Deep Reinforcement Learning with Vector Quantized Encoding

Anonymous authors
Paper under double-blind review

Abstract

Human decision-making often involves combining similar states into categories and reasoning at the level of the categories rather than the actual states. Expanding on this notion, we introduce a novel approach to augment the interpretability, robustness, and generalization of deep reinforcement learning (RL) models through state feature clustering. Our method, termed *Vector Quantized Reinforcement Learning* (VQ-RL), integrates an auxiliary classification task using vector quantized (VQ) encoding into conventional RL pipelines, improving the model's capability to group similar states into clusters with enhanced separation. Additionally, we propose two regularization techniques to bolster cluster separation and alleviate potential risks associated with VQ training. Our simulations demonstrate that VQ-RL enhances our understanding of the internal space of deep RL, as well as the robustness and generalization capabilities of state-of-the-art RL methods across various domains.

1 Introduction

Human thinking reflects clustering characteristics in many modalities such as reasoning and planning. For example, consider the CartPole game (Barto et al., 1983), in which an agent controls a cart with a pole balanced vertically on it, moving the cart left or right as necessary to keep the pole from falling. Humans do not assess each state precisely like computers do before outputting corresponding actions; instead, they cluster similar states into categories and respond accordingly, as illustrated in Figure 1. This observation motivates us to explore the clustering properties of features within deep reinforcement learning (RL) models. Our goal is to better understand the internal dynamics of deep RL and investigate whether using these properties can enhance the robustness and generalization of such models.

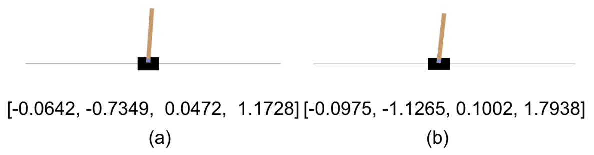


Figure 1: Example states in CartPole. The vector below each figure represents the state, with the components representing the following continuous features: cart position (CP), cart velocity (CV), pole angle (PA), and pole angular velocity (PAV). Each component is a continuous decimal value. In (a) and (b), the pole is tilting to the right, so the player will need to move the cart to the right to keep the pole upright.

Vector quantized variational autoencoders (VQ-VAE) (Oord et al., 2017) are a family of generative models that combine the variational autoencoder (VAE) framework with discrete latent representations through a parameterization of the posterior distribution. So far, most works on VQ-VAEs have focused on generative applications, such as generating images, audio, video, etc. (Oord et al., 2017; Gârbacea et al., 2019; Razavi et al., 2019; Tjandra et al., 2020). However, this encoding and training process can also generate an embedding space with improved clustering properties —i.e., (i) features within the same cluster are more tightly grouped, and (ii) there is greater separation between different clusters. Furthermore, according to the visualizations of t-distributed stochastic neighbor integration (t-SNE) (Van der Maaten & Hinton, 2008) in Mnih et al. (2015) and Zahavy et al. (2016), features with the same or similar semantics are more likely to cluster together in the embedding space with dimensionality reduction. These properties provide the potential to implement our proposed clustering intuition in this study.

To apply this idea to RL decision-making processes, we introduce a novel learning pipeline, named *vector quantized reinforcement learning* (VQ-RL) in this work. In our experiments, we assess the effectiveness of VQ-RL, examining whether the enhanced clustering results in improved interpretability, robustness, and generalization. In summary, we make the following contributions.

- Inspired by the clustering nature of human reasoning, we propose a novel multi-task learning framework based on VQ encoding, and show that the framework results in improved clustering properties.

- We introduce two regularization methods to further improve the separation between clusters in the VQ encoding space and avoid the risk in the codebook training that may be caused by large values of embedding vectors.

- We conduct three sets of experiments using state-of-the-art RL methods across various domains to showcase the effectiveness of our proposed architecture in improving interpretability, robustness, and generalization.

2 Related Work

Although deep neural networks have excelled in numerous RL problems (Levine et al., 2016; Lee et al., 2019a; Jaderberg et al., 2019; Silver et al., 2016), their inherent black-box nature complicates the interpretation of decision-making processes. Moreover, achieving human-level robustness and generalization remains a formidable challenge. Consequently, there is a growing research effort focused on proposing more interpretable, robust, and generalized methods for RL.

Interpretability Programmatically Interpretable Reinforcement Learning (PIRL) (Verma et al., 2018) represents policies using a high-level, domain-specific programming language designed to generate interpretable and verifiable agent policies. To improve the interpretability of the subtasks in hierarchical decision-making, Lyu et al. (2019) introduced symbolic planning into RL and proposed the Symbolic Deep Reinforcement Learning (SDRL) framework that can handle both high-dimensional sensory inputs and symbolic planning. A soft attention model for RL domains was introduced in Mott et al. (2019) to show that the model learns to query separately about space and content (“where” vs. “what”). Such et al. (2018), Wang et al. (2018), and Zahavy et al. (2016) proposed visualization techniques for feature space for better interpretability of RL methods, showing that features with similar semantics are located closely.

