Imagine Big from Small: Unlock the Cognitive Generalization of Deep Reinforcement Learning from Simple Scenarios

Anonymous Authors[1]

Abstract

The policies learned by humans in simple scenarios can be deployed in complex scenarios with the same task logic through limited feature alignment training, a process referred to as cognitive generalization or systematic generalization. Thus, a plausible conjecture is that unlocking cognitive generalization in DRL could enable effective generalization of policies from simple to complex scenarios through reward-agnostic fine-tuning. This would eliminate the need for designing reward functions in complex scenarios, thus reducing environment-building costs. In this paper, we propose a general framework to enhance the cognitive generalization ability of standard DRL methods. Our framework builds a cognitive latent space in a simple scenario, then segments the latent space to cluster samples with similar environmental influences into same subregion. During the fine-tuning in the complex scenario, the policy uses cognitive latent space to align the new sample with the same subregion sample collected from the simple scenario and approximates the rewards and Q values of the new samples for policy update. Based on this framework, we propose *Granular Ball Reinforcement Leaning* (GBRL), a practical algorithm via Variational Autoencoder (VAE) and Granular Ball Representation. GBRL achieves effective policy generalization on various difficult scenarios with the same task logic.

1. Introduction

Deep reinforcement learning (DRL) has witnessed striking empirical achievements in agents' training on difficult sequential decision-making tasks ranging from games (Mnih et al., 2013) to robotic control (Kaufmann et al., 2023). Notably, the effect of standard reinforcement learning relies on high-quality environments for training, such as accurate dynamic modeling of complex scenarios and well-designed reward functions (Sutton & Barto, 1999). As a result, deploying a policy learned in a simple scenario to a complex task with the same task logic (e.g., larger map, more branches and obstacles) becomes challenging (Kirk et al., 2023), which is a manifestation of low generalization ability. In contrast, humans can achieve this by mentally alignment simple task policy with complex scenario features. Brain science encapsulates this process as cognitive generalization (Lake & Baroni, 2023). Thus, a plausible conjecture is that unlocking the cognitive generalization capabilities of DRL may improve the performance of the pre-trained policy in the complex scenario of the same task logic.

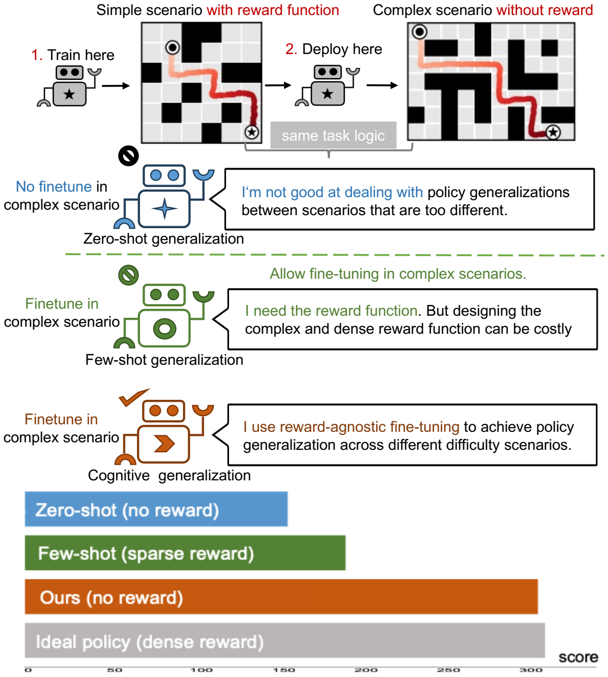
Simple scenario with reward function
Complex scenario without reward
1. Train here
2. Deploy here
same task logic
No finetune in complex scenario
I'm not good at dealing with policy generalizations between scenarios that are too different.
Zero-shot generalization
Allow fine-tuning in complex scenarios.
Finetune in complex scenario
I need the reward function. But designing the complex and dense reward function can be costly
Few-shot generalization
Finetune in complex scenario
I use reward-agnostic fine-tuning to achieve policy generalization across different difficulty scenarios.
Cognitive generalization
Zero-shot (no reward)
Few-shot (sparse reward)
Ours (no reward)
Ideal policy (dense reward)
score
0
50
100
150
200
250
300

Figure 1. A motivational demonstration of cognitive generalization on maze tasks. The bottom part shows the test scores for the complex maze scenario. Cognitive generalization enables *simple scenario policies* to be effectively deployed to *complex scenarios* with *the same task logic* via *reward-agnostic fine-tuning*.

Cognitive generalization's setting is different from "gener-

[1] Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

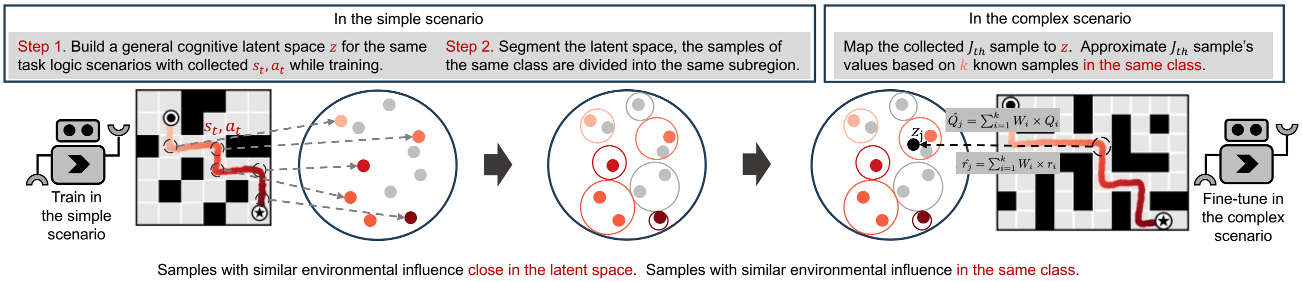
In the simple scenario
Step 1. Build a general cognitive latent space z for the same task logic scenarios with collected s_t, a_t while training.
Step 2. Segment the latent space, the samples of the same class are divided into the same subregion.
In the complex scenario
Map the collected J_{th} sample to z. Approximate J_{th} sample's values based on k known samples in the same class.
Train in the simple scenario
s_t, a_t
z_j
$\hat{Q}_j = \sum_{i=1}^{k} W_i \times Q_i$
$\hat{r}_j = \sum_{i=1}^{k} W_i \times r_i$
Fine-tune in the complex scenario
Samples with similar environmental influence close in the latent space. Samples with similar environmental influence in the same class.

Figure 2. The overview of our ideal framework for mainstream DRL algorithms.

alization across variations in environment structures" (Fan et al., 2021). "generalization across variations in environment structures" mainly focuses learning a robust policy on scenarios with multiple levels and prevent overfitting, i.e. focusing on the average score overall scenarios (Igl et al., 2020). However, we focus on how can the policy perform well in significantly more complex scenarios after only training in simple scenarios without the assistance of intermediate-level scenarios. Specifically, we consider enabling policies trained on simple scenarios to be effectively deployed to complex scenarios with the same task logic through reward-agnostic fine-tuning. The characteristics of this setting are as follows: 1) The initial policy only allows training in a single simple scenario and cannot improve the generalization ability by using cross-task knowledge like meta learning (Beck et al., 2023). 2) Both the test scenario and the training scenario share the same task logic, action space, and observation space, but the test scene setting is more complex. 3) Only the raw complex test scenario without reward feedback is provided. By adhering to this setting, we aim to reduce the cost associated with training effective policies for challenging tasks. For instance, in the game industry, engineers often have to construct complex reward functions for AI-controlled NPCs to enhance their humanoid behavior. This cost increases with the complexity of the game map (de Villiers & Sabatta, 2020). Even in simple simulation environments with consistent task logic, modifying the reward function entails labor-intensive and expensive manual debugging, involving the addition of auxiliary components and the delicate balancing of various factors. An empirical verification is in the Appendix D.1

Recently, there is some emerging research exploring the generalization capability of DRL, which mainly focuses on zero-shot generalization and few-shot generalization. The Zero-shot setting prohibits the policy from using any information from the test scenario and focuses more on the generalization of the same difficulty tasks (Tanabe et al., 2022). Few-shot generalization methods alleviate these restrictions to enhance policy performance in complex test scenarios, allowing for limited training in the test scenario (Eysenbach et al., 2020). However, the design of reward functions for test scenarios proves to be difficult and costly (de Villiers & Sabatta, 2020). Additionally, most methods require multi-task training based on a meta learning framework, which is not suitable for single-task generalization. In comparison to the strictly limited zero-shot generalization setting, the proposed cognitive generalization allows fine-tuning in the test scenario to enhance generalization feasibility, and at a lower cost than the few-shot setting which necessitates reward feedback in the test scenario (as shown in Figure 1). *Please refer to Appendix A for detailed comparison of related work.*

In this paper, we propose a general framework to improve the cognitive generalization ability of standard DRL methods. Our method stems from the idea of domain adaptation learning (Eysenbach et al., 2020): the agent's experience in the source domain can be used in the target domain with the same task logic. According to this, we assume that scenarios of the same task logic share a general cognitive latent space that contains all the samples. The overview idea is shown in Figure 2, we first build a cognitive latent space of state-action pairs in the simple scenario and cluster the points with similar impact on the environment, i.e., causing similar changes in the environment. Then we segment the latent space according to samples collected in the simple scene, the samples in the same subregion are considered to be of the same class. By doing this, When obtaining a new sample in the test task, we can map it to the segmented cognitive latent space. The Q value and reward can be estimated according to the semantically similar points, i.e., surrounding points in latent space, which enables effective policy fine-tuning in reward-agnostic scenarios. The key to realizing the above framework is to find the ideal latent space building and adaptive space segmentation methods.

