Discovering Hierarchical Achievements in Reinforcement Learning via Contrastive Learning

Seungyong Moon[1], **Junyoung Yeom**[1], **Bumsoo Park**[2], **Hyun Oh Song**[1]*
[1]Seoul National University, [2]KRAFTON
{symoon11,yeomjy,hyunoh}@mllab.snu.ac.kr
bumsoo.park96@krafton.com

Abstract

Discovering achievements with a hierarchical structure in procedurally generated environments presents a significant challenge. This requires an agent to possess a broad range of abilities, including generalization and long-term reasoning. Many prior methods have been built upon model-based or hierarchical approaches, with the belief that an explicit module for long-term planning would be advantageous for learning hierarchical dependencies. However, these methods demand an excessive number of environment interactions or large model sizes, limiting their practicality. In this work, we demonstrate that proximal policy optimization (PPO), a simple yet versatile model-free algorithm, outperforms previous methods when optimized with recent implementation practices. Moreover, we find that the PPO agent can predict the next achievement to be unlocked to some extent, albeit with limited confidence. Based on this observation, we introduce a novel contrastive learning method, called *achievement distillation*, which strengthens the agent's ability to predict the next achievement. Our method exhibits a strong capacity for discovering hierarchical achievements and shows state-of-the-art performance on the challenging Crafter environment in a sample-efficient manner while utilizing fewer model parameters.

1 Introduction

Deep reinforcement learning (RL) has recently achieved remarkable successes in solving challenging decision-making problems, including video games, board games, and robotic controls [35, 50, 18, 44]. However, these advancements are often restricted to a single deterministic environment with a narrow set of tasks. To successfully deploy RL agents in real-world scenarios, which are constantly changing and open-ended, they should generalize well to new unseen situations and acquire reusable skills for solving increasingly complex tasks via long-term reasoning. Unfortunately, many existing algorithms exhibit limitations in learning these abilities and tend to memorize action sequences rather than truly understand the underlying structures of the environments [32, 27].

To assess the abilities of agents in generalization and long-term reasoning, we focus on the problem of discovering hierarchical achievements in procedurally generated environments with high-dimensional image observations. In each episode, an agent navigates a previously unseen environment and receives a sparse reward upon accomplishing a novel subtask labeled as an *achievement*. Importantly, each achievement is semantically meaningful and can be reused to complete more complex achievements. Such a setting inherently demands strong generalization and long-term reasoning from the agent.

Previous work on this problem has mainly relied on model-based or hierarchical approaches, which involve explicit modules for long-term planning. Model-based methods employ a latent world model that predicts future states and rewards for learning long-term dependencies [20, 21, 1, 53]. While

*Corresponding author

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

these methods have shown effectiveness in discovering hierarchical achievements, particularly in procedurally generated environments, they are constructed with large model sizes and often require substantial exploratory data, which limits their practicality. Hierarchical methods aim to reconstruct the dependencies between achievements as a graph and employ a high-level planner on the graph to direct a low-level controller toward the next achievement to be unlocked [51, 10, 57]. However, these methods rely on prior knowledge of achievements (*e.g.*, the number of achievements), which is impractical in open-world scenarios where the exact number of achievements cannot be predetermined. Additionally, they necessitate a significant number of offline expert data to reconstruct the graph.

To address these issues, we begin by exploring the potential of proximal policy optimization (PPO), a simple and flexible model-free algorithm, in discovering hierarchical achievements [47]. Surprisingly, PPO outperforms previous model-based and hierarchical methods by adopting recent implementing practices. Furthermore, upon analyzing the latent representations of the PPO agent, we observe that it has a certain degree of predictive ability regarding the next achievement, albeit with high uncertainty.

Based on this observation, we propose a novel self-supervised learning method alongside RL training, named *achievement distillation*. Our method periodically distills relevant information on achievements from episodes collected during policy updates to the encoder via contrastive learning [40]. Specifically, we maximize the similarity in the latent space between state-action pairs and the corresponding next achievements within a single episode. Additionally, by leveraging the uniform achievement structure across all environments, we maximize the similarity in the latent space between achievements from two different episodes, matching them using optimal transport [5]. This learning can be seamlessly integrated into PPO by introducing an auxiliary training phase. Our method demonstrates state-of-the-art performance in discovering hierarchical achievements on the challenging Crafter benchmark, unlocking all 22 achievements with a budget of 1M environment steps while utilizing only 4% of the model parameters compared to the previous state-of-the-art method [19].

2 Preliminaries

2.1 Markov decision processes with hierarchical achievements

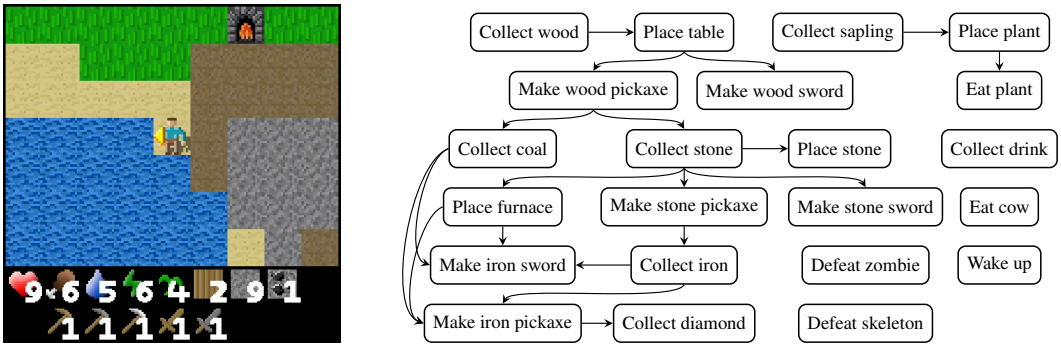
We formalize the problem using Markov decision processes (MDPs) with hierarchical achievements [57]. Let $\mathbb{M}$ represent a collection of such MDPs. Each environment $\mathcal{M}_i \in \mathbb{M}$ is defined by a tuple $(\mathcal{S}_i, \mathcal{A}, \mathcal{G}, p, r, \rho_i, \gamma)$. Here, $\mathcal{S}_i \subset \mathcal{S}$ is the image observation space, which has visual variations across different environments, $\mathcal{A}$ is the action space, $\mathcal{G}$ is the achievement graph with a hierarchical structure, $p : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ is the transition probability function, $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the achievement reward function, $\rho_i \in \mathcal{P}(\mathcal{S}_i)$ is the initial state distribution, and $\gamma \in [0, 1]$ is the discount factor.

The achievement graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a directed acyclic graph, where each vertex $v \in \mathcal{V}$ represents an achievement and each edge $(u, v) \in \mathcal{E}$ indicates that achievement v has a dependency on achievement u. To unlock achievement v, all of its ancestors (*i.e.*, achievements in the path from the root to v) must also be unlocked. When an agent unlocks a new achievement, it receives an achievement reward of 1. Note that each achievement can be accomplished multiple times within a single episode, but the agent will only receive a reward when unlocking it for the first time. Specifically, let $b \in \{0, 1\}^{|\mathcal{V}|}$ be a binary vector indicating which achievements have been unlocked and $c : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{V} \cup \{\emptyset\}$ be a function determining whether a transition tuple results in the completion of an achievement. Then, the achievement reward function is defined as

$$r(s_t, a_t, s_{t+1}) = \begin{cases} 1 & \text{if } \exists v_i \in \mathcal{V} : b[i] = 0, c(s_t, a_t, s_{t+1}) = v_i \\ 0 & \text{otherwise.} \end{cases}$$

This reward structure provides an incentive for the agent to explore the environments and discover a new achievement, rather than repeatedly accomplishing the same achievements.

We assume that the agent has no prior knowledge of the achievement graph, including the number of achievements and their dependencies. Additionally, the agent has no direct access to information about which achievements have been unlocked. Instead, the agent must infer this information indirectly from the reward signal it receives. Given this situation, our objective is to learn a generalizable policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ that maximizes the expected return (*i.e.*, unlocks as many achievements as possible) across all environments of $\mathbb{M}$.



(a) Agent's observation

(b) Achievement graph with a hierarchical structure.

Figure 1: Overview of Crafter. Each environment is procedurally generated and partially observable. The objective is to unlock as many achievements as possible within a single episode.

2.2 Crafter environment

We primarily utilize the Crafter environment as a benchmark to assess the capabilities of an agent in solving MDPs with hierarchical achievements [19]. Crafter is an open-world survival game with 2D visual inputs, drawing inspiration from the popular 3D game Minecraft [17]. This is optimized for research purposes, with fast and straightforward environment interactions and clear evaluation metrics. The game consists of procedurally generated environments with varying world map layouts, terrain types, resource placements, and enemy spawn locations, each of which is uniquely determined by an integer seed. An agent can only observe its immediate surroundings as depicted in Figure 1a, which makes Crafter partially observable and thus challenging. To survive, the agent must acquire a variety of skills, including exploring the world map, gathering resources, building tools, placing objects, and defending against enemies. The game features a set of 22 hierarchical achievements that the agent can unlock by completing specific prerequisites, as illustrated in Figure 1b. For instance, to make a wood pickaxe, the agent needs to collect wood, place a table, and stand nearby. This achievement structure is designed to require the agent to learn and utilize a wide range of skills to accomplish increasingly challenging achievements, such as crafting iron tools and collecting diamonds.