Robustness and generalization An approach for robustness to action uncertainty was proposed in Tessler et al. (2019), which provided a robust policy learning method and improved performance in the absence of perturbations. To increase robustness to noise and adversaries, Lütjens et al. (2020) introduced an online certified defense to the Deep Q-Network policy training. Wang et al. (2020) studied deep RL models in noisy reward scenarios and developed a robust framework for these scenarios. Furthermore, Cobbe et al. (2019) introduced CoinRun, a benchmark for generalization in RL, and studied factors in neural network training that may affect generalization. Packer et al. (2018) proposed a benchmark and experimental protocol and conducted a systematic empirical study of generalization. Various neural network training techniques have also been proposed in the literature to improve generalization in RL (Lee et al., 2019b; Zhang et al., 2018; Igl et al., 2019).

3 Method

In this section, we explain our methodology and architecture, which builds on the VQ-VAE framework. Here we provide a brief overview of the relevant VQ-VAE equations and concepts. The VQ-VAE model starts with an encoder network that maps an input x to a continuous latent representation $E(x)$. Following this, $E(x)$ is quantized via the nearest neighbor look-up in the codebook, $\mathbf{e} \in \mathbb{R}^{K \times D}$, where K is the number of vectors in the codebook and D is the dimension of the continuous embedding. The K embedding vectors are represented as an index of the codebook, $\mathbf{e}_i \in \mathbb{R}^D, i \in 1 \dots K$. This nearest neighbor vector is then fed into the decoder network to reconstruct the input x. The following equation summarizes the process of the quantization of VQ encoding:

$$\begin{gathered} \text{Quantize}\left(E\left(\mathbf{x}\right)\right) = \mathbf{e}_k, \\ k = \arg\min_j \|E(\mathbf{x}) - \mathbf{e}_j\| \end{gathered} \tag{1}$$

The loss function for VQ-VAE training is as follows:

$$\mathcal{L}(\mathbf{x}, D(\mathbf{e})) = \|\mathbf{x} - D(\mathbf{e})\|_2^2 + \|sg[E(\mathbf{x})] - \mathbf{e}\|_2^2 + \beta\|sg[\mathbf{e}] - E(\mathbf{x})\|_2^2 \tag{2}$$

where E is the encoder and D is the decoder, sg is a stop-gradient operator, and β is the weight preventing the encoder outputs from fluctuating between different code vectors.

3.1 VQ-RL

Deep RL architectures usually contain two modules: a feature extractor and a policy network, whose designs vary according to the specific requirements of the problems being studied (Bellemare et al., 2017; Kaiser et al., 2019; Arulkumaran et al., 2017). To improve the extracted features' clustering properties in the embedding space, we introduce an auxiliary task with VQ encoding to the classic RL pipeline. This auxiliary task aims to predict the most probable output action under the current policy by performing a classification task and will lead to clustering together features with the same maximum possible output, such as the example in Figure 1. The architecture of our model is shown in Figure 2, and the visualization of vector quantizer is shown in the appendix.

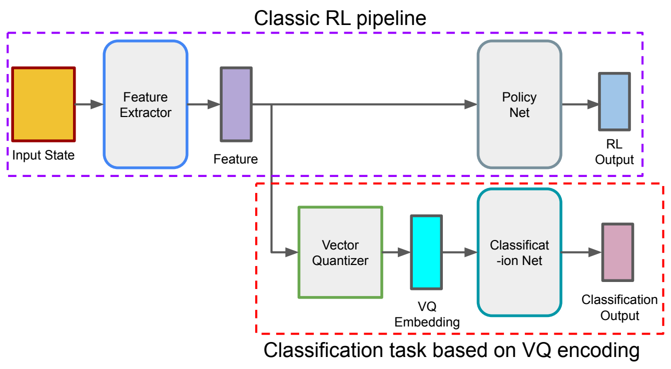


Figure 2: Overview of the VQ-RL architecture. VQ-RL adds an auxiliary classification task based on VQ encoding to the classic RL pipeline. The vector quantizer re-encodes features, returning the closest embedding from the codebook as the output. Based on the VQ embedding, the classification net outputs the action with the highest probability from the current policy.

The training process in VQ-VAE causes two effects: (i) the encoded features learn to be as close as possible to the closest embedding vector in the codebook, and (ii) the embedding vectors in the codebook learn to be as close as possible to all the features in their corresponding cluster. Theoretically, VQ encoding can help achieve clustering properties without performing additional unsupervised tasks. Similar to the encoding process in VQ-VAE, the vector quantizer in VQ-RL returns (i) the embedding in the codebook as the output that is closest to the input feature and (ii) the encoding index identifying the cluster.

Nevertheless, in the reconstruction task of VQ-VAE, features are segmented into multiple parts and VQ encoding is processed for each segment separately. For example, in the image reconstruction task in VQ-VAE, a 32×32 embedding matrix with $K = 512$ for ImageNet is required. In order to cluster features in VQ-RL, we set the embedding vector dimension in the codebook to be the same as the input feature size.