Inspired by the recent remarkable successes of Deep Representation Learning in RL (Li et al., 2021; Chandak et al., 2019) and Granular Ball Representation (GBR) in small datasets adaptive clustering (Xia et al., 2023), we design *Granular Ball Reinforcement Leaning* (GBRL) as a practical algorithm. The implementation is divided into two stages: ❶ GBRL employs deep representation learning,i.e. VAE, to encode state-action pairs into a unified latent space, effectively capturing their underlying relationships. Particularly, GBRL utilizes an environmental dynamic loss to ensure that samples exerting similar environmental influences are closely positioned within the latent space. Then, GBRL uses GBR to segment the space according to the distribution of samples collected in simple scenes via adaptive clustering. ❷ Finally, we develop an evaluation system utilizing a granular ball style latent space. This system is used to approximate rewards and Q values of samples, enabling DRL algorithm to fine-tune according to TD-learning in test scenarios where explicit rewards are not available.

In summary, the contributions of this paper are as follows:

- To the best of our knowledge, this paper is the first to focus on unlocking the cognitive generalization ability of DRL. We propose a general framework to enhance cognitive generalization for mainstream DRL algorithms.
- Based on our framework, we provide a practical algorithm called Granular Ball Reinforcement Learning (GBRL) via Deep Representation Learning and Granular Ball Representation.
- GBRL outperforms various challenging cognitive generalization benchmarks by building a general cognitive latent space for scenarios with the same task logic.
- Additionally, GBRL surpasses both zero-shot and few-shot methods (utilize dense reward feedback in complex scenarios) across numerous tasks.

2. Backgrond

2.1. Deep Reinforcement Learning

DRL can be formulated based on Markov Decision Process (MDP) as a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$ with a state set $\mathcal{S}$, an action set $\mathcal{A}$, transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$, reward function $\mathcal{R}$, discounted factor $\gamma \in [0, 1)$. The transition probability function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$ maps states and actions to a probability distribution over the next states. The goal of RL is to learn a policy by maximizing the expected discounted return $R_t = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r_{t+k}]$, where γ is a discount factor for future rewards to dampen their effect.

2.2. Granular Ball Representation

Granular Ball Representation (Xia et al., 2019) (GBR), also known as Multi-granularity Ball Representation, is an important self-supervised data-covering method that has theoretically proved to be effective (Xia et al., 2022) and successfully ventured into diverse AI domains (Xie et al., 2023). It employs granular balls of varying sizes to adaptively represent and envelop the sample space, facilitating learning based on these granular balls. Therefore, GBR can theoretically fit arbitrary distributions and adaptively extend sparse fine-grained distributions to coarse-grained distributions (Xia et al., 2023).

We first introduce the definition of the granular ball on which GBR is based. Given a dataset D, $GB_j(j = 1, 2, ..., k)$denotes a granular ball generated based on D, with k representing the total number of granular balls. The specific definition of granular ball is as follows: $GB_j = \{y_i | i = 1, 2, ..., m_j\}$, m_j denotes the number of data points contained in GB_j. Center of the GB_j is defined as $c_j = \frac{1}{m_j} \sum_{i=1}^{m_j} y_i$. And radius is $u_j = \frac{1}{m_j} \sum_{i=1}^{m_j} (|y_i - c_j|)$.

Based on the above definition. To achieve a multi-granularity representation of data, GBR focuses on refining granule balls in an iterative splitting manner: ❶ All data points are initial as one granular ball first. ❷ Then the granular balls that satisfy the conditions are split into two GB (conditions are detailed below), and the refinement process ends when all GB can no longer be split. ❸ Finally, GB are merged according to the overlap degree.

3. Method

In this section, we describe in detail how we construct a general training framework to improve the cognitive generalization ability of policies. We first introduce the overall process of our framework in Section 3.1. Then, in section 3.2, self-supervised deep representation learning (i.e., Variational Autoencoder, VAE) and Granular Ball Representation (GBR) methods are introduced into the framework to provide an efficient practical algorithm called *Granular Ball Reinforcement Learning* (GBRL). For simplicity, we focus on the following exposition on actor-critic methods and adopt model-free DRL algorithm DDPG (Lillicrap et al., 2015) for demonstration.

3.1. Framework

Our goal is to design a training framework for injecting cognitive generalization into policies. By doing this, the policies can be effectively deployed to complex scenarios with the same task logic after reward-agnostic training. Recent research of domain generalization assumes that the pretrained data set contains key features in a high-dimensional space and successfully aligns these key features with the test dataset (Wang & Deng, 2018). Drawing inspiration from these works, our hypothesis is that the training scenario encompasses essential features (i.e., state-action pairs) that are relevant to the policy and can be extrapolated to the test scenario. Taking maze as an example, the general maze decision logic is to select a key action (e.g., turn, stop) in a key state (e.g., corner, encounter an obstacle) and find a path to the goal (end state). These key state-action pairs remain consistent across varying levels of complexity. Thus, the knowledge that the policy learns about key state-action pairs in simple scenarios can help it better make correct decisions at key states in complex scenarios.

Based on the above premise, cognitive generalization of DRL can be formally modeled as the following problem: *How to align complex scenario state-action pairs with simple scenario ones to help policies achieve generalization.* Consequently, we propose a two-stage process: **Stage 1)** We first train a standard DRL policy in the simple scenario and let the policy collect samples (state-action pairs) during training to build cognitive latent space z. Particularly, adjacent samples in z have similar environmental influences (i.e., causing similar changes in the environment). Then we employ an adaptive space segmentation method $C()$ to divide latent space z into subregions according to the sample distribution in z. Each subregion corresponds to a class of samples, and samples of the same class can be aligned with each other because they have similar environmental influences. **Stage 2)** In the test scenario without reward feedback, we deploy the policy trained in stage 1 and collect samples for fine-tuning. When the j_{th} sample is collected, we encode it into z and get its latent variable z_j, then use the rewards and Q values of k known samples in the class where z_j resides to approximate value $\hat{Q}_j$ and reward $\hat{r}_j$:

$$\hat{Q}_j = \sum_{i=1}^{k} \mathrm{W}_i \times Q_i,\ \hat{r}_j = \textstyle\sum_{i=1}^{k} \mathrm{W}_i \times r_i, \tag{1}$$

$$k \text{ neighbors} = \begin{cases} k \text{ nearest points in } cl_i, & \text{if } z_j \in cl_i \\ k \text{ nearest points in space } z, & \text{else} \end{cases}, \tag{2}$$

for accurately approximating, we use similarity weight $W_i = \frac{\zeta(i)}{\sum_{i=1}^{k} \zeta(i)}, \zeta(i) = 1 - \frac{|z_j - z_i|}{\sum_{i=1}^{k} |z_j - z_i|}$ to balance the influence of k neighbor. The effect of W_i is that samples closer to z_j have more influence on the approximation of z_j. Because the sample distance represents the similarity of its impact on the environment, and samples with similar environmental influences have similar Q values and rewards. cl_i denotes the i-th class obtained after processing by $C()$. With $\hat{Q}$ and $\hat{r}$, the policy can be trained in the complex scenario without reward feedback via TD-learning (Mnih et al., 2013). Notably, if z_j does not belong to any class, we take the sample j as the center and select k closest samples (analysis in Appendix D.4). Taking DDPG as an example, after training in the simple scenario, DDPG continues to optimize its policy in its own training way using $\hat{Q}$ and $\hat{r}$ in the complex scenario:

$$L_Q(\theta_i) = \mathbb{E}_{s,a,s'}\left[(y - Q_\theta(s,a))^2\right] \tag{3}$$

Where $y = \hat{r} + \gamma\hat{Q}(s', \pi_{\bar{\tau}}(s'))$, s' is the next state and $\bar{\tau}$ denotes the target network parameters. $\hat{r}$ and $\hat{Q}$ calculated by Eq.1 represent target Q and current reward. The actor is updated according to the Deterministic Policy Gradient (Lillicrap et al., 2015):

$$\nabla_\tau J(\tau) = \mathbb{E}_s\left[\nabla_{\pi_\tau(s)} Q_{\theta_1}(s, \pi_\tau(s)) \nabla_\tau \pi_\tau(s)\right] \tag{4}$$

3.2. Granular Ball Reinforcement Learning

3.2.1. Building Cognitive Latent Space via Environmental Influence-aware VAE

We use VAE's representational ability to construct cognitive latent spaces in simple scenarios. For an action a and corresponding states s, our encoder $q_\phi(z|s,a)$ parameterized by ϕ builds a cognitive latent space z, and maps state-action paris (s,a) into the latent variable $z_{s,a} \in \mathbb{R}^{d_1}$ (d_1 denotes the dimension of z). Decoder $p_\psi(\hat{s}, \hat{a}|z)$ parameterized by ψ then reconstructs the state-action pairs (s,a) from z. We summarize the encoding and decoding process:

$$\begin{aligned} &\textbf{Encoder}: z_{s,a} \sim q_\phi(\cdot|a,s) \quad, \forall\ s,a\ , \\ &\textbf{Decoder}: \hat{s}, \hat{a} = p_\psi(z_{s,a}, s) \quad, \forall\ z_{s,a}\ . \end{aligned} \tag{5}$$

According to (Kingma & Welling, 2013), Variational Auto-Encoder (VAE) can be optimized by maximizing the variational lower bound. The loss function of our VAE is L_{VAE}:

$$\begin{aligned} L_{VAE}(\phi,\psi) = \mathbb{E}_{s,a\sim D^\star, z\sim p_\psi}\big[\|\hat{s} - s\|_2^2 + \|\hat{a} - a\|_2^2 \\ + D_{KL}\big(q_\phi(\cdot|s,a \| N(0,I))\big)\big]\ , \end{aligned} \tag{6}$$

where $D^\star$ is the sample set collected by the policy in the simple scenario. The first term is the reconstruction error (i.e., mean square error, MSE), the last term is the Kullback Leibler divergence D_{KL} between the variational posterior of latent representation z and the standard Gaussian prior. By using the reparameterization trick (Kingma & Welling, 2013), $\hat{s}, \hat{a}$ is differentiable with respect to ψ and ϕ. For any latent variables $z_{s,a}$, they are decoded into s, a conveniently by the VAE decoder.

