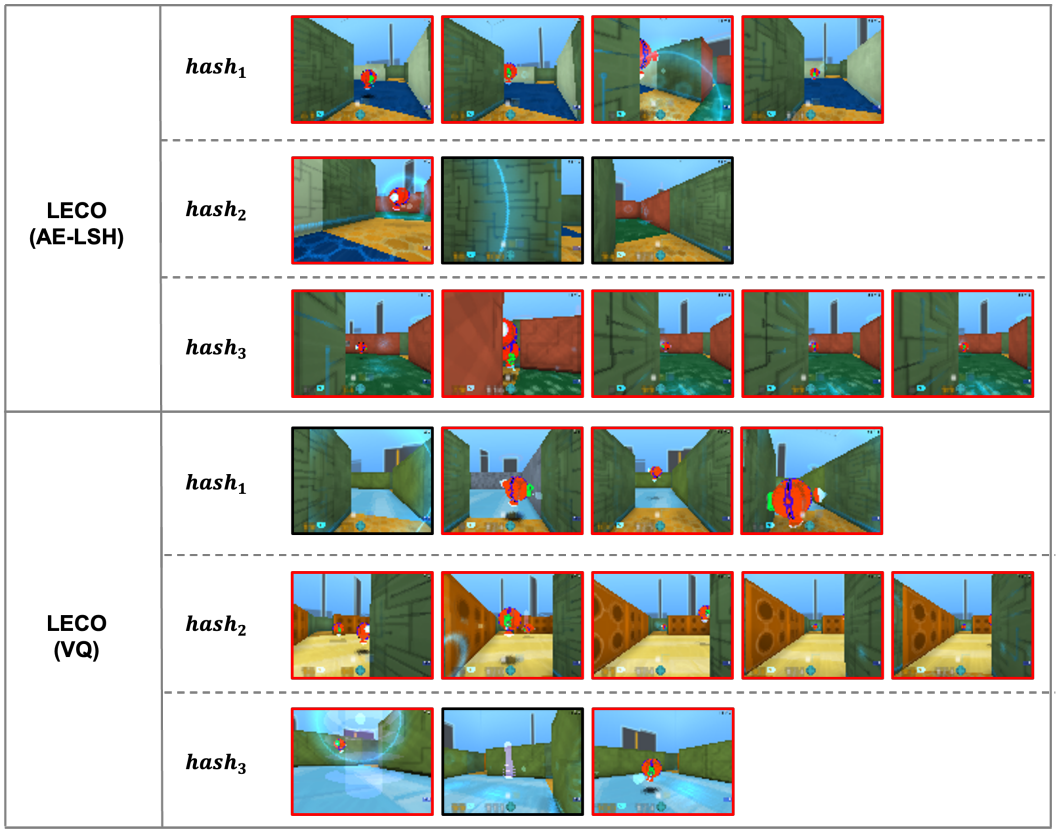


Figure 16: Hash samples of the episodic hash memory of LECOs on DMLab-lasertag_three_opponents_small. Each row represents a hash index and the partially observed states that are mapped into this hash. The state that contained the opponents is highlighted in red.