2.3 Proximal policy optimization

PPO is one of the most successful model-free policy gradient algorithms due to its simplicity and effectiveness [47]. PPO learns a policy $\pi_\theta : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ and a value function $V_\theta : \mathcal{S} \to \mathbb{R}$, which are parameterized by neural networks. During training, PPO first collects a new episodes $\mathcal{T}$ using the policy $\pi_{\theta_{\text{old}}}$ immediately prior to the update step. Subsequently, PPO updates the policy network using these episodes for several epochs to maximize the clipped surrogate policy objectives given by

$$J_\pi(\theta) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{T}}\left[\min\left(\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t, \text{clip}\left(\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}, 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right],$$

where $\hat{A}_t$ is the estimated advantage computed by generalized advantage estimate (GAE) [46]. PPO simultaneously updates the value network to minimize the value objective given by

$$J_V(\theta) = \mathbb{E}_{s_t\sim\mathcal{T}}\left[\frac{1}{2}\left(V_\theta(s_t) - \hat{V}_t\right)^2\right],$$

where $\hat{V}_t = \hat{A}_t + V_{\theta_{\text{old}}}(s_t)$ is the bootstrapped value function target.

In image-based RL, it is common practice to optimize the policy and value networks using a shared network architecture [13, 56]. An image observation is first passed through a convolutional encoder $\phi_\theta : \mathcal{S} \to \mathbb{R}^h$ to extract a state representation, which is then fed into linear heads to compute the policy and value function. Sharing state representations between the policy and value networks is crucial to improving the performance of agents in high-dimensional state spaces. However, relying solely on policy and value optimization to train the encoder can lead to suboptimal state representations, particularly in procedurally generated environments [9]. To address this issue, recent studies introduce an auxiliary training phase alongside the policy and value optimization that trains the encoder with auxiliary value or self-supervised objectives [9, 36].

3 Motivation

3.1 PPO is a strong baseline for hierarchical achievements

Despite being a simple and ubiquitous algorithm, PPO is less utilized than model-based or hierarchical approaches for solving MDPs with hierarchical achievements. This is due to the fact that PPO does not have an explicit component for long-term planning or reasoning, which is believed to be essential for solving hierarchical tasks. However, a recent study has shown that a PPO-based algorithm is also successful in solving hierarchical achievement on the Minecraft environment, albeit with the aid of pre-training on human video data [4].

Based on this observation, we first investigate the effectiveness of PPO in solving hierarchical achievements on Crafter without pre-training. We adopt the recent implementation practices proposed in Andrychowicz et al. [2], Baker et al. [4]. Concretely, we modify the default ResNet architecture in IMPALA as follows [14]:

- **Network size**: We increase the channel size from [16, 32, 32] to [64, 128, 128] and the hidden size from 256 to 1024.
- **Layer normalization**: We add layer normalization before each dense or convolutional layer [3].
- **Value normalization**: We keep a moving average for the mean and standard deviation of the value function targets and update the value network to predict the normalized targets.

We train the modified PPO on Crafter for 1M environment steps and evaluate the success rates for unlocking achievements (please refer to Section 5.1 for the evaluation). Figure 2 shows that the slight modification in implementing PPO significantly improves the performance, increasing the score from 8.17 to 15.60. Notably, this outperforms the current state-of-the-art DreamerV3, which achieves a score of 14.77.

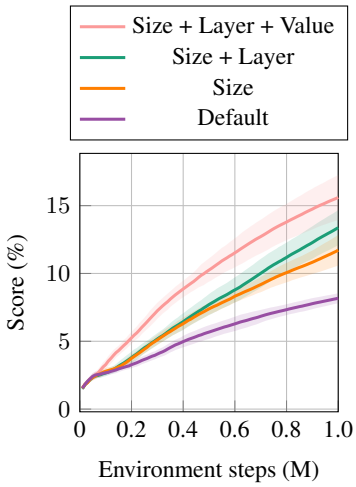


Figure 2: Score curves of PPO.

3.2 Representation analysis of PPO

Since Crafter environments are procedurally generated, simply memorizing successful episodes is insufficient for achieving high performance. We hypothesize that the PPO agent acquires knowledge beyond mere memorization of action sequences, possibly including information about achievements. To validate this, we analyze the learned latent representations of the encoder, as inspired by Wijmans et al. [54]. Specifically, we collect a batch of episodes using an expert policy and subsample a set of states for training. For each state s in the training set, we freeze its latent representation $\phi_\theta(s)$ from the encoder. Subsequently, we train a linear classifier using this representation as input to predict the very next achievement unlocked in the episode containing s. Finally, we evaluate the classification accuracy and the prediction confidence of the ground-truth labels on a held-out test set. The detailed experimental settings are provided in Appendix A.

Surprisingly, PPO achieves a nontrivial accuracy of 44.9% in the 22-way classification. However, Figure 3 shows that the prediction outputs lack confidence with a median value of 0.240. This suggests that the learned representations of the PPO encoder are not strongly correlated with the next achievement to be unlocked and the agent may struggle to generate optimal action sequences towards a specific goal.

This finding warrants providing additional guidance to the encoder for predicting the next achievements with high confidence. However, since the agent has no access to the achievement labels, it is challenging to guide the agent in a supervised fashion. Therefore, it is necessary to explore alternative approaches to guide the agent toward predicting the next achievements.

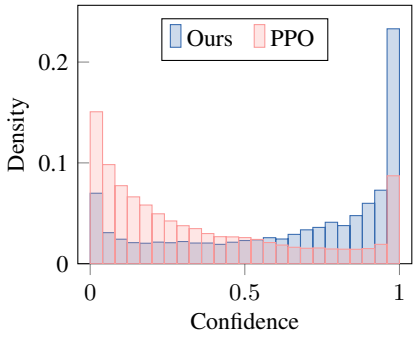


Figure 3: Histogram for the confidence of next achievement prediction.

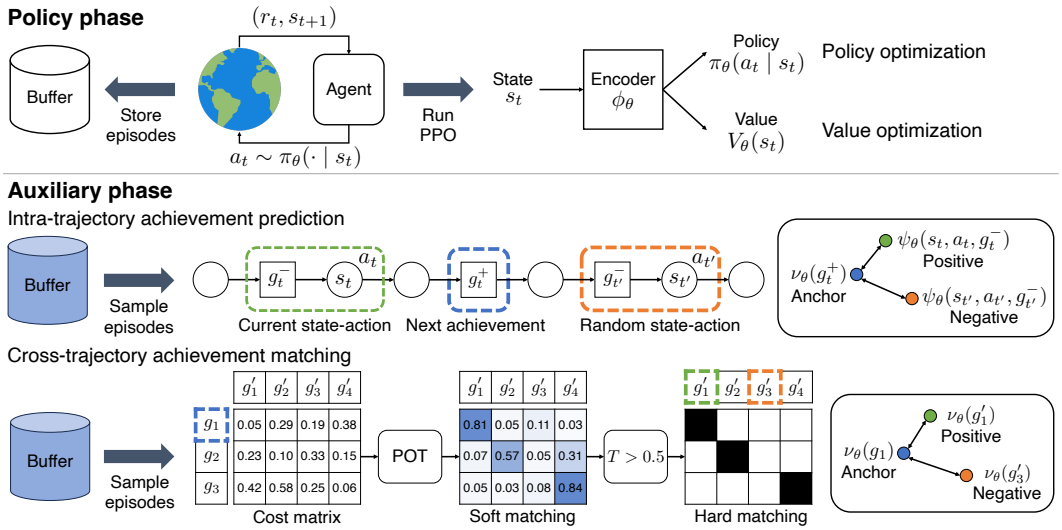


Figure 4: Illustration of achievement distillation.

4 Contrastive learning for achievement distillation

In this section, we introduce a new self-supervised learning method that works alongside RL training to guide the encoder in predicting the next achievement to be unlocked. This approach distills relevant information about discovered achievements from episodes collected during multiple policy updates into the encoder via contrastive learning. This method consists of two key components:

- **Intra-trajectory achievement prediction**: Within an episode, this maximizes the similarity in the latent space between a state-action pair and its corresponding next achievement.
- **Cross-trajectory achievement matching**: Between episodes, this maximizes the similarity in the latent space for matched achievements.

For ease of notation, we denote the sequence of unlocked achievements within an episode as $(g_i)_{i=1}^m$ and their corresponding timesteps as $(t_i)_{i=1}^m$, where each achievement g_i is defined by a transition tuple $(s_{t_i}, a_{t_i}, s_{t_i+1})$. For each timestep t, we represent the very next achievement as $g_t^+ = g_u$, where $u = \min\{i \mid t \leq t_i\}$, and the very previous achievement as $g_t^- = g_l$, where $l = \max\{i \mid t > t_i\}$.