To further meet the requirements set out in section 4.1, we should make adjacent samples in z have similar environmental influences. We make z capture the environmental influences by predicting the environmental dynamics (state transition residual): Decoder predicts the residual difference between the state after the execution of a and the current state s. h_{ψ_1} is a subnetwork followed the decoder. For any transition sample (s,a), the state residual is denoted by $\delta_{s,s'} = s' - s$. $p_{state} = h_{\psi_1} \circ p_\psi$. $\circ$ denotes the cascaded heads. The reason we use cascaded heads is traditional parallel heads for both reconstruction and the state residual prediction followed by the same latent features may interfere with optimizing individual objectives and hinder the learning of the shared representation (Azabou et al., 2021). The predictions $\hat{\delta}_{s,s'}$ is produced as:

$$\hat{\delta}_{s,s'} = p_{state}(z_{s,a}) \quad \forall\ z_{s,a}\ . \tag{7}$$

Then we apply an unsupervised learning loss based on environmental dynamics prediction to refine the latent space and make samples with similar environmental influences closer. The benefits of dynamics predictive representation are also demonstrated in (Whitney et al., 2019):

$$L_{dy}(\phi, \psi, \psi_1) = \mathbb{E}_{s,a,s'}\left[\|\hat{\delta}_{s,s'} - \delta_{s,s'}\|\right] . \tag{8}$$

Above all, the ultimate training loss for our representation is denoted as follows:

$$L(\phi, \psi, \psi_1) = L_{VAE}(\phi, \psi) + \beta L_{dy}(\phi, \psi, \psi_1) , \tag{9}$$

where β is a hyper-parameter that controls the relative importance of the L_{dy} and L_{VAE}. L only depends on the environmental dynamic data which is reward-agnostic (Erraqabi et al., 2021). Notably, VAE is periodically trained in the test scenario in stage 2 to continuously refine the latent space z. We use state transition as a measure of environmental influence for the following reasons: ❶ State transition does not require a per-step reward, it can be used in non-reward test scenarios. ❷ Evaluation of Q values in the early stage of training is inaccurate. In contrast, state transition is more reliable and accessible. ❸ The same reward or Q value may correspond to different dynamic changes, but the same environmental change has the same reward or Q value.

3.2.2. Adaptively Segmenting Latent Space via Granular-ball Representation

In the previous section, we introduce the construction of a general cognitive latent space z. Based on the inference from section 3.1, samples in complex scenarios can also be mapped to the space z. This section describes how to segment cognitive latent space into subregions via GBR and divide the samples in the complex scene into different subregions. Intuitively, a simple method is to directly align each new sample with its k nearest neighbors in the latent space z. However, aligning solely based on nearest neighbors, without considering the associated class, can lead to inaccuracies. Specifically, data on the edge may be negatively influenced by samples from different classes. By segmenting the latent space and focusing on subregion-specific neighbors, we mitigate the risk of such cross-class errors, ensuring a more robust alignment.

Intuitively, an effective adaptive space segmentation method should satisfy the following characteristics: ❶ adaptive segmenting without the number of classes, because the classes of data can not be known in advance.❷ Unsupervised clustering based on data distribution.❸ Clear class boundary for space segmentation. Unfortunately, mainstream clustering algorithms' high dependence on prior parameters (e.g. classes number) and inefficiency in dealing with sparse distributions led us to explore alternative approaches in practice (detailed experiments and analysis in Appendix D.4). Meanwhile, the success of Granular ball representation (GBR) in small dataset clustering has attracted our attention (Xia et al., 2023). GBR divides categories by balls without prior knowledge, and the data in the same ball is considered to belong to the same category. GBR exhibits distinct class boundaries for effective space segmentation, eliminating the necessity for specifying the number of classes, thus aligning perfectly with our requirements.

Following the two steps that GBR clusters data in Euclidean space (① **Generation of** GB ② **Merging of** GB), we use GBR to cluster data set $D^\star$ (collected from the simple scenario) in the cognitive latent space z. We define GB (granular ball) by its assigned data $z_i = q_\psi(s_i, a_i)$, centroid c and radius u. $q_\psi(s_i, a_i)$ indicates the encode of i-th state-action pair in $D^\star$.

$$GB = \{z_{s_i,a_i} | i = 1, 2, ..., m\}, \tag{10}$$

$$c = \frac{1}{m}\sum_{i=1}^{m} z_i, \quad u = \frac{1}{m}\sum_{i=1}^{m}(|z_i - c|), \tag{11}$$

where m represents the number of data contained in GB. Partitioning the subspaces is an iterative process: The whole data set $D^\star$ is initialed as a granular ball first. Then the granular balls that satisfy the split conditions are split (conditions described below). And the refinement process ends when all granular balls can no longer be split which represents the completion of the expansion of the data distribution. To achieve an adaptive evaluation of whether GB satisfies the split conditions, we use GBR's original definition of DM (Distribution Measure) which is computed by dividing the number of data points by the sum of the radius $u^\star$ in a GB:

$$DM = \frac{u^\star}{m}, \quad u^\star = \frac{1}{m}\sum_{i=1}^{m} u_i, \tag{12}$$

The principle of DM measuring the quality of the GB is: If $DM_A > DM_B$, denotes A is sparser than B. DM_A and DM_B denote the data distribution quality values for granular ball A and granular ball B. *See Appendix B.1 for efficient proof.*

Generation of GB. We initiate the iterative splitting process by identifying the point z_1 that is farthest from the center c_A of the initial granular ball A. Subsequently, we proceed to locate the point z_2, which is farthest from z_1. Next, we calculate the mid-point, $\dot{z}_1$, between c_A and z_1, as well as the mid-point, $\dot{z}_2$, between c_A and z_2. These mid-points are designated as the initial cluster centers. To form two candidate granular balls, A_1 and A_2, we allocate the remaining points to either $\dot{z}_1$ or $\dot{z}_2$ based on their respective distances. Then, we used DM to adaptively determine whether to split. The definition of the split threshold DM_{weight} is:

$$DM_{weight} = \frac{|A_1|}{|A|} DM_{A_1} + \frac{|A_2|}{|A|} DM_{A_2}, \tag{13}$$

where $|A|$, $|A_1|$ and $|A_2|$ denote the number of data points contained within the corresponding GB. If $DM_A > DM_{weight}$, the volume of GB_A is greater than the weighted sum of the volumes of GB_{A_1} and GB_{A_2} based on the number of data points, GB_A will divide into GB_{A_1} and GB_{A_2}. *See appendix B.2 for theoretical proof.* This splitting process proceeds iteratively.

To prevent excessive splitting of GB, we introduce the hyperparameter minimum capacity η. GB stops splitting when its capacity $< \eta$ after splitting. According to the above interactive processing, numerous granular balls are obtained. We gather all the generated GB into a set called GBS (the granular-ball set): $GBS = \{GB_i | i = 1, 2, ..., M\}$, where M represents the number of GB contained in GBS.

Merging of GB. To achieve a multi-granularity representation, $D^\star$ is initially split to obtain a GBS, as illustrated in Eq.10-13. However, unconstrained interactive splitting leads to GBS redundancy (duplicate data coverage). Therefore, the overlapping granular balls within GBS are merged together to effectively represent the density distribution of the dataset. This interactive process stops until the number of GBS remains the same. Overlapping occurs when GB_i and GB_j meet the following merging conditions:

$$c_i - c_j - (u_i + u_j) < \omega, \omega = \frac{\min(u_i, u_j)}{\min(x_i, x_j)}, \tag{14}$$

where x_i represents the cumulative value of the number of overlaps with adjacent GB for GB_i, respectively. Similarly, x_j is associated with GB_j. The specific calculation rules for ω can be understood as GB_i and GB_j are the cumulative values of the overlapping times of the neighbor granular ball as the adjustment coefficients of ω, which can dynamically adjust the criterion that meets the adjacent rules. To prevent the wrong differentiation of the granular balls, the larger the cumulative value is, the stricter the standard that meets the adjacent rules. *Appendix E shows pseudocode of GBR.*

Value approximation in the complex scenario. When space segmentation is complete, policies are deployed in complex scenarios for fine-tuning. We classify samples collected from complex scenario via GBS, Eq.1 can be rewritten as:

$$\hat{Q}_j = \sum_{i=1}^{k} W_i \times Q_i, \ \hat{r}_j = \sum_{i=1}^{k} W_i \times r_i \tag{15}$$

$$k \text{ neighbors} = \begin{cases} \text{nearest } k \ \in GB_i, & \text{if } z_j \in GB_i \\ \text{nearest } k \in GB_i, \min \frac{|z_j - c_i|}{u_i} & \text{otherwise} \end{cases}, \tag{16}$$

we use the "distance-radius ratio" $\min \frac{|z_j - c_i|}{u_i}$ to rationally select GB for z_j. Compared to Euclidean distance, this method prevents the sample from being mistakenly placed into the small GB (the smaller the GB, the more special the semantics). With GBRL, agents encode the samples of the test scenario into z. Then evaluate the new sample $\hat{Q}$ value and rewards $\hat{r}$ according to Eq.15, and finally updates policy according to Eq.3,Eq.4. Pseudo code 1 shows the complete process.