The most straightforward choice for the number of VQ embeddings in the codebook is the number of possible actions, e.g., CartPole contains only two performable actions—thus, for this domain, we would simply set the number of VQ embeddings to two. However, in practice, the human brain may further divide the state space into more regions than there are possible primitive actions in the domain. For instance, in CartPole, although it is necessary to perform the 'move left' action both when the pole is slightly to the left and very far to the left, we tend to distinguish them into two types of states since it affects future actions. Therefore, we set the number of embeddings to be larger than the number of performable actions in the experiments to further explore the embedding space's internal status in greater detail and to better emulate human reasoning.

3.2 Regularization and loss function

To improve the robustness of the codebook learning, we introduce and modify two forms of regularization from François-Lavet et al. (2019) into our training. The first, $\mathcal{L}_{d1}$ is defined as follows:

$$\mathcal{L}_{d1} = \sum_{i \neq j} \exp\left(-C_d \left\|\mathbf{e}_i - \mathbf{e}_j\right\|_2\right) \tag{3}$$

where C_d is a constant. In contrast to the form in François-Lavet et al. (2019) which considers random pairs, we calculate the sum of the loss for all possible pairs in the codebook. This prevents any candidate embedding vectors from getting too close to each other.

Similar to the prior used in the original VAE (Kingma & Welling, 2013), we penalize the values of embedding vectors out of an L_∞ ball of radius 1. This prevents the risk that may be caused by large values of embedding vectors and is expressed as:

$$\mathcal{L}_{d2} = \max_i \left(\|\mathbf{e}_i\|_\infty^2 - 1, 0\right) \tag{4}$$

The total regularization loss is then:

$$\mathcal{L}_{\text{reg}} = \mathcal{L}_{d1} + \mathcal{L}_{d2} \tag{5}$$

We define our VQ encoding loss function as the sum of the encoding loss function from (2) and regularization loss from (5) scaled by a constant factor, λ_{reg}:

$$\mathcal{L}_{\text{vq-enc}} = \|sg[E(\mathbf{x})] - \mathbf{e}\|_2^2 + \beta\|sg[\mathbf{e}] - E(\mathbf{x})\|_2^2 + \lambda_{\text{reg}}\mathcal{L}_{\text{reg}} \tag{6}$$

Our proposed auxiliary task is a typical classification problem, and so we use Softmax cross entropy loss, given as $\mathcal{L}_{\text{class}}$, for the total training loss. Which is given as the following equation:

$$\mathcal{L}_{\text{class}} = -\sum_{i=1}^{N_a} a_i \log\left(\sigma(C(e))\right) \tag{7}$$

where σ is the softmax function, a_i is the action, N_a is the number of possible actions, and C is the classification net.

Thus, the total objective of our VQ-RL training can be written as:

$$\mathcal{L}_{\text{vq-rl}} = \mathcal{L}_{\text{rl}} + \lambda_{\text{vq-enc}}\mathcal{L}_{\text{vq-enc}} + \lambda_{\text{class}}\mathcal{L}_{\text{class}} \tag{8}$$

where $\mathcal{L}_{\text{rl}}$ is the training loss from the corresponding RL algorithm, $\mathcal{L}_{\text{class}}$ is the classification loss, *i.e.*, Softmax cross-entropy loss, $\lambda_{\text{vq-enc}}$ is the weight of the VQ encoding loss, and λ_{class} is the weight of the classification loss. $\mathcal{L}_{\text{vq-enc}}$ gathers the features extracted by the feature extractor around the VQ embeddings and ensures a nontrivial distance between the embeddings. $\mathcal{L}_{\text{class}}$ leads the training of VQ encodings to align with policy learning. Therefore, all these endow VQ-RL with the potential to understand states better and improve the generalizability and robustness of RL models. We will present our results of testing this hypothesis in the next section.

4 Simulations

In this section, we investigate three aspects of VQ-RL: interpretability, robustness, and generalization, in three domains: CartPole, Acrobot (Sutton, 1995), and LunarLander (Brockman et al., 2016). We employ state-of-the-art RL algorithms, namely DQN (Mnih et al., 2015), A2C (Mnih et al., 2016), and PPO (Schulman et al., 2017), for our analysis. Details regarding hyperparameter settings, neural network designs, training procedures, and additional domain tests are provided in the appendix.

4.1 CartPole

Our analysis compared three different approaches: (i) DQN using our VQ-RL framework (which we call VQ-DQN), (ii) VQ-DQN with our two proposed regularization methods (which we call VQ-DQN-Reg), and (iii) the original DQN algorithm.

Interpretability To investigate interpretability we explore the embedding spaces of the three methods. Specifically, we randomly select 2,000 legal states from the state space of CartPole, and then use the three feature extractors of trained models to obtain the corresponding features. Following this, we perform principal component analysis (PCA) on those 2,000 features. The visualization of the PCA results are presented in Figure 3 and Figure 4. Figure 3 shows that the features obtained by DQN are scattered across a relatively large space. There is no distinct boundary between features. However, as depicted in Figure 4, due to the tight clustering of features around their respective embeddings, a closer inspection is required to discern the internal distribution. These findings align with the hypothesis proposed in the Introduction, suggesting that the VQ-RL framework enhances the clustering properties of features.