4.1 Intra-trajectory achievement prediction

Given a state-action pair (s_t, a_t) and its corresponding next achievement g_t^+ within an episode τ, we train the encoder ϕ_θ to produce similar representations for them through contrastive learning [40, 24]. Specifically, we regard g_t^+ as the anchor and (s_t, a_t) as the positive. We also randomly sample another state-action pair $(s_{t'}, a_{t'})$ from the same episode to serve as the negative. Subsequently, we obtain the normalized representations of the anchor, positive, and negative, denoted as $\nu_\theta(g_t^+)$, $\psi_\theta(s_t, a_t)$, and $\psi_\theta(s_{t'}, a_{t'})$, respectively. Finally, we minimize the following contrastive loss to maximize the cosine similarity between the anchor and positive representations while minimizing the cosine similarity between the anchor and negative representations:

$$L_{\text{pred}}(\theta) = -\mathbb{E}_{\substack{(s_t,a_t)\sim\tau \\ (s_{t'},a_{t'})\sim\tau}} \left[\log \left(\frac{\exp(\psi_\theta(s_t, a_t)^\top \nu_\theta(g_t^+)/\lambda)}{\exp(\psi_\theta(s_t, a_t)^\top \nu_\theta(g_t^+)/\lambda) + \exp(\psi_\theta(s_{t'}, a_{t'})^\top \nu_\theta(g_t^+)/\lambda)} \right) \right],$$

where $\lambda > 0$ is the temperature parameter.

To obtain the state-action representation $\psi_\theta(s_t, a_t)$, we calculate the latent representation of the state $\phi_\theta(s_t)$ from the encoder and concatenate it with the action a_t using a FiLM layer [42]. The resulting vector is then passed through an MLP layer and normalized. To obtain the achievement representation $\nu_\theta(g_t^+)$, we simply calculate the residual of the latent representations of the two consecutive states from the encoder and normalize it, as motivated by Nair et al. [38].

While this contrastive objective encourages the encoder to predict the next achievement in the latent space, it can potentially lead to distortions in the policy and value networks due to the changes in the encoder. To address this, we jointly minimize the following regularizers to preserve the outputs of the policy and value networks, following the practice in Moon et al. [36]:

$$R_\pi(\theta) = \mathbb{E}_{s_t \sim \tau}\left[D_{\mathrm{KL}}\left(\pi_{\theta_{\mathrm{old}}}(\cdot \mid s_t) \parallel \pi_\theta(\cdot \mid s_t)\right)\right], \quad R_V(\theta) = \mathbb{E}_{s_t \sim \tau}\left[\frac{1}{2}\left(V_\theta(s_t) - V_{\theta_{\mathrm{old}}}(s_t)\right)^2\right],$$

where $\pi_{\theta_{\mathrm{old}}}$ and $V_{\theta_{\mathrm{old}}}$ are the policy and value networks immediately prior to the contrastive learning, respectively and D_{KL} denotes the KL divergence.

4.2 Cross-trajectory achievement matching

Since Crafter environments are procedurally generated, the achievement representations learned solely from intra-trajectory information may include environment-specific features that limit generalization. To obtain better achievement representations, we leverage the common achievement structure shared across all episodes.

We first match the sequences of unlocked achievements from two different episodes in an unsupervised fashion. Given two achievement sequences $\mathbf{g} = (g_i)_{i=1}^m$ and $\mathbf{g}' = (g'_j)_{j=1}^n$, we define the cost matrix $M \in \mathbb{R}^{m \times n}$ as the cosine distance between the achievement representations:

$$M_{ij} = 1 - \nu_\theta(g_i)^\top \nu_\theta(g'_j).$$

Subsequently, we regard these two sequences as discrete uniform distributions and compute a soft-matching $T \in \mathbb{R}^{m \times n}$ between them using partial optimal transport, which can be solved by

$$T = \underset{T \geq 0}{\arg\min} \langle T, M \rangle + \alpha \sum_{i=1}^{m} \sum_{j=1}^{n} T_{ij} \log T_{ij}$$
$$\text{subject to } T\mathbf{1} \leq \mathbf{1},\ T^\top \mathbf{1} \leq \mathbf{1},\ \mathbf{1}^\top T^\top \mathbf{1} = \min\{m, n\},$$

where $\alpha > 0$ is the entropic regularization parameter [5]. Here, we set the total amount of probability mass to be transported to the minimum length of the two sequences for simplicity. However, some unlocked achievements in one sequence may not exist in the other sequence, and therefore should not be transported. In this case, the optimal amount of probability mass to be transported should be less than the minimum length. To address this, we compute the conservative hard matching T^* from T by thresholding the probabilities less than 0.5 (*i.e.*, $T^* = \mathbb{1}[T > 0.5]$). Note that this also encourages each achievement to be matched to at most one other achievement. We provide examples of matching results in Appendix B.

We train the encoder to produce similar representations for the matched achievements according to T^* through contrastive learning. Specifically, suppose that the ith achievement of the source sequence g_i is matched with the kth achievement of the target sequence g'_k. Then, we consider g_i as the anchor and g'_k as the positive. We also randomly sample another achievement g'_j from the target sequence to serve as the negative. Subsequently, we obtain the normalized representations of these achievements. Finally, we minimize the following contrastive loss to maximize the cosine similarity between the anchor and positive representations while minimizing the cosine similarity between the anchor and negative representations:

$$L_{\mathrm{match}}(\theta) = -\mathbb{E}_{g_i \sim \mathbf{g}, g'_j \sim \mathbf{g}'}\left[\log\left(\frac{\exp(\nu_\theta(g_i)^\top \nu_\theta(g'_k)/\lambda)}{\exp(\nu_\theta(g_i)^\top \nu_\theta(g'_k)/\lambda) + \exp(\nu_\theta(g_i)^\top \nu_\theta(g'_j)/\lambda)}\right)\right],$$

As in Section 4.1, we jointly minimize the policy and value regularizers to prevent distortions.

4.3 Achievement representation as memory

We further utilize the achievement representations learned in Sections 4.1 and 4.2 as memory for the policy and value networks. Specifically, given a state s_t and its corresponding previous achievement g_{t^-}, we concatenate the latent state representation $\phi_\theta(s_t)$ with the previous achievement representation $\nu_\theta(g_{t^-})$. The resulting vector is then fed into the policy and value heads to output an action distribution and value estimate, respectively.

We also utilize the previous achievement for the achievement prediction task in Section 4.1. Given a state-action pair (s_t, a_t) and its corresponding previous achievement g_t^-, we concatenate the latent state representation $\phi_\theta(s_t)$ from the encoder with a_t and $\nu_\theta(g_t^-)$. The resulting vector is then fed into an MLP layer and normalized to obtain the representation $\psi_\theta(s_t, a_t, g_t^-)$ for the next achievement prediction. This is interpreted as learning forward dynamics in the achievement space.

4.4 Integration with RL training

We integrate the contrastive learning method proposed in Sections 4.1 to 4.3 with PPO training by introducing two alternating phases, the policy and auxiliary phases. During the policy phase, which is repeated several times, we update the policy and value networks using newly-collected episodes and store them in a buffer. During the auxiliary phase, we update the encoder to optimize the contrastive objectives in conjunction with the policy and value regularizers using all episodes in the buffer. We call this auxiliary learning *achievement distillation*. The illustration and pseudocode are presented in Figure 4 and Algorithm 1, respectively.

Algorithm 1 PPO with achievement distillation

Require: Policy network π_θ, value network V_θ
for phase = $1, 2, \dots$ **do**
  Reset the buffer $\mathcal{B}$
  for iter = $1, 2, \dots, N_\pi$ **do** ▷ PPO training
    Collect episodes $\mathcal{T}$ using π_θ and add them to $\mathcal{B}$
    for epoch = $1, 2, \dots, E_\pi$ **do**
      Optimize $J_\pi(\theta)$ and $J_V(\theta)$ using $\mathcal{T}$
    end for
  end for
  $\pi_{\theta_{\text{old}}} \leftarrow \pi_\theta$, $V_{\theta_{\text{old}}} \leftarrow V_\theta$
  for iter = $1, 2, \dots, E_{\text{aux}}$ **do** ▷ Achievement distillation
    Optimize $L_{\text{pred}}(\theta)$, $R_\pi(\theta)$, and $R_V(\theta)$ using $\mathcal{B}$
    Optimize $L_{\text{match}}(\theta)$, $R_\pi(\theta)$, and $R_V(\theta)$ using $\mathcal{B}$
  end for
end for

5 Experiments

5.1 Experimental setup

To assess the effectiveness of our method in discovering hierarchical achievements, we train the agent on Crafter for 1M environment steps and evaluate its performance, following the protocol in Hafner [19]. We measure the success rates for all 22 achievements across all training episodes as a percentage and calculate their geometric mean to obtain our primary evaluation score[2]. Note that the geometric mean prioritizes unlocking challenging achievements. We also measure the episode reward, which indicates the number of achievements unlocked within a single episode, and report the average across all episodes within the most recent 100K environment steps. We conduct 10 independent runs using different random seeds for each experimental setting and report the mean and standard deviation. The code can be found at `https://github.com/snu-mllab/Achievement-Distillation`.

We compare our method with our backbone algorithm PPO and four baseline methods that have been previously evaluated on Crafter in other research work: DreamerV3, LSTM-SPCNN, MuZero + SPR, and SEA [21, 52, 53, 57]. DreamerV3 is a model-based algorithm that has achieved state-of-the-art performance on Crafter without any pre-training. LSTM-SPCNN is a model-free algorithm based on PPO that employs a recurrent and object-centric network to improve performance on Crafter. MuZero + SPR is a model-based algorithm that has demonstrated state-of-the-art performance on Crafter by utilizing unsupervised pre-training. SEA is a hierarchical algorithm based on IMPALA that employs

[2]The score is computed by $S = \exp(\frac{1}{N}\sum_{i=1}^{N} \ln(1+s_i)) - 1$, where $s_i \in [0, 100]$ is the success rate of the ith achievement and $N = 22$.