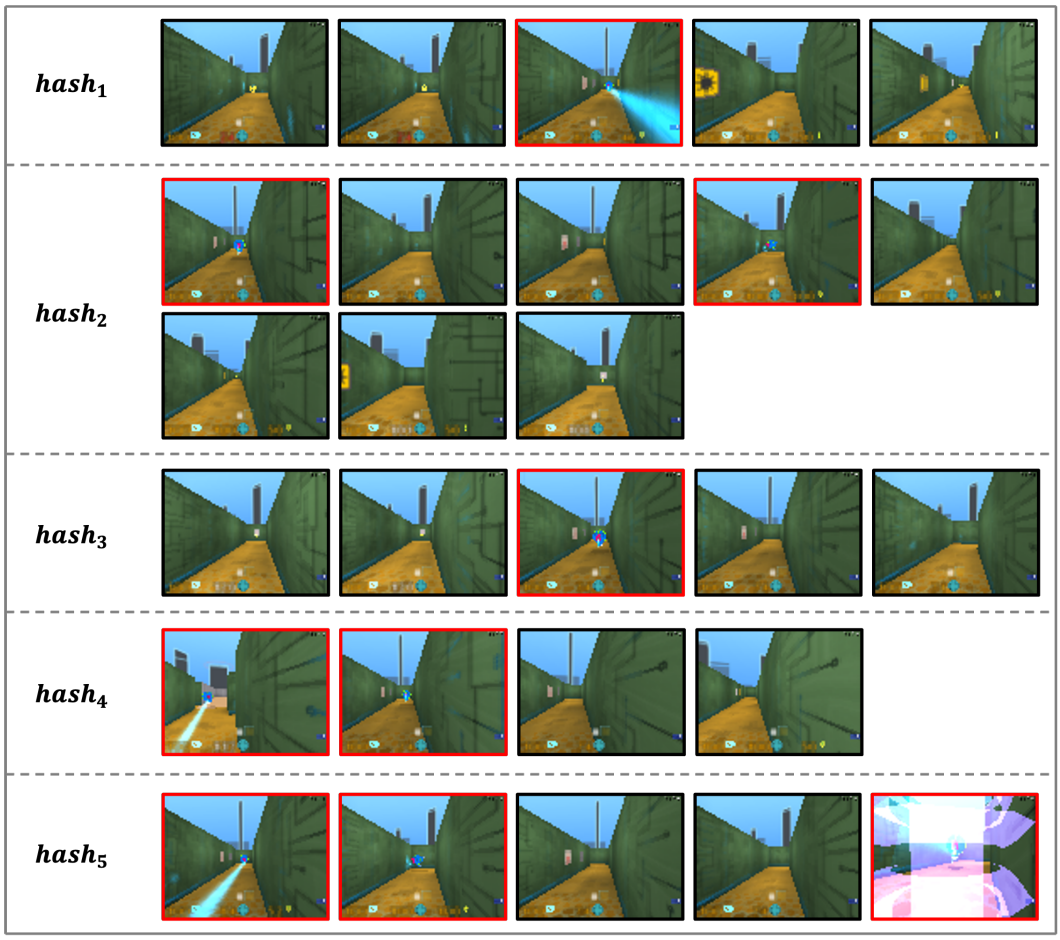


Figure 17: Hash samples of the episodic hash memory of LECO(AE-LSH) on DMLab-lasertag_three_opponents_large. Each row represents a hash index and the partially observed states that are mapped into this hash. The state that contained the opponents is highlighted in red.

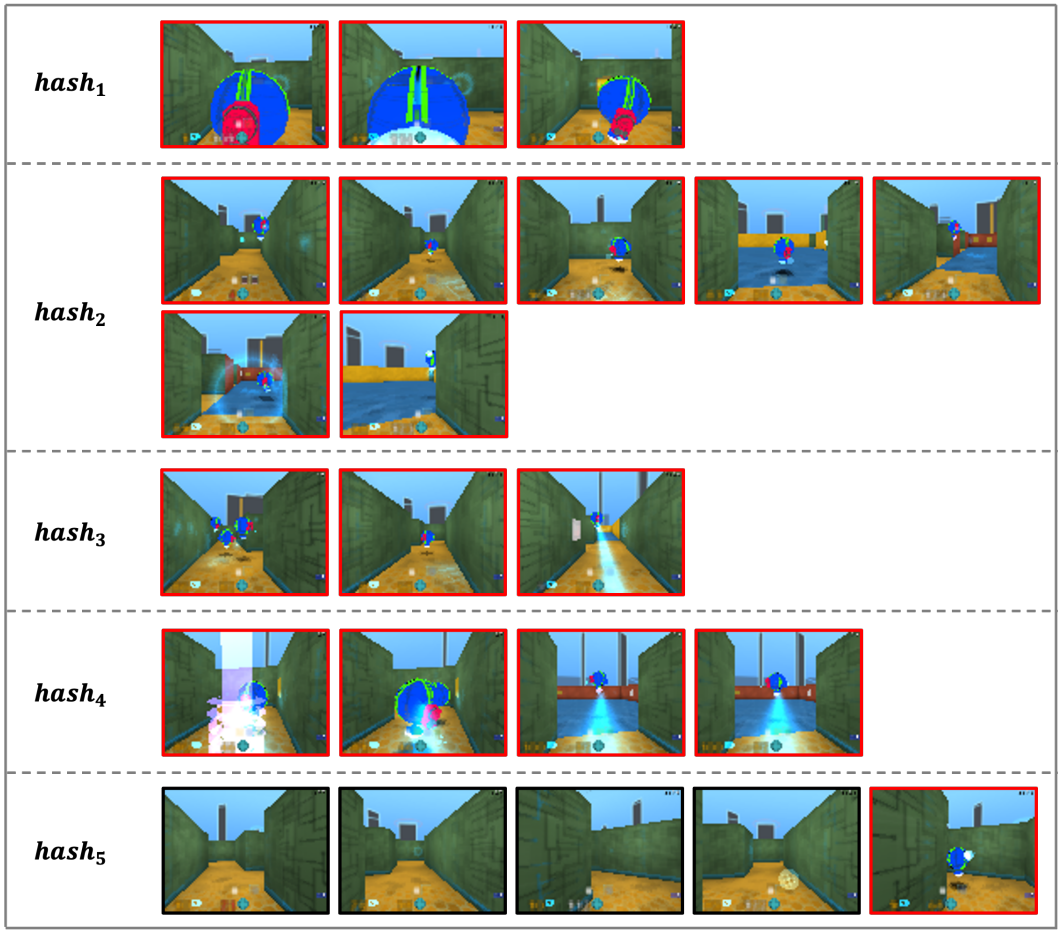


Figure 18: Hash samples of the episodic hash memory of LECO(VQ) on DMLab-lasertag_three_opponents_large. Each row represents a hash index and the partially observed states that are mapped into this hash. The state that contained the opponents is highlighted in red.