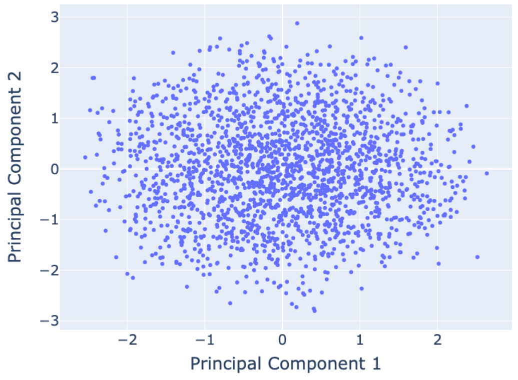
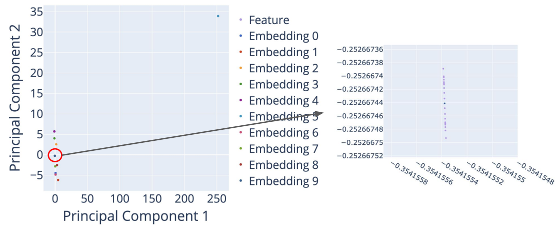
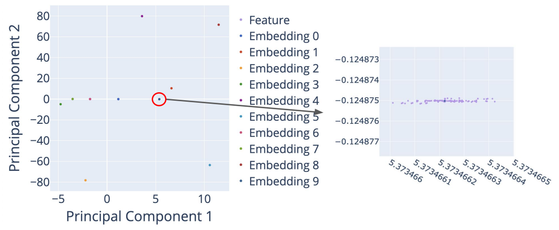


Figure 3: PCA visualization of features learned using DQN on the CartPole-v1 domain.



(a) PCA visualization of VQ-DQN



(b) PCA visualization of VQ-DQN-Reg

Figure 4: Visualizations of features and embeddings for the CartPole-v1 domain using VQ-DQN and VQ-DQN-Reg. The right part of each sub-figure zooms in on the point representing Embedding 9.

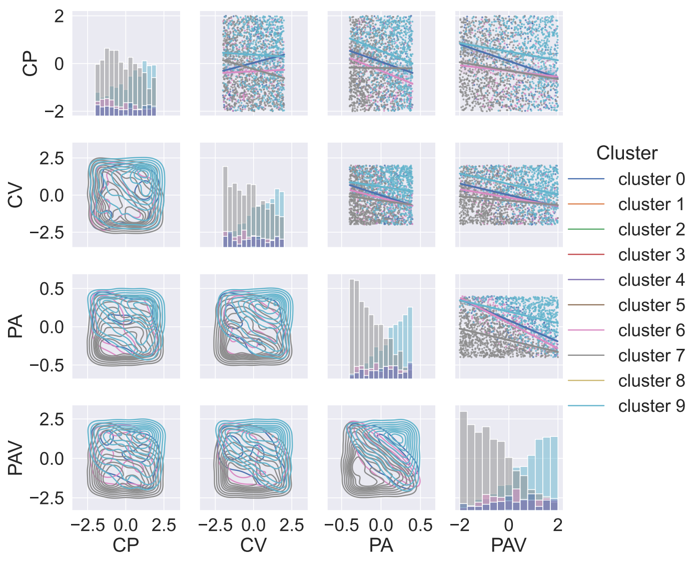


Figure 5: The state clusters for VQ-DQN-Reg in the CartPole domain. Plots above the diagonal show the distribution of clusters and the corresponding linear regression lines, plots along the diagonal are histograms, and plots below the diagonal are contour plots of the Gaussian kernel density estimate (KDE) plots of clusters. We see that PA and PAV are the primary state components that distinguish the clusters. In each state component, each cluster presents a continuous distribution in a specific interval.

Figure 5 illustrates the clustering of the 2,000 states of VQ-DQN-Reg. Since VQ-RL clusters states, we can gain further insight into the internal information of deep models in decision-making, such as the priority of state components, and the distribution of states in clusters.

Robustness In this experiment, we analyze how noise affects VQ-RL. This is common in real world applications of RL. For example, in the driving of unmanned vehicles, rain, and snow will produce noise in the visual input. Therefore, we need to demonstrate the method's robustness to noise to ensure the safety of RL technology. Specifically, we add normally distributed noise $n \sim \mathcal{N}(0, 1)/\delta$ to the input states of CartPole, where δ is a scaling factor to control the level of noise. We test the robustness of the three methods on 20 episodes with random initial states. The results for different levels of noise are shown in Table 1. The VQ-RL framework significantly improves the robustness of DQN, with the regularized VQ-DQN performing better than VQ DQN without regularization.