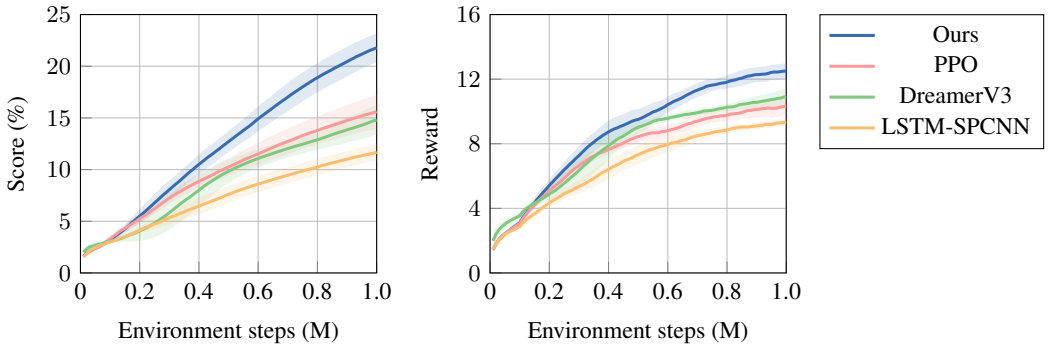


Figure 5: Score and reward curves.

a high-level planner to discover achievements on Crafter. We provide the implementation details and hyperparameters in Appendix C.

5.2 Crafter results

Table 1 and Figure 5 present the Crafter scores and rewards obtained by our method and the baselines. Our method outperforms all the baselines trained from scratch in both the metrics by a considerable margin, with a score of 21.79% and a reward of 12.60. Notably, our method exhibits superior score performance compared to MuZero + SPR, which utilizes pre-collected exploratory data and employs a computationally expensive tree search algorithm for planning, while achieving comparable rewards.

Figure 6 shows the individual success rates for all 22 achievements of our method and two successful baselines, DreamerV3 and LSTM-SPCNN. Remarkably, our method outperforms the baselines in unlocking challenging achievements. For instance, our method collects iron with a probability over 3%, which is 20 times higher than DreamerV3. This achievement is extremely challenging due to its scarcity on the map and the need for wood and stone tools. Moreover, our method crafts iron tools with a probability of approximately 0.01%, which is not achievable by either of the baselines. Finally, our method even succeeds in collecting diamonds, the most challenging task, on individual runs.

Table 1: Scores and rewards. MuZero + SPR† denotes the results replicated from the original paper.

Method		Parameters	Score (%)	Reward
Human Expert		-	50.5 ± 6.8	14.3 ± 2.3
From scratch	Ours	9M	$\mathbf{21.79 \pm 1.37}$	$\mathbf{12.60 \pm 0.31}$
	PPO	4M	15.60 ± 1.66	10.32 ± 0.53
	DreamerV3	201M	14.77 ± 1.42	10.92 ± 0.53
	LSTM-SPCNN	135M	11.67 ± 0.80	9.34 ± 0.23
	MuZero + SPR†	54M	4.4 ± 0.4	8.5 ± 0.1
	SEA	1.5M	1.22 ± 0.13	0.63 ± 0.08
Pre-training	MuZero + SPR†	54M	16.4 ± 1.5	12.7 ± 0.4

5.3 Model size analysis

We compare the model sizes between our method and the baselines. As shown in Table 1, our method achieves better performance with fewer parameters. In particular, our method only requires 4% of the parameters used by DreamerV3. Note that while our method has twice as many parameters as PPO, most of this increase is due to the networks used for the auxiliary learning, which are not utilized during inference. Additionally, we test our method with a smaller model by reducing the channel size to [16, 32, 32] and the hidden dimension to 256, resulting in a total of 1M parameters. Notably, it still outperforms the baselines with a score of 17.07%.

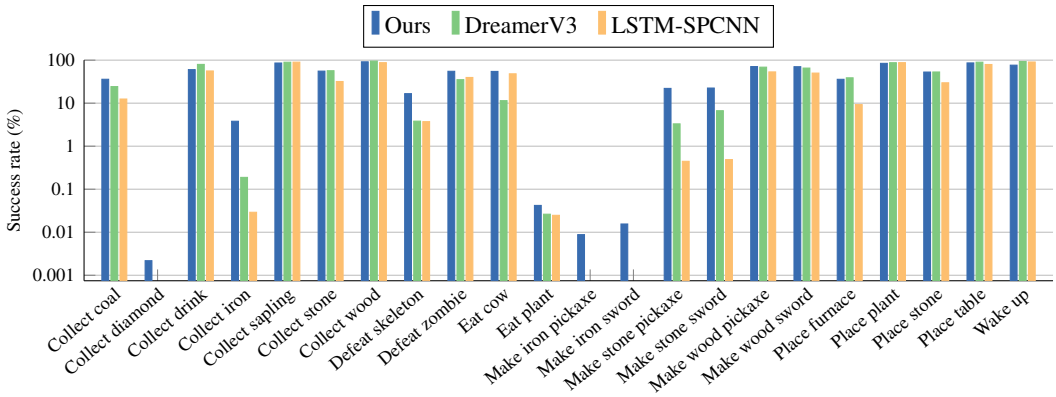


Figure 6: Individual success rates for all achievements.

5.4 Representation analysis of achievement distillation

To validate whether our method induces the encoder to have better representations for predicting the next achievements, we conduct an analysis of the latent representations of the encoder using the same approach as described in Section 3. Our method achieves a classification accuracy of 73.6%, which is a 28.7%p increase compared to PPO. Furthermore, Figure 3 demonstrates that our method produces predictions with significantly higher confidence than PPO, with a median value of 0.752.

5.5 Ablation studies

We conduct ablation studies to evaluate the individual contribution of our proposed method, intra-trajectory achievement prediction (I), cross-trajectory achievement matching (C), and memory (M). Table 2 shows that while intra-trajectory achievement prediction is the most significant contributor, cross-trajectory achievement matching and memory also play important roles in improving the performance of our method.

Table 2: Ablation studies.

I	C	M	Score (%)
✗	✗	✗	15.60 ± 1.66
✓	✗	✗	19.02 ± 1.65
✓	✓	✗	20.36 ± 1.79
✓	✓	✓	$\mathbf{21.79 \pm 1.37}$

5.6 Extension to value-based algorithms

In our previous experiments, we use the on-policy policy gradient algorithm, PPO, as our backbone RL algorithm. To assess the adaptability of our method to other RL paradigms, we extend our experiments to the popular off-policy value-based algorithm, QR-DQN, introducing a slight modification [11]. Specifically, we employ Huber quantile regression to preserve the Q-network's output distribution in alignment with the value function optimization in QR-DQN. We train the agent on Crafter for 1M environment steps and evaluate its performance. Notably, our method also proves effective for value-based algorithms, elevating the score from 4.14 to 8.07. The detailed experimental settings and results are provided in Appendix D.

5.7 Application to other environments

To evaluate the broad applicability of our method to diverse environments, we conduct experiments on two additional benchmarks featuring hierarchical achievements: Procgen Heist and MiniGrid [8, 7]. Heist is a procedurally generated environment whose goal is to steal a gem hidden behind a sequence of blue, green, and red locks. To open each lock, an agent must collect a key with the corresponding color. Heist introduces another challenge, given that the color of wall and background can vary between environments, whereas Crafter maintains fixed color patterns for its terrains. Additionally, we create a customized a door-key environment using MiniGrid to evaluate the effectiveness of our method on a deeper achievement graph. Notably, our method significantly improves the performance of PPO in Heist, increasing the score from 29.6 to 71.0. Our method also outperforms PPO in the MiniGrid environment by a substantial margin, elevating the score from 3.33 to 8.04. The detailed experimental settings and results can be found in Appendix E.

6 Related work

Discovering hierarchical achievements in RL One major approach to this problem is model-based algorithms. DreamerV3 learns a world model that predicts future states and rewards and trains an agent using imagined trajectories generated by the model [21]. While achieving superior performance on Crafter, it requires more than 200M parameters. MuZero + SPR trains a model-based agent with a self-supervised task of predicting future states [45, 53]. However, it relies on pre-training with 150M environment steps collected via RND to improve its performance on Crafter [6].

Another approach is hierarchical algorithms, while many of these methods are only tested on grid-world environments [51, 10]. HAL introduces a classifier that predicts the next achievement to be done and uses it as a high-level planner [10]. However, it relies on prior information on achievements, such as what achievement has been completed, and does not scale to the high-dimensional Crafter. SEA reconstructs the Crafter achievement graph using 200M offline data collected from a pre-trained IMPALA policy and employs a high-level planner on the graph [57]. However, it requires expert data to fully reconstruct the graph and has not been tested on a sample-efficient regime.

There are only a few studies that have explored model-free algorithms. LSTM-SPCNN uses an LSTM and a size-preserving CNN to improve the performance of PPO on Crafter [25, 30, 52]. However, it requires 135M parameters and the performance gain is modest compared to PPO with a CNN.