Algorithm 1 GBRL-DDPG

```
Initialize: Actor $\pi_\tau$, critic networks $Q_\theta$,VAE $q_\phi, p_\psi$, replay buffer $D$ and whole data set $D^\star$.

Stage 1 in simple scenario
while $t <$ maximum training time do
    Fill $D, D^\star$ with data generated by $\pi_\tau$
    Update $Q_\theta$ and $\pi_\tau$ using samples in $D$.
    Update $q_\phi, p_\psi$ using $D^\star$. Build space $z$.    ▷ Eq.9
end while
Use samples in $D^\star$ to construct granular ball set $GBS$ in space $z$.    ▷ Eq.10-14

Stage 2 in complex scenario
Clear $D$.
while $t <$ policy fine-tuning time do
    $a_t = \pi_\tau(s_t)$ (with Gaussian noise)
    Execute $a_t$, observe new state $s'$
    $a_{t'} = \pi_\tau(s_{t'})$, fill $D, D^\star$.
    $z_{s_{t'}, a_{t'}} = q_\phi(s_{t'}, a_{t'})$.
    Determine the $GB_i$ to which $z_{s_{t'}, a_{t'}}$ belongs.
    Get $Q(\hat{s_{t'}}, a_{t'}), \hat{r_{t'}}$.    ▷ Eq.15
    Sample from $D$, update $Q_\theta$ and $\pi_\tau$.    ▷ Eq.3,4
    if reach unsupervised training time then Update $q_\phi, p_\psi$ using samples in $D^\star$, and rebuild $GBS$.
    end if
end while
```

4. Experiments

We empirically evaluate GBRL to answer the following research questions. (1) **RQ1**: **Effectiveness**. Can GBRL achieve mainstream algorithms' policies generalization from simple scenarios to complex scenarios without complex scenario reward feedback? When given reward feedback, does GBRL perform well compared to few-shot and zero-shot methods? (2) **RQ2**: **Cognitive latent space**. Are points with similar influence on the environment close in cognitive latent space, and does it play a role in improving the effect? (3) **RQ3**: **Adaptive sample classification**. Whether the GBR has an effect on the training of the policy by adaptively classifying new samples into the correct class. *For more hyperparameter sensitivity analysis please refer to Appendix D.2, i.e., analyisis of the neighbour number k and the effect of GB minimize capacity η on the policy.*

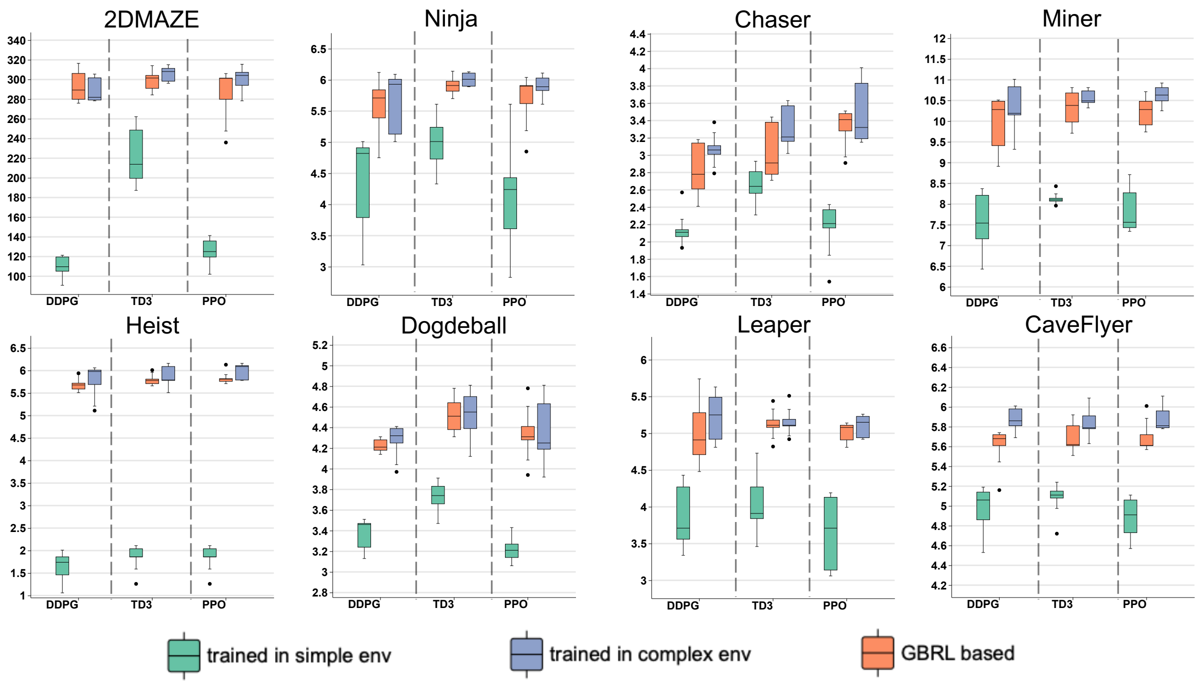

Figure 3. Evaluation of GBRL combined with mainstream DRL algorithms (DDPG, TD3, PPO) in six test tasks. GBRL significantly improves the performances of three DRL algorithms, and GBRL policies are equivalent to the policies trained on complex scenarios in most scenarios. The results of ten runs of each algorithm are summarized as shown in the figure.

4.1. Baselines

As far as we know, there are no specific methods for cognitive generalization. Thus, we take the DRL methods trained in complex scenarios and policies trained on simple scenarios as the baselines to visualize the boost GBRL provides to online DRL algorithms.

Further, we choose two types of methods that are similar to our setting to show the advantage of Cognitive Generation: one popular method for zero-shot generalization (i.e., M2TD3 (Tanabe et al., 2022)) and two popular methods for few-shot generalization (i.e. IDAAC (Raileanu & Fergus, 2021) and ExpGen (Zisselman et al., 2024)). See Appendix C for detailed hyperparameter setting and practical network architecture for GBRL.

- M2TD3 focuses on addressing the worst-case scenarios under stochastic dynamics. It achieves generalization on unknown scenarios with small differences.
- IDAAC is a SOTA few-shot method on Procgen. It decouples the optimization of the policy and value function and introduces an auxiliary loss to ensure that generic features are learned.
- ExpGen: A novel few-shot method that sets new SOTA results on ProcGen by training an additional ensemble of agents that optimize reward.

4.2. Benchmarks

We evaluate methods on two open source benchmarks. 2DMAZE in D4RL (Fu et al., 2020) and seven environments from the Procgen (Cobbe et al., 2020) which is specifically designed to measure generalization by introducing separate test- and training levels. Each environment supports two well-calibrated difficulty settings. In the 2DMAZE task, we used two scenarios: medium and hard, policies are trained on the medium scenario and tested on the hard scenario. All Procgen environments use a discrete 15 dimensional action space and produce $64 \times 64 \times 3$ RGB observations. To ensure that the performance of the algorithm is not affected by the image input, we define the background as black for all tasks and uniformly use a pre-trained MLP to convert the pixel input into a 128 dimension vector input. We use Procgen's easy setup, so for each game, agents are trained on an easy level and tested on the hard level. *See the appendix C.4 for a detailed description of the environment.*

4.3. Main results (RQ1)

We evaluate GBRL combined with the mainstream DRL algorithm in six scenarios. We select PPO, TD3 and DDPG as the backbones, and use policies that are directly trained by corresponding algorithms in complex scenarios as the measurement standard. To adapt PPO, GBRL converts the approximation of $Q(s, a)$ to the approximation of $V(s)$. The training step in simple scenarios is $1e6$ and in complex ons is $5e6$, $D^{\star}$ is $2e6$. See the Appendix C.2 for detailed hyperparameters. The results in Figure 4 show that GBRL significantly improves the performance of all DRL algorithms. When GBRL is used to train policies on simple scenarios, the performance is significantly improved. GBRL stimulates the potential of the policies after fine-tuning without reward, making its performance comparable to that of the ideal policies. In the Q-learning method, with the support of GBRL, DDPG performs better than the ideal DDPG in half tasks. We analyzed that the reason is that GBRL uses the average value of Q values of peripheral points to approximate new samples, thereby solving the problem of overestimation of Q values. TD3 adds a critic on the basis of DDPG to prevent overestimation, so the gain of GBRL is slightly smaller but also reaches the ideal setting level.To show the advantage of cognitive generalization in single-task complex scenario generalization, we relax the constraint and provide a reward function for complex scenarios to make the few-shot method deploy effectively. GBRL is compared with three SOTA generalized DRL algorithms in seven Procgen tasks. To ensure fairness, all methods have the same number of fine-tuning steps, i.e. $3e5$. Tab.1 confirms that GBRL outperforms baselines after the same step of reward-agnostic fine-tuning.

Task	M2TD3	IDAAC	ExpGen	GBRL-TD3
Ninja	5.13 ± 0.32	6.03 ± 0.14	6.15 ± 0.07	6.16 ± 0.11
Chaser	1.35 ± 0.06	3.14 ± 0.12	3.46 ± 0.26	3.27 ± 0.08
Heist	4.31 ± 0.73	4.72 ± 0.42	5.48 ± 0.45	5.65 ± 0.53
Leaper	2.7 ± 0.63	5.30 ± 1.19	5.17 ± 0.94	5.21 ± 1.03
Miner	7.17 ± 2.12	10.23 ± 0.7	11.66 ± 2.78	12.32 ± 1.97
Miner	8.69 ± 0.91	9.83 ± 1.04	10.37 ± 1.14	10.67 ± 0.62
Dodgeball	3.59 ± 1.14	4.83 ± 0.67	4.83 ± 0.67	4.80 ± 0.41

Table 1. The evaluations of scores on six typical Generalization tasks. GBRL outperforms on almost tasks. Average of 5 runs.

4.4. Evaluation of cognitive latent space (RQ2)

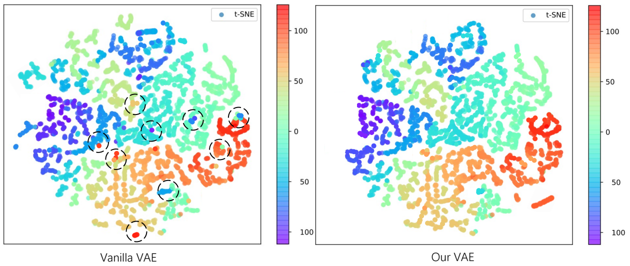

Figure 4. 2D t-SNE visualizations of learned representation, colored by 1D t-SNE of the corresponding environmental impact.