δ	DQN	VQ-DQN	VQ-DQN-Reg
200	**500.0 ± 0.0**	**500.0 ± 0.0**	**500.0 ± 0.0**
150	487.15 ± 56.01	**500.0 ± 0.0**	**500.0 ± 0.0**
100	395.25 ± 183.21	470.1 ± 89.73	**482.45 ± 76.50**
90	395.3 ± 167.32	464.7 ± 87.76	**484.45 ± 67.78**
80	311.0 ± 189.07	468.85 ± 87.23	**470.95 ± 88.13**
70	211.40 ± 193.87	393.7 ± 140.32	**411.45 ± 131.24**

Table 1: Results for the noisy CartPole-v1 test.

Generalization To study the generalizability of VQ-RL, we propose a modified version of CartPole and name it Gen-CartPole. In this environment, we consider changing three game-related parameters that affect the game's transition function between states: the mass of the cart (m_c), the mass of the pole (m_p), and the length of the pole (l). In each episode in training, the parameter values will be randomly selected from among the three training sets given in Table 2. It is worth noting that we take a window of the last four consecutive states as input states for models, in order to recognize the current dynamics. After completing training, we test the performance of the three models on the test sets in the bottom of Table 2 which also shows their results. We see that VQ-RL performs better than the original DQN.

(m_c, m_p, l)	DQN	VQ-DQN	VQ-DQN-Reg
(0.5, 0.05, 0.25)	123.55 ± 20.18	450.95 ± 99.20	$\mathbf{500.0 \pm 0.0}$
(0.75, 0.075, 0.375)	451.80 ± 89.17	$\mathbf{500.0 \pm 0.0}$	$\mathbf{500.0 \pm 0.0}$
(0.95, 0.095, 0.45)	$\mathbf{500.0 \pm 0.0}$	$\mathbf{500.0 \pm 0.0}$	$\mathbf{500.0 \pm 0.0}$
(1.05, 0.105, 0.55)	$\mathbf{500.0 \pm 0.0}$	$\mathbf{500.0 \pm 0.0}$	$\mathbf{500.0 \pm 0.0}$
(1.3, 0.13, 0.675)	365.05 ± 50.53	$\mathbf{500.0 \pm 0.0}$	$\mathbf{500.0 \pm 0.0}$
(1.5, 0.15, 0.75)	316.85 ± 27.58	$\mathbf{500.0 \pm 0.0}$	$\mathbf{500.0 \pm 0.0}$

Table 2: Parameters and results for Gen-CartPole-v1. For the training sets, we used three parameter combinations $(m_c, m_p, l) \in \{(0.9, 0.09, 4), (1.0, 0.1, 0.5), (1.1, 0.11, 0.6)\}$.

4.2 Acrobot

We investigate Acrobot, a task where the agent controls a two-link pendulum to swing up and stabilize it in an upright position. Similar to the CartPole experiment, we evaluate three methods: A2C, VQ-A2C, and VQ-A2C-Reg.

Interpretability Due to spatial constraints, we have included the state cluster distribution figure of Acrobot in the appendix, which is close to the observation in Figure 5.

Robustness Consistent with the robustness testing approach used for CartPole, we assess the impact of noise on the three methods. Results of the evaluation are presented in Table 3, where the first row labeled "None" indicates the condition without added noise. It is evident that VQ-RL improves the robustness of the original A2C method against noise.

δ	A2C	VQ-A2C	VQ-A2C-Reg
None	-84.05 ± 23.81	-79.10 ± 8.40	$\mathbf{-78.65 \pm 7.80}$
1.00	-85.75 ± 11.93	$\mathbf{-85.05 \pm 14.64}$	-90.30 ± 15.84
0.90	-96.35 ± 21.22	-103.75 ± 32.00	$\mathbf{-79.80 \pm 12.52}$
0.80	-93.30 ± 18.26	-108.70 ± 43.88	$\mathbf{-87.85 \pm 15.89}$
0.70	-97.95 ± 20.97	-89.60 ± 12.34	$\mathbf{-88.50 \pm 17.44}$
0.60	-107.95 ± 21.99	$\mathbf{-105.20 \pm 25.47}$	-105.90 ± 29.95

Table 3: Results for noisy Acrobot-v1.

Generalization Following the generalization testing method used for CartPole, we introduce a modified version of Acrobot named Gen-Acrobot. In this environment, we vary three parameters: the mass of link 1 (m_1), the mass of link 2 (m_2), and the moments of inertia for both links (moi). Our results in Table 4 demonstrate that VQ-A2C-Reg outperforms the other two approaches.

(lm_1, lm_2, moi)	A2C	VQ-A2C	VQ-A2C-Reg
(1.2, 1.2, 1.2)	-114.25 ± 32.40	-136.35 ± 68.41	$\mathbf{-106.85 \pm 14.00}$
(1.1, 1.1, 1.1)	-104.75 ± 40.40	-114.05 ± 14.50	$\mathbf{-90.50 \pm 15.40}$
(0.9, 0.9, 0.9)	-77.15 ± 23.55	-89.70 ± 30.77	$\mathbf{-74.30 \pm 19.50}$
(0.75, 0.75, 0.75)	-65.55 ± 13.97	-88.65 ± 41.05	$\mathbf{-64.65 \pm 24.19}$
(0.4, 0.4, 0.4)	-49.65 ± 10.77	-47.10 ± 9.94	$\mathbf{-43.65 \pm 8.80}$

Table 4: Parameters and results for Gen-Acrobot-v1. For the training sets, we use $lm_1 \in \{0.95, 0.85, 0.65, 0.55\}$ and set $lm_1 = lm_2 = moi$.