Representation learning in RL There has been a large body of work on representation learning for improving sample efficiency on a single environment [28, 55, 48], or generalization on procedurally generated environments [33, 34, 36]. However, representation learning for discovering hierarchical achievements has little been explored. A recent study has evaluated the performance of the previously developed representation learning technique SPR on Crafter, but the results are not promising [48, 53].

While widely used in other domains, optimal transport has recently garnered attention in RL [12, 15, 31]. However, the majority of the studies focus on imitation learning, where optimal transport is employed to match the behavior of an agent with offline expert data. In this work, we utilize optimal transport to obtain generalizable representations for achievements in the online setting.

7 Conclusion

In this work, we introduce a novel self-supervised method for discovering hierarchical achievements, named *achievement distillation*. This method distills relevant information about achievements from episodes collected during policy updates into the encoder and can be seamlessly integrated with a popular model-free algorithm PPO. We show that our proposed method is capable of discovering the hierarchical structure of achievements without any explicit component for long-term planning, achieving state-of-the-art performance on the Crafter benchmark using fewer parameters and data.

While we utilize only minimal information about hierarchical achievements (*i.e.*, the agent receives a reward when a new achievement is unlocked), one limitation of our work is that we have not evaluated the transferability of our method to an unsupervised agent without any reward. A promising future direction would be developing a representation learning method that can distinguish achievements in a fully unsupervised manner and combining it with curiosity-driven exploration techniques [49, 16].

Acknowledgements

This work was partly supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2020-0-00882, (SW STAR LAB) Development of deployable learning intelligence via self-sustainable and trustworthy machine learning, 80%, and No. 2022-0-00480, Development of Training and Inference Methods for Goal-Oriented Artificial Intelligence Agents, 20%). This research was supported by a grant from KRAFTON AI. This material is based upon work supported by the Air Force Office of Scientific Research under award number FA2386-23-1-4047. Hyun Oh Song is the corresponding author.

References

[1] Ankesh Anand, Jacob C Walker, Yazhe Li, Eszter Vértes, Julian Schrittwieser, Sherjil Ozair, Theophane Weber, and Jessica B Hamrick. Procedural generalization by planning with self-supervised world models. In *ICLR*, 2022.

[2] Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Leonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What matters for on-policy deep actor-critic methods? a large-scale study. In *ICLR*, 2021.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[4] Bowen Baker, Ilge Akkaya, Peter Zhokov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (VPT): Learning to act by watching unlabeled online videos. In *NeurIPS*, 2022.

[5] Jean-David Benamou, Guillaume Carlier, Marco Cuturi, Luca Nenna, and Gabriel Peyré. Iterative bregman projections for regularized transportation problems. *SIAM Journal on Scientific Computing*, 2015.

[6] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *ICLR*, 2019.

[7] Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo de Lazcano, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *arXiv preprint arXiv:2306.13831*, 2023.

[8] Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *ICML*, 2020.

[9] Karl W Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. Phasic policy gradient. In *ICML*, 2021.

[10] Robby Costales, Shariq Iqbal, and Fei Sha. Possibility before utility: Learning and using hierarchical affordances. In *ICLR*, 2022.

[11] Will Dabney, Mark Rowland, Marc Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *AAAI*, 2018.

[12] Robert Dadashi, Leonard Hussenot, Matthieu Geist, and Olivier Pietquin. Primal wasserstein imitation learning. In *ICLR*, 2021.

[13] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

[14] 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 *ICML*, 2018.

[15] Arnaud Fickinger, Samuel Cohen, Stuart Russell, and Brandon Amos. Cross-domain imitation learning via optimal transport. In *ICLR*, 2022.

[16] 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. In *NeurIPS*, 2022.

[17] William H. Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. MineRL: A large-scale dataset of Minecraft demonstrations. In *IJCAI*, 2019.

[18] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.

[19] Danijar Hafner. Benchmarking the spectrum of agent capabilities. In *ICLR*, 2022.

[20] Danijar Hafner, Timothy P Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *ICLR*, 2021.

[21] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[23] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI*, 2018.

[24] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. In *ICLR*, 2019.

[25] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

[26] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[27] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of zero-shot generalisation in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 2023.

[28] Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. In *ICML*, 2020.

[29] Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*. Springer, 2002.

[30] Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. In *NeurIPS*, 2020.

[31] Yicheng Luo, zhengyao jiang, Samuel Cohen, Edward Grefenstette, and Marc Peter Deisenroth. Optimal transport for offline imitation learning. In *ICLR*, 2023.

[32] Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 2018.

[33] Bogdan Mazoure, Remi Tachet des Combes, Thang Long Doan, Philip Bachman, and R Devon Hjelm. Deep reinforcement and infomax learning. In *NeurIPS*, 2020.

[34] Bogdan Mazoure, Ahmed M Ahmed, R Devon Hjelm, Andrey Kolobov, and Patrick MacAlpine. Cross-trajectory representation learning for zero-shot generalization in RL. In *ICLR*, 2022.

[35] 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.

[36] Seungyong Moon, JunYeong Lee, and Hyun Oh Song. Rethinking value function learning for generalization in reinforcement learning. In *NeurIPS*, 2022.

[37] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 1957.

[38] Suraj Nair, Yuke Zhu, Silvio Savarese, and Li Fei-Fei. Causal induction from visual observations for goal directed tasks. *arXiv preprint arXiv:1910.01751*, 2019.

[39] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[40] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[41] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

[42] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *AAAI*, 2018.

[43] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

[44] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 2020.

[45] Julian Schrittwieser, Thomas K Hubert, Amol Mandhane, Mohammadamin Barekatain, Ioannis Antonoglou, and David Silver. Online and offline reinforcement learning by planning with a learned model. In *NeurIPS*, 2021.

[46] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *ICLR*, 2016.

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

[48] Max Schwarzer, Ankesh Anand, Rishab Goel, R Devon Hjelm, Aaron Courville, and Philip Bachman. Data-efficient reinforcement learning with self-predictive representations. In *ICLR*, 2021.

[49] Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *ICML*, 2020.

[50] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016.

[51] Sungryull Sohn, Hyunjae Woo, Jongwook Choi, and Honglak Lee. Meta reinforcement learning with autonomous inference of subtask dependencies. In *ICLR*, 2020.

[52] Aleksandar Stanić, Yujin Tang, David Ha, and Jürgen Schmidhuber. Learning to generalize with object-centric agents in the open world survival game crafter. *arXiv preprint arXiv:2208.03374*, 2022.

[53] Jacob Walker, Eszter Vértes, Yazhe Li, Gabriel Dulac-Arnold, Ankesh Anand, Théophane Weber, and Jessica B Hamrick. Investigating the role of model-based learning in exploration and transfer. *arXiv preprint arXiv:2302.04009*, 2023.

[54] Erik Wijmans, Manolis Savva, Irfan Essa, Stefan Lee, Ari S. Morcos, and Dhruv Batra. Emergence of maps in the memories of blind navigation agents. In *ICLR*, 2023.

[55] Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Reinforcement learning with prototypical representations. In *ICML*, 2021.

[56] Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving sample efficiency in model-free reinforcement learning from images. In *AAAI*, 2021.

[57] Zihan Zhou and Animesh Garg. Learning achievement structure for structured exploration in domains with sparse reward. In *ICLR*, 2023.

A Representation analysis

To analyze the latent representations learned by PPO and our method, we construct a dataset using an expert policy. Specifically, we initially train the expert policy using our method with 1M environment steps. Note that this policy is trained with a different seed from the policies used for evaluation. Next, we collect a batch of episodes using the expert policy, resulting in a dataset containing 215,578 states. From this dataset, we subsample 50,000 states for the training set and 10,000 states for the test set.

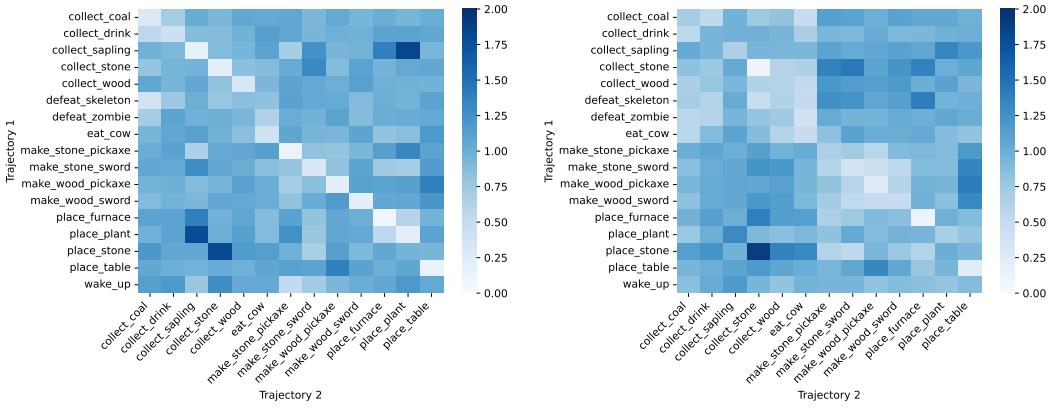
For each method, we acquire the latent representations from the encoder for the training set and freeze them. We then train a linear classifier using these representations to predict the very next achievements. Each achievement is labeled from 0 to 21, representing the different possible achievements in Crafter. We optimize the classifier for 500 epochs using the Adam optimizer with a learning rate of 1e-3 [26]. Finally, we measure the classification accuracy and the prediction probability (*i.e.*, confidence) for the ground-truth label on the test set.