To demonstrate that a high-quality latent space can be constructed using our VAE, we train vanilla VAE and our VAE (with environmental influence regularization) following the GBRL process on Miner-Hard scenario. Then, we adopt t-SNE (van der Maaten & Hinton, 2008) to visualize the learned cognitive latent spaces in a 2D plane. We color each action based on its impact on the environment i.e., $\delta_{s,s'}$. Figure 4 shows that our VAE constructs a smoother and more compact latent space, where points with similar impacts on the environment are clustered in the latent space. Besides, results in Tab.2 show that GBRL with our VAE outperforms GBRL with vanilla VAE by an average of $> 20\%$ on six Procgen tasks, which further proves the effectiveness of our VAE in constructing cognitive latent space.

Task	GBRL with Vanilla VAE	GBRL with our VAE	improvement
Heist	4.81 ± 0.18	5.25 ± 0.53	9.15%
Miner	9.36 ± 2.85	11.07 ± 2.97	18.27%
Dodgeball	4.09 ± 0.72	4.80 ± 0.41	17.36%
Ninja	5.08 ± 0.31	6.16 ± 0.11	21.26%
Leaper	4.15 ± 0.53	5.21 ± 1.03	25.54%
Chaser	2.35 ± 0.15	3.17 ± 0.08	34.89%

Table 2. Ablation experiments for the environment predicting regularization of VAE. Average of 5 runs.

4.5. Evaluation of GBR (RQ3)

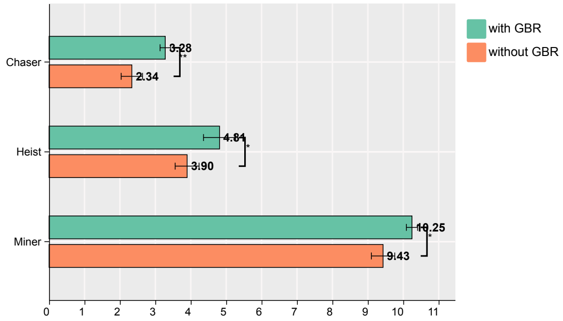

Figure 5. Evaluation of GBR in three tasks by using TD3.

To show the effectiveness of GBR for sample classification, we conduct experiments on three Procgen tasks to compare the effect of computing Q values only using k-neighbor points in the latent space and computing Q values with the assistance of GBR. The results in Figure 5 prove that with GBR, policies perform better. We think this is because GBR prevents samples from being affected by irrelevant points by classifying samples into corresponding GB.

5. Conclusion

In this paper, we first observe that cognitive generalization can help DRL algorithm achieve low-cost efficient policy generalization from simple scenarios to complex scenarios with the same task logic. And we propose a general framework to unlock the cognitive generalization of DRL algorithm by constructing a general cognitive latent space. Further, an effective practical method, i.e. GBRL, is provided with the help of GBR and VAE. GBRL achieves SOTA performance on most tasks through reward-agnostic generalization training.

6. Impact Statements

By rethinking our paper, we think this research may have the following implications:

① This paper is the first to discuss cognitive generalization in the reinforcement learning community and proposes that improving the cognitive generalization ability of reinforcement learning algorithms is helpful in realizing low-cost policy generalization from simple scenarios to complex scenarios. In the future, with the development of the cognitive generalization ability of reinforcement learning, a new sim-to-real framework may be formed, which will have a positive impact on the fields of automatic driving, game agent control and robot control to reduce the cost of scenario modeling and reward system design.

② In addition, we provide a general framework for enhancing DRL methods' cognitive generalization, which provides the possibility to explore this emerging field.

③ Our framework introduces a novel clustering algorithm (GBR) into reinforcement learning and extends policy learning ability by combining GBR with standard DRL algorithms. This may lead to better communication between the machine learning community and the reinforcement learning community. Inspire researchers to use the advantages of GBR to solve key problems of DRL.

References

Abdullah, M. A., Ren, H., Ammar, H. B., Milenkovic, V., Luo, R., Zhang, M., and Wang, J. Wasserstein robust reinforcement learning. *arXiv preprint arXiv:1907.13196*, 2019a.

Abdullah, M. A., Ren, H., Ammar, H. B., Milenkovic, V., Luo, R., Zhang, M., and Wang, J. Wasserstein robust reinforcement learning. *arXiv preprint arXiv:1907.13196*, 2019b.

Amagata, D. and Hara, T. Fast density-peaks clustering: multicore-based parallelization approach. In *Proceedings of the 2021 International Conference on Management of Data*, pp. 49–61, 2021.

Azabou, M., Azar, M. G., Liu, R., Lin, C.-H., Johnson, E. C., Bhaskaran-Nair, K., Dabagia, M., Avila-Pires, B., Kitchell, L., Hengen, K. B., et al. Mine your own view: Self-supervised learning through across-sample prediction. *arXiv preprint arXiv:2102.10106*, 2021.

Beck, J., Vuorio, R., Liu, E. Z., Xiong, Z., Zintgraf, L., Finn, C., and Whiteson, S. A survey of meta-reinforcement learning. *arXiv preprint arXiv:2301.08028*, 2023.

Chandak, Y., Theocharous, G., Kostas, J., Jordan, S., and Thomas, P. Learning action representations for reinforcement learning. In *International conference on machine learning*, pp. 941–950. PMLR, 2019.

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

de Villiers, B. and Sabatta, D. Hindsight reward shaping in deep reinforcement learning. In *2020 International SAUPEC/RobMech/PRASA Conference*, pp. 1–7. IEEE, 2020.

Erraqabi, A., Zhao, M., Machado, M. C., Bengio, Y., Sukhbaatar, S., Denoyer, L., and Lazaric, A. Exploration-driven representation learning in reinforcement learning. In *ICML 2021 Workshop on Unsupervised Reinforcement Learning*, 2021.

Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pp. 226–231, 1996.

Eysenbach, B., Asawa, S., Chaudhari, S., Levine, S., and Salakhutdinov, R. Off-dynamics reinforcement learning: Training for transfer with domain classifiers. *arXiv preprint arXiv:2006.13916*, 2020.

Fan, L., Wang, G., Huang, D.-A., Yu, Z., Fei-Fei, L., Zhu, Y., and Anandkumar, A. Secant: Self-expert cloning for zero-shot generalization of visual policies. *arXiv preprint arXiv:2106.09678*, 2021.

Fu, J., Kumar, A., Nachum, O., Tucker, G., and Levine, S. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Hu, H., Liu, J., Zhang, X., and Fang, M. An effective and adaptable k-means algorithm for big data cluster analysis. *Pattern Recognition*, 139:109404, 2023.

Igl, M., Farquhar, G., Luketina, J., Boehmer, W., and Whiteson, S. Transient non-stationarity and generalisation in deep reinforcement learning. *arXiv preprint arXiv:2006.05826*, 2020.

Kaufmann, E., Bauersfeld, L., Loquercio, A., Müller, M., Koltun, V., and Scaramuzza, D. Champion-level drone racing using deep reinforcement learning. *Nature*, 620 (7976):982–987, 2023.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kirk, R., Zhang, A., Grefenstette, E., and Rocktäschel, T. A survey of zero-shot generalisation in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 76: 201–264, 2023.

Lake, B. M. and Baroni, M. Human-like systematic generalization through a meta-learning neural network. *Nature*, 623(7985):115–121, 2023.

Li, B., Tang, H., Zheng, Y., Hao, J., Li, P., Wang, Z., Meng, Z., and Wang, L. Hyar: Addressing discrete-continuous action reinforcement learning via hybrid action representation. *arXiv preprint arXiv:2109.05490*, 2021.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Mordatch, I., Lowrey, K., and Todorov, E. Ensemble-cio: Full-body dynamic motion planning that transfers to physical humanoids. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5307–5314. IEEE, 2015.

Morimoto, J. and Doya, K. Robust reinforcement learning. *Neural computation*, 17(2):335–359, 2005.

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

Slaoui, R. B., Clements, W. R., Foerster, J. N., and Toth, S. Robust visual domain randomization for reinforcement learning. *arXiv preprint arXiv:1910.10537*, 2019.

Sutton, R. S. and Barto, A. G. Reinforcement learning: An introduction. *Robotica*, 17(2):229–235, 1999.

Tanabe, T., Sato, R., Fukuchi, K., Sakuma, J., and Akimoto, Y. Max-min off-policy actor-critic method focusing on worst-case robustness to model misspecification. *Advances in Neural Information Processing Systems*, 35: 6967–6981, 2022.

van der Maaten, L. and Hinton, G. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. URL http://jmlr.org/papers/v9/vandermaaten08a.html.

Wang, M. and Deng, W. Deep visual domain adaptation: A survey. *Neurocomputing*, 312:135–153, 2018.

Whitney, W., Agarwal, R., Cho, K., and Gupta, A. Dynamics-aware embeddings. In *International Conference on Learning Representations*, 2019.

Xia, S., Liu, Y., Ding, X., Wang, G., Yu, H., and Luo, Y. Granular ball computing classifiers for efficient, scalable and robust learning. *Information Sciences*, 483:136–152, 2019.

Xia, S., Dai, X., Wang, G., Gao, X., and Giem, E. An efficient and adaptive granular-ball generation method in classification problem. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

Xia, S., Wang, G., and Gao, X. Granular ball computing: an efficient, robust, and interpretable adaptive multi-granularity representation and computation method. *arXiv preprint arXiv:2304.11171*, 2023.

Xie, J., Kong, W., Xia, S., Wang, G., and Gao, X. An efficient spectral clustering algorithm based on granular-ball. *IEEE Transactions on Knowledge and Data Engineering*, 2023.