4.3 LunarLander

We explore LunarLander in which the agent navigates a spacecraft to safely land on the lunar surface. Similar to the previous experiments, we assess three methods: PPO, VQ-PPO, and VQ-PPO-Reg.

Interpretability To accommodate space limitations, the state cluster distribution figure of LunarLander has been included in the appendix, aligning with the observations made in Figure 5.

Robustness Following the testing methodology for CartPole and Acrobot, we examine how noise affects the three methods. The evaluation results are displayed in Table 5, with the first row denoted as "None" representing the condition without added noise. Clearly, VQ-RL enhances the robustness of the original PPO method when exposed to noise.

δ	PPO	VQ-PPO	VQ-PPO-Reg
None	249.69 ± 16.79	241.87 ± 15.03	$\mathbf{261.30 \pm 23.65}$
19	187.71 ± 66.69	230.03 ± 69.64	$\mathbf{239.34 \pm 45.04}$
17	168.98 ± 62.36	218.32 ± 64.60	$\mathbf{239.02 \pm 50.55}$
15	152.0 ± 40.87	216.34 ± 65.14	$\mathbf{243.85 \pm 24.59}$
13	140.69 ± 36.50	215.98 ± 82.47	$\mathbf{228.81 \pm 59.71}$
11	127.49 ± 55.29	$\mathbf{215.68 \pm 68.31}$	184.53 ± 89.07
9	133.63 ± 36.05	93.55 ± 68.28	$\mathbf{142.63 \pm 62.17}$

Table 5: Results for the noisy LunarLander-v2 test.

Generalization To follow the generalization testing methodology employed for CartPole and Acrobot, we introduce a modified version of LunarLander called Gen-LunarLander. In this setup, we manipulate two parameters: the main engine power (*mep*) and the side engine power (*sep*). Our findings in Table 6 reveal that when both engines' power exceeds the training range, all three algorithms can achieve scores surpassing the game's threshold (200). However, when both engines' power falls below the training range, making it more challenging for the agent to control the spacecraft's movement to accomplish the landing objective, VQ-PPO-Reg demonstrates superior generalization compared to the other two methods.

(mep, sep)	PPO	VQ-PPO	VQ-PPO-Reg
(16, 0.9)	$\mathbf{257.93 \pm 32.10}$	252.54 ± 18.87	237.86 ± 22.24
(15, 0.8)	$\mathbf{257.89 \pm 27.77}$	248.53 ± 22.38	240.26 ± 15.82
(11, 0.4)	180.33 ± 87.90	$\mathbf{228.92 \pm 16.73}$	210.62 ± 53.24
(10.5, 0.35)	150.04 ± 113.93	151.07 ± 112.56	$\mathbf{171.51 \pm 101.08}$
(10, 0.3)	162.36 ± 120.29	162.44 ± 104.30	$\mathbf{184.71 \pm 83.66}$
(9.5, 0.25)	96.43 ± 132.48	62.00 ± 145.80	$\mathbf{162.63 \pm 105.35}$

Table 6: Results for the Gen-LunarLander-v2 test sets along with their parameter sets. For the training sets, we use the following set of values for (mep, sep): $\{(12, 0.5), (13, 0.6), (14, 0.7)\}$.

5 Conclusion and Future Endeavors

In this paper, we introduce a novel framework aimed at enhancing the clustering properties of the embedding space in deep RL methods. In our experimental analysis, we investigate the efficacy of VQ-RL concerning interpretability, robustness, and generalization. Our findings across three test domains suggest the following: 1) VQ-RL enhances the clustering properties of deep RL features, facilitating a deeper understanding of the internal dynamics of deep RL models; 2) VQ-RL demonstrates varying degrees of improvement in model robustness and generalization across different testing environments; 3) the incorporation of regularization methods proves effective and indispensable for preserving the robustness and generalization capabilities of VQ-RL.

Our future work involves expanding and refining our current efforts. This includes exploring additional environments and deep RL algorithms to validate the applicability of VQ-RL, extending VQ-RL to environments with continuous action outputs, and investigating its performance under various types of noise and generalization conditions.

References

Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pp. 449–458. PMLR, 2017.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Raymond H. Chan, Chung-Wa Ho, and Mila Nikolova. Salt-and-pepper noise removal by median-type noise detectors and detail-preserving regularization. *IEEE Trans. Image Process.*, 14(10): 1479–1485, 2005. doi: 10.1109/TIP.2005.852196. URL https://doi.org/10.1109/TIP.2005.852196.

Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, pp. 1282–1289. PMLR, 2019.

Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 2048–2056. PMLR, 2020. URL http://proceedings.mlr.press/v119/cobbe20a.html.