B Examples of cross-trajectory achievement matching

To demonstrate the effectiveness of cross-trajectory achievement matching, we provide an example of matching results for our method and PPO. We first collect two different episodes using an expert policy, following the same procedure outlined in Appendix A. Subsequently, we acquire the representations of the achievement sequences for each method. Ultimately, we perform the matching process between the sequences of achievement representations for each method.

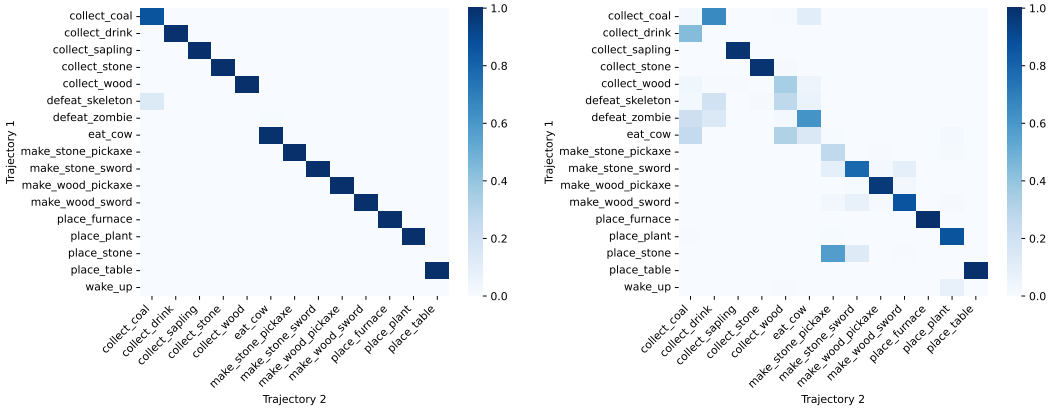
Figure 7 visualizes the cosine distance between the achievement representations from the two episodes. Remarkably, our method exhibits a lower cosine distance between the same achievements compared to PPO. This highlights the effectiveness of cross-trajectory achievement matching in facilitating the learning of generalizable representations for achievements across different episodes.

Figure 8 illustrates the soft matching results computed using partial optimal transport [5]. Notably, the matching result of PPO contains inaccurate or unconfident matchings, while our method does not suffer from this issue. Consequently, the hard matching result of PPO exhibits inaccurate matchings, as depicted in Figure 9b. For instance, "Defeat zombie" in the first episode is matched with "Eat cow" in the second episode and "Collect wood" is not matched at all. In contrast, Figure 9a shows that our method successfully matches identical achievements between episodes. Furthermore, our method avoids matching achievements in one episode that do not exist in the other episode.



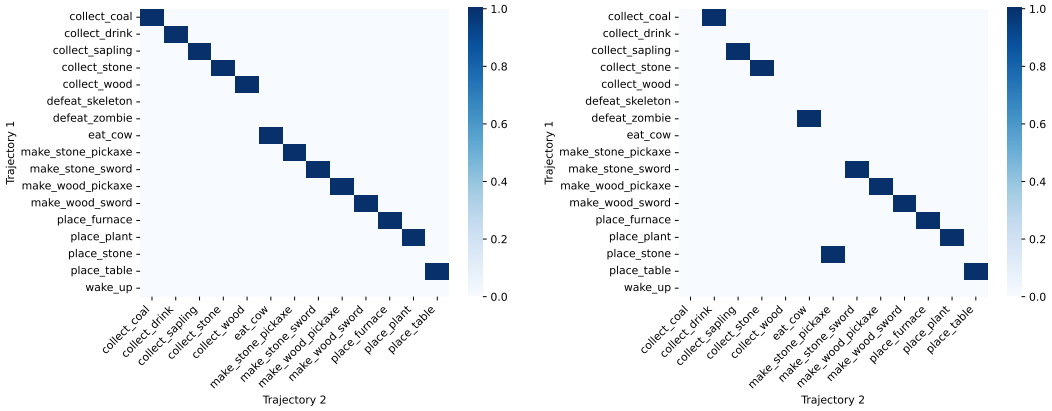
(a) Ours (b) PPO

Figure 7: Cosine distance between achievement representations.



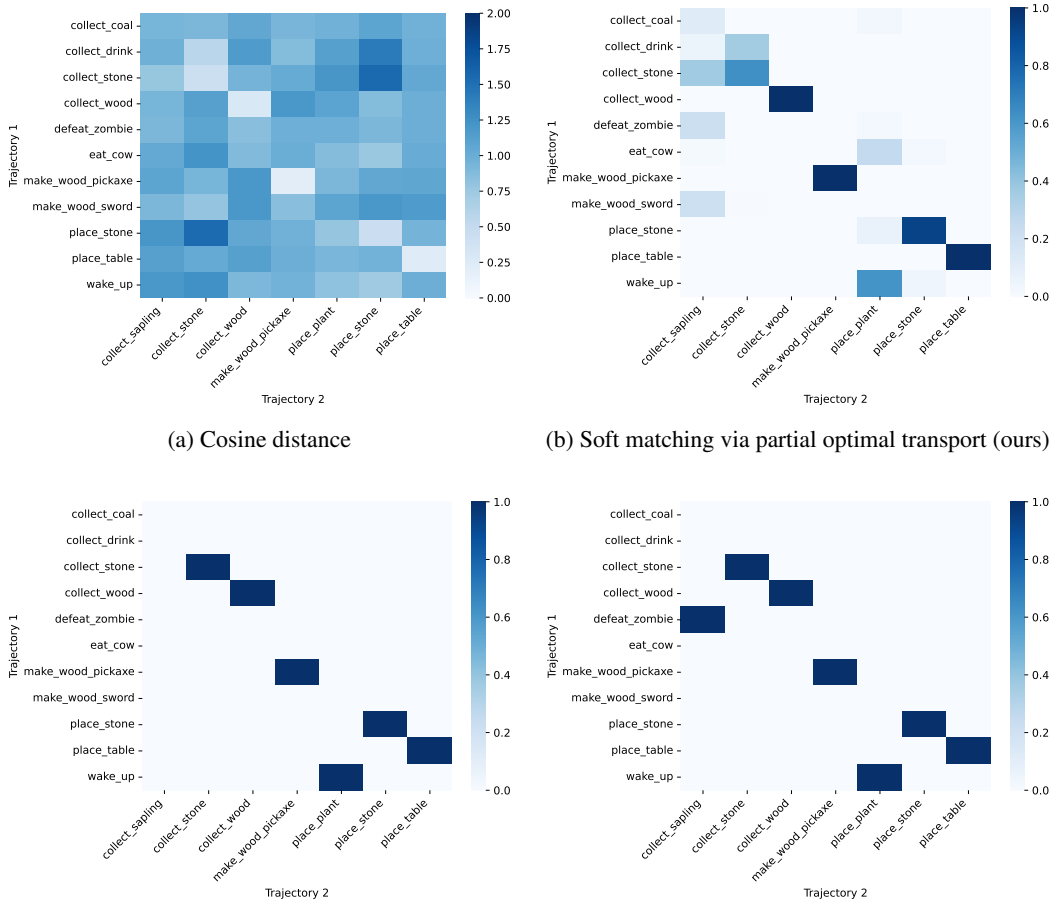
(a) Ours (b) PPO

Figure 8: Soft matching via partial optimal transport.



(a) Ours (b) PPO

Figure 9: Hard matching via thresholding.



(a) Cosine distance

(b) Soft matching via partial optimal transport (ours)

(c) Hard matching via thresholding (ours)

(d) Hard matching via Hungarian algorithm

Figure 10: Matching results of our matching algorithm and the Hungarian algorithm.

We further explore the effectiveness of our matching algorithm that utilizes partial optimal transport followed by thresholding during the early stage of training. Specifically, we first collect two different episodes using a policy trained with our method for up to 100 epochs. Subsequently, we employ our matching algorithm to calculate the hard matching between the achievement sequences extracted from these episodes and compare it with the Hungarian algorithm, which is commonly used in bipartite graph matching [37].

Figure 10 provides an example of the matching results obtained using our matching algorithm and the Hungarian algorithm. In the case of the Hungarian algorithm, "Defeat zombie" in the first episode is incorrectly matched with "Collect sapling" in the second episode, as shown in Figure 10d. In contrast, our matching algorithm effectively avoids matching "Defeat zombie" in the first episode, as depicted in Figure 10c.

C Experimental settings

C.1 Computational resources

All experiments are conducted on an internal cluster, with each node consisting of two AMD EPYC 7402 CPUs, 500GB of RAM, and eight NVIDIA RTX 3090 GPUs. We utilize PyTorch as our primary deep learning framework [41].