Zisselman, E., Lavie, I., Soudry, D., and Tamar, A. Explore to generalize in zero-shot rl. *Advances in Neural Information Processing Systems*, 36, 2024.

A. Related Work

A.1. Zero-shot Generalization Reinforcement Learning

Zero-shot Generalization Reinforcement Learning assumes that expecting the agent to learn an optimized policy without sampling data from test environments. In such cases, simulators are used to ensure robustness (Morimoto & Doya, 2005) to guarantee a certain level of performance without sampling from the real system. The most straightforward way is domain randomization (Mordatch et al., 2015) where relevant parts of the simulator are randomized to make it resilient to changes. Another line of work focuses on addressing the worst-case scenarios under stochastic simulator dynamics (Abdullah et al., 2019a). Robustness can also be achieved w.r.t. actions (Abdullah et al., 2019b), that arise when certain controllers become unavailable in the real environment. Due to the set limitations, this class of methods is mainly applied to tasks such as sim2real where the training and testing scenarios are extremely similar but there are some parameter differences (e.g., background, external forces). They do not focus on cognitive generalization which is committed to improving policy generalization between scenarios with large difficulty gaps (i.e. obvious difficulty differences caused by larger maps with more branches).

A.2. Few-shot Generalization Reinforcement Learning

Few-shot Generalization Reinforcement Learning allows fine-tuning of the pre-trained agent with a limited of steps in the complete test scenarios. two orthogonal approaches have been developed to achieve generalization. The first approach called domain adaptation (Eysenbach et al., 2020) is more related to our work. These methods first assume the agent's experience in the source domain should look similar to its experience in the target domain. They learn two classifiers to distinguish transitions between the simulated and test environments and incorporate them into the reward function to account for the dynamics shift. However, most of these methods require the reward function of the test scenario to train effectively. Some even require cross-task knowledge as a guide. By contrast, cognitive generalization can improve policy generalization in single-task by reward-agnostic training.

The second approach is to achieve the stability of the policy in different scenarios training by optimizing the framework of the DRL algorithm. Such as, IDAAC (Raileanu & Fergus, 2021) decouples the optimization of the policy and value function and introduces an auxiliary loss to prevent overfitting when training across tasks which can ensure that generic features are learned. ITER (Igl et al., 2020) achieves stability of policies trained on different scenarios via knowledge distillation. Most of these methods focus on addressing policy generalization under image input differences rather than the generalization of task difficulty perspective.

B. Supplementary Proof

B.1. Proof 1

Definition 1 (DM)**.** DM is computed by dividing the number of data points by the sum of the radius s in a GB. The definition of DM is as follows:

$$u^\star = \sum_{i=1}^{m} |z_i - c|, \quad DM = \frac{u^\star}{m}. \tag{17}$$

Lemma 1. Let DMA and DMB denote the data distribution quality values for granular ball A and granular ball B, respectively, each containing m data points. If $DM_A > DM_B$, then A is sparser than B.

Proof. Let $\sigma = \frac{1}{m}\sum_{i=1}^{m} |x_i - c|$ denotes the mean radii for each granular ball. According to Eq.B.2, DM is actually the mean radii for each granular ball. Let σ_A and B denote the mean radii of A and B respectively. So, based on $DM_A > DM_B$, we have $\sigma_A > \sigma_B$. For A, if a data point is located at a distance greater than σ_A from the center of the ball, it is considered sparse. Since A has a larger average radius, more data points are expected to be located farther away from the center, leading to a sparser data distribution. Therefore, in the case of equal data points, a higher DM value indicates a sparser data distribution in the granular ball. From Lemma 1, it can be observed that the DM value can effectively measure the sparsity level of the data distribution within a granular ball.

B.2. Proof 2

Definition 2 (Weighted DM value). The definition of DM_{weight} is as follows:

$$DM_{weight} = \frac{A_1}{A} DM_{A_1} + \frac{|A_2|}{A} DM_{A_2}, \tag{18}$$

where $|A|, |A_1|$ and A_2 denote the number of data points contained within the corresponding granular-balls. If DM_A is greater than DM_{weight}, A will divide into A_1 and A_2.

Lemma 2. If $DM_A > DM_{weight}$, the volume of granular ball A is greater than the weighted sum of the volumes of granular ball A_1 and granular ball A_2 based on the number of data points.

Proof. Let's consider a d-dimensional granular ball A, A_1 and A_2 with mean radii σ_A, σ_{A_1} and σ_{A_2} respectively. $|A|$, $|A_1|$, and $|A_2|$ represent the respective number of data points in each granular ball. $V(A)$ and $[V(A1) + V(A2)]$ weight represent the volume of A and the weighted volumes of A_1 and A_2, respectively. is the gamma function.

$$\text{We have:} |A| = |A_1| + |A_2| \text{and} DM_A > DM_{weight} \tag{19}$$

$$\Rightarrow |DM_A| > \frac{|A_1|}{A} DM_{A_1} + \frac{A_2}{A} DM_{A_2} \tag{20}$$

$$\Rightarrow \sigma_A > \frac{A_1}{A} \sigma_{A_1} + \frac{A_2}{A} \sigma_{A_2} \tag{21}$$

$$\Rightarrow \sigma_A^d > \frac{A_1}{A} * \sigma_{A_1}^d + \frac{A_2}{A} * \sigma_{A_2}^d. \tag{22}$$

$$\tag{23}$$

$$\text{We have:} V(A) = \frac{\pi^{\frac{d}{2}} * \sigma_A^d}{(\frac{d}{2}+1)} \text{ and } [V(A_1) + V(A_2)]_{weight} \tag{24}$$

$$= \frac{|A_1|}{|A_2|} * V(A_1) + \frac{|A_2|}{|A|} * V(A_2) \tag{25}$$

$$= \frac{|A_1|}{|A_2|} * \frac{\pi^{\frac{d}{2}} * \sigma_A^d}{(\frac{d}{2}+1)} + \frac{|A_2|}{|A|} * V(A_2)(\frac{d}{2}+1) \tag{26}$$

$$= \pi_{d/2} * (\frac{|A_1|}{A} * \sigma_{A_1}^d + \frac{|A_2|}{|A|} * \sigma_{A_2}^d * \sigma_{A_2}^d)/(\frac{d}{2}+1) \tag{27}$$

$$\Rightarrow V(A) - [V(A_1) + V(A_2)]_{weight}. \tag{28}$$

$$= \pi^{\frac{d}{2}} * (\sigma_A^d - \frac{A_1}{A} * \sigma_{A_1}^d - \frac{A_2}{A} * \sigma_{A_2}^d)/(\frac{d}{2}+1) \tag{29}$$

$$= (\sigma_A^d - \frac{A_1}{A} * \sigma_{A_1}^d) > 0. \tag{30}$$

Therefore, the numerator in the above equation is positive. Furthermore, since all the values involved are positive, we can see that the denominator $(\frac{d}{2}+1)$ is positive as well. Hence $V(A)[V(A1) + V(A2)]_{weight}$ is positive, implying that the volume of A is greater than the weighted sum of the volumes of A_1 and A_2 based on the number of data points.

From Lemma 2, it can be seen that splitting can reduce the amount of blank space within a granular ball, thereby providing a better characterization of the dataset.

C. Experimental Details

C.1. Network Structure

Our codes are implemented with Python 3.7.9 and Torch 1.7.1. All experiments were run on a single NVIDIA GeForce GTX 2080Ti GPU. Each single training trial ranges from 4 hours to 17 hours, depending on the algorithms and environments. Our TD3 is implemented with reference to github.com/sfujim/TD3 (TD3 source-code). DDPG and PPO are implemented

Layer	Actor Network	Critic Network
Fully Connected	(state dim, 256)	(statedim + η dim + latent space dim, 128)
Activation	ReLU	ReLU
Fully Connected	(256, 128)	(256, 128)
Activation	ReLU	ReLU
Fully Connected	(128,action dim)	(128, 1)
Activation	Tanh	None

Table 3. Network Structures for DRL Methods

with reference to https://github.com/sweetice/Deep-reinforcement-learning-with-pytorch. For a fair comparison, all the baseline methods have the same network structure (except for the specific components of each algorithm) as our MARS-TD3 implementation. As shown in Tab.3, we use a two-layer feed-forward neural network of 256 and 256 hidden units with ReLU activation (except for the output layer) for the actor network for all algorithms. For DDPG the critic denotes the Q-network. For PPO, the critic denotes the V-network. All algorithms (TD3, DDPG, PPO) output two heads at the last layer of the actor network, one for latent action and another for dynamic transition potential.

Model Component	layer	dimension
Conditional Encoder Network	Fully Connected (encoding)	$(\mathbb{R}^x, 256)$
	Fully Connected	$(256, 256)$
	Activation	ReLU
	Fully Connected (mean)	(256, latent space dim)
	Activation	None
	Fully Connected (log std)	(256, latent space dim)
	Activation	None
Conditional Decoder, Prediction Network	Fully Connected (latent)	(latent space dim, 256)
	Fully Connected	(256, 256)
	Activation	ReLU
	Fully Connected (reconstruction)	(256, action dim)
	Fully Connected (reconstruction)	(256, state dim)
	Activation	None
	Fully Connected	$(256, 256)$
	Activation	ReLU
	Fully Connected (prediction)	(256, state dim)
	Activation	None

Table 4. Network structures for the Multi-step action representation (MARS).

The structure of our VAE is shown in Tab.4. We use element-wise product operation (Mnih et al., 2013) and cascaded head structure (Kaufmann et al., 2023) to our model.

C.2. Hyperparameter

For all experiments, we use the raw state and reward from the environment, and no normalization or scaling is used. No regularization is used for the actor and the critic in all algorithms. An exploration noise sampled from N(0, 0.1) (Kirk et al., 2023) is added to all baseline methods when selecting an action. The discounted factor is 0.99 and we use Adam Optimizer (Lake & Baroni, 2023) for all algorithms. Tab.5 shows the common hyperparameters of algorithms used in all our experiments.