Vincent François-Lavet, Yoshua Bengio, Doina Precup, and Joelle Pineau. Combined reinforcement learning via abstract representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 3582–3589, 2019.

Cristina Gârbacea, Aäron van den Oord, Yazhe Li, Felicia SC Lim, Alejandro Luebs, Oriol Vinyals, and Thomas C Walters. Low bit-rate speech coding with vq-vae and a wavenet decoder. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 735–739. IEEE, 2019.

Maximilian Igl, Kamil Ciosek, Yingzhen Li, Sebastian Tschiatschek, Cheng Zhang, Sam Devlin, and Katja Hofmann. Generalization in reinforcement learning with selective noise injection and information bottleneck. *arXiv preprint arXiv:1910.12911*, 2019.

Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

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

Alex X Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *arXiv preprint arXiv:1907.00953*, 2019a.

Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. Network randomization: A simple technique for generalization in deep reinforcement learning. *arXiv preprint arXiv:1910.05396*, 2019b.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Björn Lütjens, Michael Everett, and Jonathan P How. Certified adversarial robustness for deep reinforcement learning. In *Conference on Robot Learning*, pp. 1328–1337. PMLR, 2020.

Daoming Lyu, Fangkai Yang, Bo Liu, and Steven Gustafson. Sdrl: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 2970–2977, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937. PMLR, 2016.

Alex Mott, Daniel Zoran, Mike Chrzanowski, Daan Wierstra, and Danilo J Rezende. Towards interpretable reinforcement learning using attention augmented agents. *arXiv preprint arXiv:1906.02500*, 2019.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. *arXiv preprint arXiv:1711.00937*, 2017.

Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. Assessing generalization in deep reinforcement learning. *arXiv preprint arXiv:1810.12282*, 2018.

Antonin Raffin, Ashley Hill, Maximilian Enerstus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable Baselines3, 5 2020. URL https://github.com/DLR-RM/stable-baselines3.

Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. In *Advances in neural information processing systems*, pp. 14866–14876, 2019.

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

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*, 529(7587):484–489, 2016.

Felipe Petroski Such, Vashisht Madhavan, Rosanne Liu, Rui Wang, Pablo Samuel Castro, Yulun Li, Jiale Zhi, Ludwig Schubert, Marc G Bellemare, Jeff Clune, et al. An atari model zoo for analyzing, visualizing, and comparing deep reinforcement learning agents. *arXiv preprint arXiv:1812.07069*, 2018.

Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, 8, 1995.

Chen Tessler, Yonathan Efroni, and Shie Mannor. Action robust reinforcement learning and applications in continuous control. In *International Conference on Machine Learning*, pp. 6215–6224. PMLR, 2019.

Andros Tjandra, Sakriani Sakti, and Satoshi Nakamura. Transformer vq-vae for unsupervised unit discovery and speech synthesis: Zerospeech 2020 challenge. *arXiv preprint arXiv:2005.11676*, 2020.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*, pp. 5045–5054. PMLR, 2018.

Jingkang Wang, Yang Liu, and Bo Li. Reinforcement learning with perturbed rewards. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 6202–6209, 2020.

Junpeng Wang, Liang Gou, Han-Wei Shen, and Hao Yang. Dqnviz: A visual analytics approach to understand deep q-networks. *IEEE transactions on visualization and computer graphics*, 25(1): 288–298, 2018.

Tom Zahavy, Nir Ben-Zrihem, and Shie Mannor. Graying the black box: Understanding dqns. In *International Conference on Machine Learning*, pp. 1899–1908. PMLR, 2016.

Amy Zhang, Nicolas Ballas, and Joelle Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *arXiv preprint arXiv:1806.07937*, 2018.

A Appendix

A.1 Training Strategy

In the CartPole and Gen-CartPole environments, we conducted experimental analysis based on the tuned models that all achieve a pass mark of 500 points. For A2C in Acrobot and PPO in LunarLander, we compared the performance of the models using hyperparameters tuned by StableBaseline3 (Raffin et al., 2020). However, for VQ-A2C, VQ-A2C-Reg in Acrobot, and VQ-PPO, VQ-PPO-Reg in LunarLander, as these models incorporate VQ encoding architecture, we fine-tuned the learning rates based on StableBaseline3's tuned hyperparameters, while keeping other hyperparameters unchanged. Similarly, in Gen-Acrobot and Gen-LunarLander, we fine-tuned the learning rates of all comparative models based on StableBaseline3's tuned hyperparameters to achieve optimal performance for experimentation.

Furtermore, we train all the models on one NVIDIA Tesla V100S 32 GB GPU and the operating system is CentOS Linux release 7.9.2009.

A.2 Architecture Details

Here we list the details of the hyperparameters and architecture used for our experiments.