C.2 Implementation details and hyperparameters

PPO Our implementation of PPO is based on the official code repository (`https://github.com/openai/Video-Pre-Training`) provided by Baker et al. [4]. Each input image has dimensions of $64 \times 64 \times 3$. The image is first processed with a ResNet encoder as proposed in IMPALA, which consists of three stacks with channel sizes of $[64, 64, 128]$ [14]. Each stack is composed of a 3×3 convolutional layer with a stride of 1, a 3×3 max pooling layer with a stride of 2, and two ResNet blocks as introduced in He et al. [22]. Subsequently, the output of the encoder is flattened into a vector of size 8192 and passed through two consecutive dense layers with output sizes of 256 and 1024, respectively. This resulting vector serves as the latent representation. Finally, the latent representation is fed into two independent dense layers, the policy and value heads. The policy head has an output size of 17 and generates a categorical distribution over the action space. The value head has an output size of 1 and produces a scalar value representing the value function. All weights are initialized using fan-in initialization, except for the policy and value heads [29]. The weights of the policy and value heads are initialized using orthogonal initialization with a gain of 0.01 and 0.1, respectively [43]. All biases are initialized to zero. We use ReLU as the activation function [39]. All network parameters are optimized using the Adam optimizer [26].

For PPO training, we slightly modify the hyperparameter setting from Stanić et al. [52]. Specifically, we decrease the number of mini-batches per epoch from 32 to 8 and the number of epochs per rollout from 4 to 3. These modified settings are commonly used in Cobbe et al. [9], Moon et al. [36]. Instead of employing the reward normalization technique proposed in the original PPO paper, we normalize the value function target with the mean and standard deviation estimated through an exponentially weighted moving average (EWMA), following the practice in Baker et al. [4]. We set the decay rate of the EWMA to 0.99. We provide the default values for the hyperparameters in Table 3.

Table 3: PPO hyperparameters.

Hyperparameter	Value
Discount factor	0.95
GAE smoothing parameter	0.65
# timesteps per rollout	4096
# epochs per rollout	3
# mini-batches per epoch	8
Entropy bonus	0.01
PPO clip range	0.2
Reward normalization	No
EWMA decay rate	0.99
Learning rate	3e-4
Max grad norm	0.5
Value function coefficient	0.5

DreamerV3 To reproduce the results, we utilize the official code repository (`https://github.com/danijar/dreamerv3`) provided by Hafner et al. [21]. We use the recommended hyperparameter setting from the original paper. Specifically, we set the model size to XL and the training ratio, which is the number of imagined steps per environment step, to 512. For more detailed information about the hyperparameter, please refer to Table 4.

LSTM-SPCNN To reproduce the results, we utilize the official code repository (`https://github.com/astanic/crafter-ood`) provided by Stanić et al. [52]. Regarding the network architecture, a

Table 4: DreamerV3 hyperparameters.

Hyperparameter	Value
GRU recurrent units	4096
CNN multiplier	96
Dense hidden units	1024
MLP layers	5
Training ratio	512

size-preserving CNN is used as the image encoder [30]. This encoder consists of four convolutional layers, each with a kernel size of 5×5, a channel size of 64, and a stride of 1, and does not have any pooling layers. The encoder output is flattened into a vector of size 262144 and fed into a dense layer with an output size of 512, yielding the latent representation. For the policy network, the latent representation is first passed through an LSTM layer with a hidden size of 256 and then fed into a dense layer with an output size of 17, producing a categorical distribution over the actions [25]. For the value network, the latent representation is passed through two consecutive dense layers with output sizes of 256 and 1, respectively, to generate a scalar value representing the value function. For PPO training, we adopt the best hyperparameter setting from the original paper. Please refer to Table 5 for the default values for the hyperparameters.

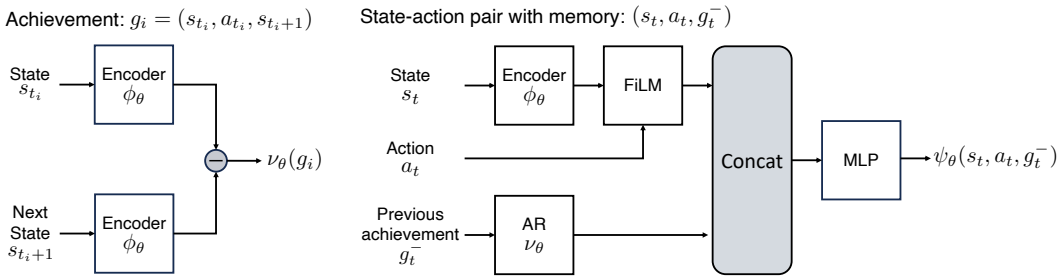
Table 5: LSTM-SPCNN hyperparameters.

Hyperparameter	Value
Discount factor	0.95
GAE smoothing parameter	0.65
# timesteps per rollout	4096
# epochs per rollout	4
# minibatches per epoch	32
Entropy bonus	0.0
PPO clip range	0.2
Reward normalization	No
Learning rate	3e-4
Max grad norm	0.5
Value function coefficient	0.5

MuZero + SPR We report the results replicated from the original paper since the official code has not been released yet [53]. It is worth noting that the paper uses an increased resolution for the input images, from 64×64 to 96×96, which can potentially result in improved performance compared to the original settings.

SEA To evaluate its performance, we utilize the official code repository (`https://github.com/pairlab/iclr-23-sea`) provided by Zhou and Garg [57]. It is important to mention that the paper uses a modified version of Crafter, where the health mechanism and associated rewards are removed. This modification allows the agent to be immortal and explore the world map without any constraints, making the environment much easier than the original. Furthermore, the paper trains an agent using a substantial number of 500M environment steps. The paper uses 200M environment steps to train the IMPALA policy for offline data collection, followed by additional 300M environment steps to train the sub-policies for exploration.

To ensure a fair comparison with other methods, we reproduce the results by training an agent on the original Crafter benchmark using 1M environment steps. Specifically, we use 400K environment steps for training the IMPALA policy and 600K environment steps for training the sub-policies, maintaining the same ratio as the original setting. Regarding the network architecture, we keep it unchanged from the original implementation. However, it is worth noting that the original implementation employs a higher resolution of 84×84 for the input images, which can potentially improve the performance. We adopt the best hyperparameter setting for training from the original paper. Please refer to Table 6 for the default values for the hyperparameters.



(a) Achievement representation (b) State-action pair representation

Figure 11: Network architectures for the representations of (a) achievements and (b) state-action pairs.

Table 6: SEA hyperparameters.

Hyperparameter	Value
Network architecture	CNN + LSTM
Hidden size	256
Learning rate	2e-4
Batch size	32
Unroll length	80
Gradient clipping	40
RMSProp α	0.99
RMSProp momentum	0
RMSProp ϵ	0.01
Discount factor	0.99
Reward normalization	Yes
# timesteps for IMPALA policy training	400K
# timesteps for achievement classifier training	100K
# timesteps for sub-policy training	600K

Achievement distillation Our method is built upon the PPO implementation, utilizing the same network architecture and hyperparameters for PPO training. For achievement distillation, we compute the representation of an achievement $g_i = (s_{t_i}, a_{t_i}, s_{t_i+1})$ by computing the difference between the latent state representations from the encoder:

$$\nu_\theta(g_i) = \phi_\theta(s_{t_i+1}) - \phi_\theta(s_{t_i}),$$

which is then normalized. This process is illustrated in Figure 11a. We also compute the representation of a state-action pair with the previous achievement as memory (s_t, a_t, g_t^-) as

$$\psi_\theta(s_t, a_t, g_t^-) = \mathrm{MLP}_\theta(\mathrm{Concat}(\mathrm{FiLM}_\theta(\phi_\theta(s_t), a_t), \nu_\theta(g_t^-))),$$

as described in Figure 11b. Specifically, the latent state representation from the encoder is combined with the action using a FiLM Layer:

$$\mathrm{FiLM}_\theta(\phi_\theta(s_t), a_t) = (1 + \eta_\theta(a_t))\phi_\theta(s_t) + \delta_\theta(a_t),$$

where η_θ and δ_θ are two-layer MLPs, each with a hidden size of 1024 [42]. The resulting vector is then concatenated with the achievement representation and passed through a two-layer MLP with a hidden size of 1024, followed by normalization.

When optimizing the contrastive objectives for achievement prediction and achievement matching, we jointly optimize the policy and value regularizer with the policy regularizer coefficient $\beta_\pi = 1.0$ and the value regularizer coefficient $\beta_V = 1.0$. We compute a soft-matching between two achievement sequences using partial optimal transport with the entropic regularizer coefficient $\alpha = 0.05$. Note that these values are set without any hyperparameter search.

For the policy and auxiliary phases, we search for the number of policy phases per auxiliary phase within the range of $\{4, 8, 16\}$ and find $N_\pi = 8$ is optimal. Similarly, we sweep over different values

for the number of epochs per auxiliary phase, considering values of $\{1, 3, 6\}$, and determine $E_{\text{aux}} = 6$ as the optimal choice. We provide the default values for the hyperparameter in Table 7.

Table 7: Achievement distillation hyperparameters.

Hyperparameter	Value
Policy regularizer coefficient (β_π)	1.0
Value regularizer coefficient (β_V)	1.0
Entropic regularizer coefficient (α)	0.05
# policy phases per auxiliary phase (N_π)	8
# epochs per auxiliary phase (E_{aux})	6

C.3 Environment details

Observation space An agent receives an image observation of dimensions $64 \times 64 \times 3$. This image contains a local, agent-centric view of the world map and the inventory state of the agent, such as the quantities of resources and tools and the levels of health, food, water, and energy.