Hyperparameter	TD3-frameskip	TD3-advance	MARS-PPO	MARS-TD3	MARS-DDPG	
Actor Learning Rate	$1e^{-4}$	$1e^{-4}$	$1e^{-4}$	$1e^{-4}$	$1e^{-4}$	$1e^{-4}$
Critic Learning Rate	$1e^{-3}$	$1e^{-3}$	$1e^{-3}$	$3e^{-4}$	$3e^{-4}$	$1e^{-3}$
Representation Model Learning Rate	None	None	None	$1e^{-4}$	$5e^{-3}$	$5e^{-3}$
Discount Factor	0.99	0.99	0.99	0.99	0.99	0.99
Batch Size	128	128	128	128	128	128
Buffer Size	$1e5$	$1e5$	$1e5$	$1e5$	$1e5$	$1e5$

Table 5. A comparison of common hyperparameter choices of algorithms. We use ‘None’ to denote the ‘not applicable’ situation.

However, GBRL relies on GBR clustering, which results in the need to tune GBR hyperparameters on different tasks to ensure policy performance. Tab.6 presents the minimum number of samples η within a GB and the maximum number of neighbors k that we used for different tasks.

Hyperparameter	2dMAZE	Ninja	Chaser	Heist	Dodgeball	Leaper	CaveFlyer	Miner
η	10	25	10	15	30	25	10	10
k	5	10	5	5	8	5	8	8

Table 6. A comparison of common hyperparameter choices of algorithms. We use 'None' to denote the 'not applicable' situation.

C.3. Additional Implementation Details

For PPO, the actor network and the critic network are updated every 2 and 10 episodes respectively for all environments. The clip range of the PPO algorithm is set to 0.2 and we use GAE (Slaoui et al., 2019) for a stable policy gradient. For DDPG, the actor network and the critic network is updated at every 1 environment step. For TD3, the critic network is updated every 1 environment step and the actor network is updated every 2 environment steps.

The default latent action dim for 2dMAZE is 5, and the latent action dim for Procgen is 15, We set the KL weight in representation loss L_{VAE} as 0.5. Environment dynamic prediction loss weight β is 5 (default).

C.4. Environments

In this paper, we use two open source Benchmarks. 2DMAZE in D4RL (website:https://github.com/Farama-Foundation/D4RL) and seven environments from the Procgen (detailed description can be seen in https://github.com/openai/Procgen) which is specifically designed to measure generalization by introducing separate test- and training levels. Each environment supports two well-calibrated difficulty settings: easy and hard. Agents are trained on an easy level and tested on a hard level. See Figure 7 for a brief description of the task.

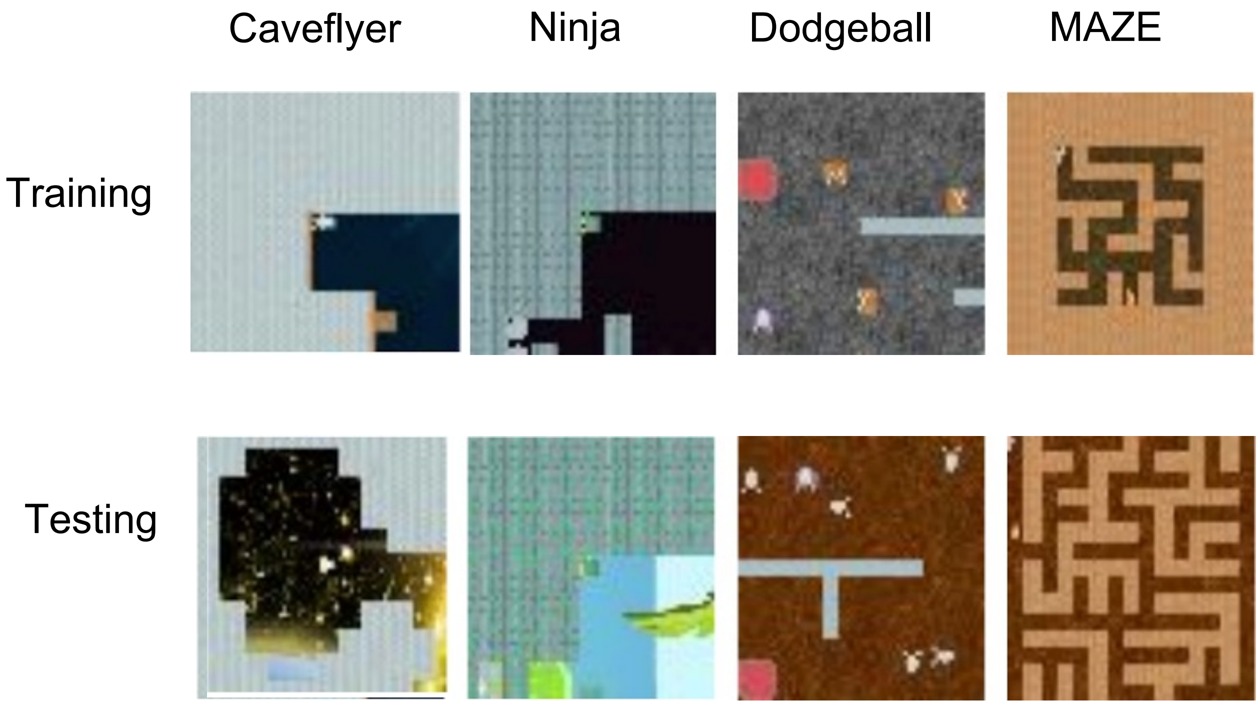

Figure 6. Brief description of the training and testing task.

- 2DMAZE. We use 2DMAZE-medium for training and 2DMAZE-Hard for testing. **The difference between the two maps is that the difficult scenario map is larger, has more branches, and the strategy needs more steps to reach the target.**
- Caveflyer. The player must navigate a network of caves to reach the exit. The player movement mimics the Atari game "Asteroids": the ship can rotate and travel forward or backward along the current axis. The majority of the reward comes from successfully reaching the end of the level, though additional reward can be collected by destroying target objects along the way with the ship's lasers. There are stationary and moving lethal obstacles throughout the level. **The difference between the two maps is that the difficult scenario map is larger, and has more lethal obstacles.**
- Chaser. Inspired by the Atari game "MsPacman". Maze layouts are generated using Kruskal's algorithm, and then walls are removed until no dead-ends remain in the maze. The player must collect all the green orbs. 3 large stars spawn that will make enemies vulnerable for a short time when collected. A collision with an enemy that isn't vulnerable results in the player's death. When a vulnerable enemy is eaten, an egg spawns somewhere on the map that will hatch into a new enemy after a short time, keeping the total number of enemies constant. The player receives a small reward for collecting each orb and a large reward for completing the level. **The difference between the two maps is that the difficult scenario map has more enemies.**
- Dodgeball. Loosely inspired by the Atari game "Berzerk". The player spawns in a room with a random configuration of walls and enemies. Touching a wall loses the game and ends the episode. The player moves relatively slowly and can navigate throughout the room. There are enemies that also move slowly and which will occasionally throw balls at the player. The player can also throw balls, but only in the direction they are facing. If all enemies are hit, the player can move to the unlocked platform and earn a significant level completion bonus. **The difference between the two maps is that the difficult scenario map has more enemies and branches.**
- Heist. The player must steal the gem hidden behind a network of locks. Each lock comes in one of three colors, and the necessary keys to open these locks are scattered throughout the level. The level layout takes the form of a maze, again generated by Kruskal's algorithm. Once the player collects a key of a certain color, the player may open the lock of that color. All keys in the player's possession are shown in the top right corner of the screen. **The difference between the two maps is that the difficult scenario map has more doors and keys to collect.**
- Miner. Inspired by the classic game "BoulderDash". The player, a robot, can dig through dirt to move throughout the world. The world has gravity, and dirt supports boulders and diamonds. Boulders and diamonds will fall through free space and roll off each other. If a boulder or a diamond falls on the player, the game is over. The goal is to collect all the diamonds in the level and then proceed through the exit. The player receives a small reward for collecting a diamond and a larger reward for completing the level. **The difference between the two maps is that the difficult scenario map is larger and has more branches.**
- Ninja. A simple platformer. The player, a ninja, must jump across narrow ledges while avoiding bomb obstacles. The player can toss throwing stars at several angles in order to clear bombs, if necessary. The player's jump can be charged over several timesteps to increase its effect. The player receives a reward for collecting the mushroom at the end of the level, at which point the episode terminates. **The difference between the two maps is that the difficult scenario map is larger and has more bomb obstacles.**

D. Additional Experiments

D.1. Empirical Verification for Reward Function Design in Simple Tasks

Proposition: In simple simulation environments with the same task logic, the best reward functions change depending on the scene size, border shape, and the distribution of objects. Adjusting the reward function requires multiple manual debugging to add auxiliary items and balance various factors in the function which can be difficult and costly.

Experimental verification. We use minigrid to construct a set of navigation tasks with the same goal(Figure **??**): Agent must take the key to open the door and go to the goal point. The observation space of each scenario is the same, but the scene scale, boundary shape, and obstacle placement are different. Scenario A is a simple task (square boundary), scenario B (square boundary) and scenario C (rectangle boundary) are complex tasks.

Firstly, we define a seemingly general reward function **R1** to guide PPO on the three scenarios separately from scratch. The first row in Table 1 indicates that although PPO performs well in scenario A, PPO falls into sub-optimal in scenarios B and C.

However, when we adjust the reward function to add the penalty item for the agent hitting the wall and manually adjust the penalty degree (i.e. R2), the results in the second row of Table.7 show that the performance of PPO in scenarios B and C has been significantly improved.