Action space Crafter features a discrete 17-dimensional action space. The complete list of possible actions is provided in Table 8. The "Do" action encompasses activities including resource collection, consumption of food and water, and combat against enemies.

Table 8: Crafter action space.

Index	Name
0	Noop
1	Move left
2	Move right
3	Move up
4	Move down
5	Do
6	Sleep
7	Place stone
8	Place table
9	Place furnace
10	Place plant
11	Make wood pickaxe
12	Make stone pickaxe
13	Make iron pickaxe
14	Make wood sword
15	Make stone sword
16	Make iron sword

Achievements Crafter consists of 22 achievements that the agent can unlock by satisfying specific requirements. The complete list of the achievements and their corresponding requirements is presented in Table 9.

Table 9: Crafter achievements and their requirements.

Name	Requirements
Collect coal	Nearby coal; wood pickaxe
Collect diamond	Nearby diamond; iron pickaxe
Collect drink	Nearby water
Collect iron	Nearby iron; stone pickaxe
Collect sapling	None
Collect stone	Nearby stone; wood pickaxe
Collect wood	Nearby wood
Defeat skeleton	None
Defeat zombie	None
Eat cow	None
Eat plant	Nearby plant
Make iron pickaxe	Nearby table and furnace; wood, coal, and iron
Make iron sword	Nearby table and furnace; wood, coal, and iron
Make stone pickaxe	Nearby table; wood and stone
Make stone sword	Nearby table; wood and stone
Make wood pickaxe	Nearby table; wood
Make wood sword	Nearby table; wood
Place furnace	Stone
Place plant	Sapling
Place stone	Stone
Place table	Wood
Wake up	None

D Extension to value-based algorithms

Our implementation of QR-DQN is derived from an open-source implementation (`https://github.com/Kaixhin/Rainbow`) of Rainbow [23]. We use the original hyperparameter settings for training. For the network architecture, we employ the ResNet encoder, consistent with our PPO implementation. We apply our contrastive learning method to the Q-network encoder prior to each target Q-network update. The default values for the hyperparameters are shown in Table 10. Our method significantly improves the performance of QR-DQN, as shown in Figure 12.

Table 10: QR-DQN hyperparameters.

Hyperparameter	Value
Discount factor	0.95
Batch size	64
Replay frequency	4
Target update	8000
Learning rate	6.25e-5
Adam ϵ	1.5e-4
Max grad norm	10
Exploration strategy	ϵ-greedy

E Application to other environments

E.1 Procgen Heist

Heist is a procedurally generated door-key environment, whose goal is to steal a gem after unlocking a sequence of blue, green, and red locks, as illustrated in Figure 13. To open each lock, an agent must collect a key with the corresponding color. We consider unlocking each lock and stealing a gem as achievements. To ensure closer alignment with Crafter, we slightly adjust the reward structure so that an agent receives a reward of 2 for opening each lock and a reward of 10 for successfully stealing a

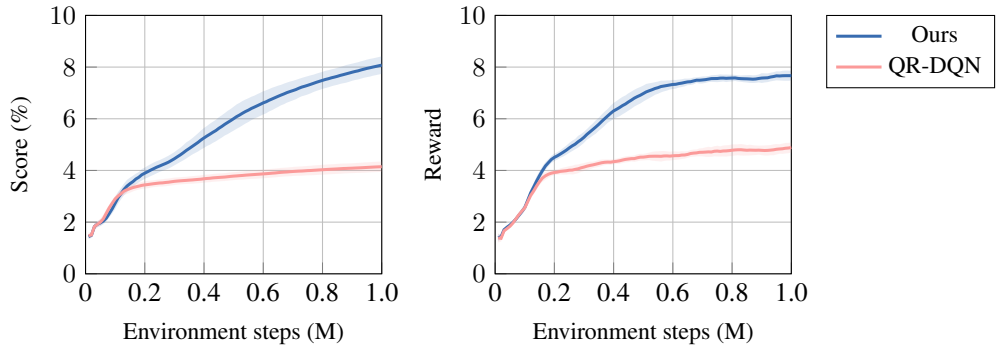


Figure 12: Crafter scores and rewards with QR-DQN algorithm.

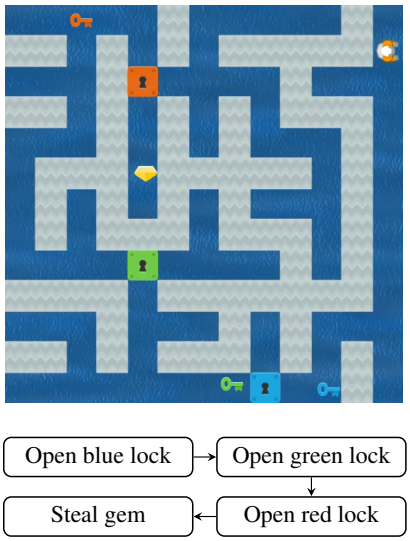


Figure 13: Overview of Procgen Heist. Each map features different layout and background color.

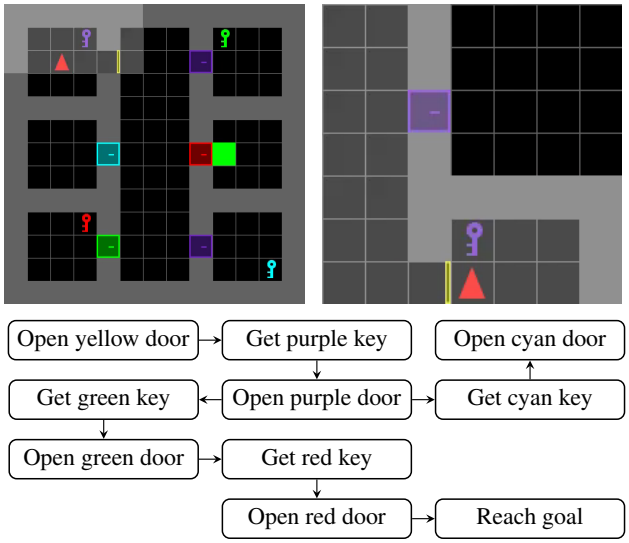


Figure 14: Overview of Custom MiniGrid environment. Each map consists of six rooms (left) and an agent observes its 7×7 surroundings (right).

gem. We train the agent in the "hard" difficulty mode for 25M environment steps and evaluate its performance in terms of the success rate of gem pilfering and the episode reward. Figure 15 shows that our method outperforms PPO by a significant margin throughout training.

E.2 MiniGrid

The design of the custom MiniGrid environment takes inspiration by TreeMaze proposed in SEA [57]. An agent must sequentially unlock doors, find keys, and finally reach the green square, as depicted in Figure 14. The environment comprises a total of 10 achievements. An agent receives a reward of 1 for unlocking a new achievement, mirroring the reward structure in Crafter. We train an agent for 1M environment steps and evaluate its performance in terms of the geometric mean of success rates and the episode reward, following the same protocol as Crafter. Figure 16 demonstrates that our method outperforms PPO and showcases reduced variance across various training seeds.

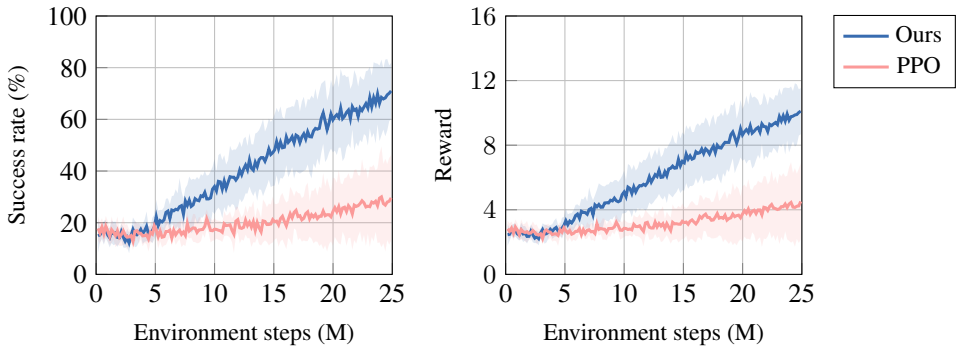


Figure 15: Procgen Heist success rate and reward curves.

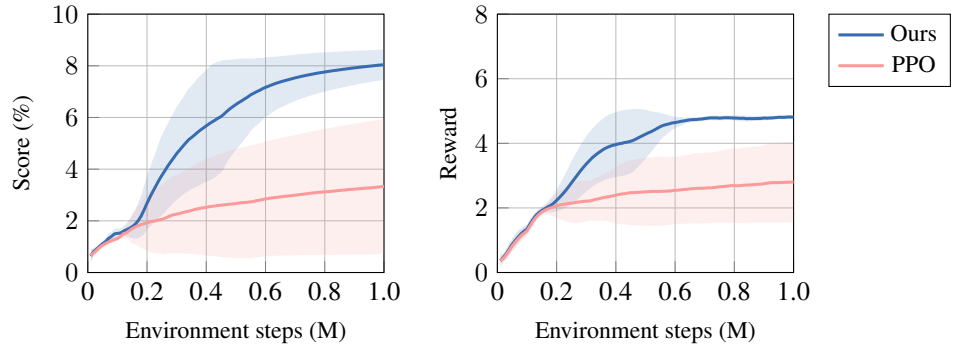


Figure 16: Minigrid score and reward curves.