Furthermore, we adjust the balance weight of each factor in the reward function (i.e. R3,R4) by consuming more time cost. Results in line 3.4 of the table show that scenario B is more suitable for R3(emphasizing the importance of finding the key) and scenario C is more suitable for R4 (emphasizing the goal reward). This indicates that the optimal reward function for different scenarios needs to be found by adjusting the weights in the function.

When we fine-tune PPO with R1 which is trained in scenario A through GBRL, PPO can achieve similar performance to the policy by a carefully designed reward function.

Conclusion: (1) The above experiments show that even in relatively simple scenarios, effective reward function design can be difficult and costly (2) GBRL shows an effective reward-agnostic fine-tuning way, which avoids the reward design of complex scenarios and realizes the effective generalization of the policy.

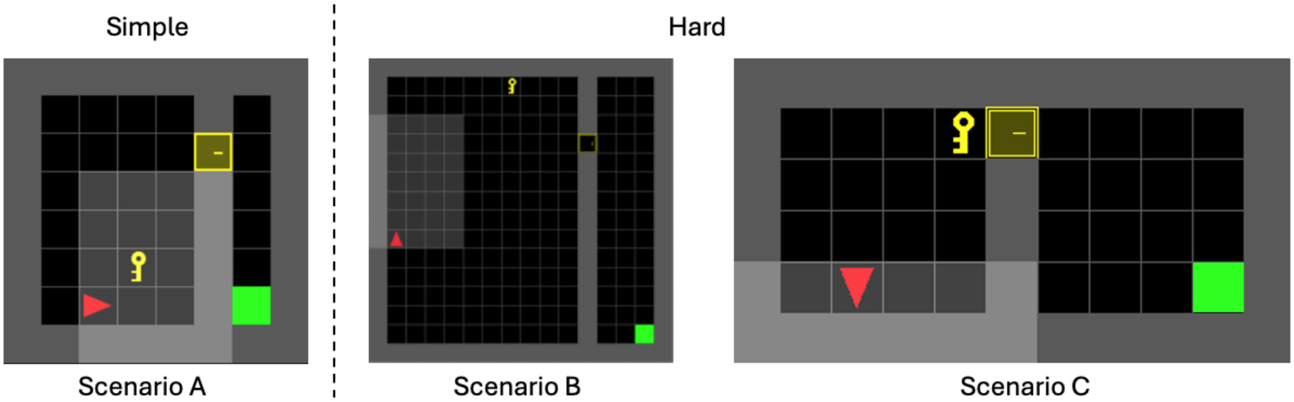

Figure 7. Three tasks with same task logic.

$R1 : R(s,a,s') = R_{goal} + R_{wall} + R_{step}$

$R2 : R(s,a,s') = R_{goal} + R_{wall} + R_{step} + R_{key}$

$R3 : R(s,a,s') = 3 \times R_{goal} + \times R_{step} + 4 \times R_{key}$

$R4 : R(s,a,s') = 2 \times R_{goal} + 2 \times R_{step} + 3 \times R_{key}$

R_{goal} : if s' is the goal state $R_{goal} = 1$ else $R_{goal} = 0$

R_{wall} : if s' is a wall state $R_{wall} = -0.1$ else $R_{wall} = 0,$

Task	**Scenario A** (%)	**Scenario B** (%)	**Scenario C** (%)
PPO with R1	60	0	20
PPO with R2	70	30	70
PPO with R3	70	60	70
PPO with R4	70	30	90
GBRL-PPO without reward	60	50	80

Table 7. Win rate (Average of 10 runs).

R_{step} : for all time steps if agent is closer to the goal $R_{step} = 0.1$,

R_{key} : if s' is the key state $R_{key} = 0.5$ else $R_{key} = 0$

D.2. Hyperparameter Sensitivity Analysis

We found that GBRL is sensitive to two hyperparameters. And the optimal parameters corresponding to different tasks are different, but the optimal range can be determined: $\eta : [10, 30]$, $k : [5, 10]$. Tab.9 shows the effect of different minimum capacities η and maximum number of neighbors k on policy generalization in Procgen Tasks. Change the hyperparameter when other parameters are set to the optimal premise.

Tasks	η =5	10	15	20
Miner	9.06 ± 1.37	11.07 ± 2.97	10.14 ± 1.87	9.75 ± 0.91
Heist	4.71 ± 0.31	4.96 ± 0.45	5.25 ± 0.53	4.25 ± 0.93
Leaper	4.57 ± 1.26	4.26 ± 0.89	4.36 ± 1.04	5.14 ± 0.53

Table 8. A comparison of different η in Procgen tasks. Scores are averaged over 5 runs.

Tasks	k =4	6	8	10	12
Miner	6.96 ± 3.08	7.15 ± 2.47	10.44 ± 1.32	11.07 ± 2.97	10.82 ± 1.44
Heist	3.91 ± 1.53	4.26 ± 1.71	4.45 ± 0.77	5.25 ± 0.53	4.61 ± 1.28
Leaper	5.14 ± 0.53	5.07 ± 1.96	4.96 ± 0.59	4.31 ± 1.12	4.15 ± 1.55

Table 9. A comparison of different k in Procgen tasks. Scores are averaged over 5 runs.

D.3. Additional Experiments for RQ2

To prove the generality of VAE for different difficulty scenarios, we randomly collect 50 samples from Three tasks (each one has easy and hard map) as the input of the trained VAE, to see whether our VAE can correctly reconstruct the samples. Results in Tab.10 show that our VAE achieves $> 82\%$ reconstruction accuracy for all tasks.

Task	**Easy**	**Hard**	**Average**
Chaser	85.2%	80.1%	82.65%
Heist	84.6%	87.4%	86.00%
Miner	82.7%	86.9%	84.80%

Table 10. The evaluations of reconstruction accuracy.

D.4. Experiments and Analysis for Cascaded Prediction Module

Our structure choice of cascaded prediction head is inspired by MYOW (Azabou et al., 2021). The reason behind this is that dynamics prediction could be more complex than continuous action reconstruction, thus usual parallel heads for both reconstruction and the state residual prediction followed by the same latent features may have interference in optimizing individual objectives and hinder the learning of the shared representation.

We carry out comparative experiments to examine how cascaded and parallel prediction (usually used) affect latent space construction without altering other elements. We test these two architectures on three ProcGen environments and note the policy (DDPG)scores in Tables 11.

Task	Ninja	Chaser	Heist
VAE with cascaded prediction	6.11 ± 0.16	3.09 ± 0.13	5.27 ± 0.08
VAE with parallelized prediction	5.85 ± 0.32	2.93 ± 0.08	4.15 ± 0.11

Table 11. Performance of DDPG (average of 5 runs).

Furthermore, we employ t-SNE to visualize the latent spaces created by two VAE architectures from Chaser, the result shows cascaded prediction can build a semantically smoother latent space. Please click this anonymous link `https://anonymous.4open.science/r/ICML_reviewer_2-48E8/VAE_arch.png`.

D.5. Experiments and Analysis for clustering algorithms

Task	Ninja	Chaser	Heist
GBR	6.16 ± 0.11	3.27 ± 0.08	3.27 ± 0.08
K-means	5.22 ± 0.43	2.86 ± 0.31	2.71 ± 0.25
DBSCAN	5.79 ± 0.26	2.53 ± 0.28	3.04 ± 0.11
DPC	4.82 ± 0.51	2.61 ± 0.41	3.10 ± 0.14

Table 12. The evaluations of cluster methods (average of 5 runs).

Specifically, there are two defects between these methods and GBR, which lead to their poor performances on our tasks: (1) The number of classes and other complex hyper-parameters (e.g., neighborhood radius, cut-off distance) need to be artificially defined in advance to ensure the efficiencies of the algorithms. However, these prior parameters are difficult to obtain or predict in the latent space due to the fuzziness of the dynamic semantics of samples in the environment. Besides, it is hard to accurately judge the number of sample classes in an unseen scenario. (2) While the granularity of previous clustering methods is relatively single, GBR can adaptively generate different sizes of balls to fit the multi-modal distribution and construct a better cognitive latent space.

To further analyze the effectiveness of GBR in our framework, we select three popular clustering algorithms as baselines, i.e. K-means (Hu et al., 2023), DBSCAN (Ester et al., 1996), and DPC (Amagata & Hara, 2021). We evaluate GBR and three SOTA baselines in three scenarios. To make a comprehensive comparison of methods, we evaluate the performance of policies and the convergence speed in complex scenes. To ensure a fair comparison, we use the optimal hyperparameter setting (determined by grid search) for all baselines and uniformly use DDPG as the policy. The architecture remains the same except for cluster methods. We ensure that all methods training is completed.

Table.12 shows that GBR outperforms the other baselines by 15% on average on the three ProcGen tasks.

E. Pseudocode of GBR

Algorithm 2 Generation of Granular Balls

```
Input $D^\star$
Output $GBS$
Initialize $GBS = \{\}$
Firstly, we consider the entire $D^\star$ as a $GB$, push $GB$ into $GBS$;
while $true$ do
    for $GB_i$ in $GBS$ do
        calculate $DM_{GB_i}$ and $DM_{weight}$ according to Eq.10-13
        if $DM_{weight} \geq DM_{GB_i}$ thenSplit $GB_i$
        end if
    end for
    if the number of GB is not changing thenbreak
    end if
end while
Return $GBS$
```

Algorithm 3 Merging of Granular-Balls

```
Get $GBS$ from Alg.2
while $true$ do
    for $GB_i$ and $GB_j$ $(i \neq j)$ in $GBS$ do
        Determine whether $GB_i$ and $GB_j$ overlap according to Eq.14
        if $GB_i$ and $GB_j$ overlap thenReplace them with $GB_i \cap GB_j$
        end if
    end for
    if the number of $GB$ in $GBS$ remains the same thenbreak
    end if
end while
Return $GBS$
```