Hyperparameter	Value
total timesteps	50,000
buffer size	10,000
learning starts	5,000
batch size	64
feature dimension	64
target update interval	50
number of VQ embeddings	10
C_d	5
train frequency	4
regularization weight	0.0005
learning rate	0.005
gamma	0.98
commitment cost	0.01
exploration final eps	0.01
exploration fraction	0.3
classification loss weight	0.5
VQ loss weight	0.1

Table 7: The hyperparameters in the VQ-DQN, VQ-DQN-Reg, and DQN trainings for CartPole-v1.

Hyperparameter	Value
total timesteps	50,000
buffer size	10,000
learning starts	500
batch size	128
feature dimension	64
target update interval	500
number of VQ embeddings	8
C_d	5
train frequency	16
regularization weight	0.0025
learning rate	0.0005
gamma	0.90
commitment cost	0.025
exploration final eps	0.1
exploration fraction	0.1
classification loss weight	0.0025
VQ loss weight	0.1

Table 8: The hyperparameters in the VQ-DQN, VQ-DQN-Reg, and DQN trainings for Gen-CartPole.

Hyperparameter	Value
total timesteps	500,000
batch size	80
feature dimension	64
number of environments	16
number of embeddings	8
C_d	5
VQ Learning Rate	0.0003238892
regularization weight	0.00025
learning rate	0.0007
gamma	0.99
gae gamma	1.0
VQ loss weight	0.25
entropy loss weight	0.0
value loss weight	0.5
classification loss weight	0.1
commitment cost	0.1
horizon	5
N Steps	5

Table 9: The hyperparameters in the VQ-A2C, VQ-A2C-Reg, and A2C trainings for Acrobot and Gen-Acrobot. Here, learning rate is used by A2C in Acrobot, while VQ learning rate is used to train other models.

Hyperparameter	Value
total timesteps	1,000,000
batch size	64
feature dimension	64
number of environments	16
number of embeddings	16
C_d	5
VQ Learning Rate	0.0003471539
regularization weight	0.001
learning rate	0.0003
gamma	0.999
gae gamma	0.98
VQ loss weight	1
entropy loss weight	0.01
value loss weight	0.5
classification loss weight	0.05
commitment cost	0.5
horizon	1024
number of epochs	4

Table 10: The hyperparameters in the VQ-PPO, VQ-PPO-Reg, and PPO trainings for LunarLander. Here, learning rate is used by PPO in LunarLander, while VQ learning rate is used to train other models.

Hyperparameter	Value
total timesteps for Minigrid	50,000
total timesteps for Gen-Minigrid	100,000
feature dimension for Minigrid	16
feature dimension for Gen-Minigrid	32
number of embeddings for Minigrid	8
number of embeddings for Gen-Minigrid	16
horizon	1,024
batch size	128
number of epochs	10
C_d	3
number of environments	1
learning rate	0.0025
regularization weight	0.001
gae gamma	0.95
gamma	0.9
commitment cost	0.7
value loss weight	0.5
entropy loss weight	0.3
VQ loss weight	0.1
classification loss weight	0.1

Table 11: The hyperparameters in the VQ-PPO, VQ-PPO-Reg, and PPO trainings for MiniGrid-Empty-Random-6x6-v0 and Gen-MiniGrid-Empty-v0.

Hyperparameter	Value
number of embeddings	32
C_d	5
VQ loss weight	0.01
classification loss weight	0.25
commitment cost	0.3
regularization weight	0.1

Table 12: The VQ encoding hyperparameters in the VQ-PPO and VQ-PPO-Reg trainings for CoinRun.

Layer Type	Dimensions	Outputs
Fully connected + ReLU	64	64
Fully connected + ReLU	64	64
Fully connected	64	N

Table 13: The architecture in the training of all models in the main paper. 'N' represents the number of possible actions that varies for different games.

MiniGrid					
Layer Type	Dimensions	kernel size	stride	padding	Outputs
Convolutional	3	4	2	1	8
Convolutional	8	2	2	1	16
flatten					
Fully connected	64				16
Fully connected	16				3
Gen-Minigrid					
Layer Type	Dimensions	kernel size	stride	padding	Outputs
Convolutional	3	4	2	1	8
Convolutional	8	2	2	1	16
flatten					
Fully connected	64				32
Fully connected	32				3

Table 14: The architecture in the VQ-PPO and PPO trainings for MiniGrid-Empty-Random-6x6-v0 and Gen-MiniGrid-Empty-Random-6x6-v0.

A.3 CoinRun

In this experiment, we leverage the Procgen version of CoinRun (Cobbe et al., 2020) for testing. In this game, the agent needs to move to the far-right of the screen and touch the gold coin to get rewards. We used 200 generated levels of CoinRun on the easy difficulty setting to train our model and the same network and hyperparameters as the one in the original paper (Cobbe et al., 2020), except for adding VQ encoding. Furthermore, each model is trained three times with different random seeds, and the evaluation of each model is based on the average of the three instances.

Interpretability The same cluster after training may contain multiple different highest probability actions, although the classification loss function constrains features in the same cluster to have the same highest probability actions as much as possible. This may be due to the choice of hyperparameters (such as relatively small weight for the loss of the classification task, etc.) and the action distribution of states with high entropy (some states may have multiple optimum actions to achieve

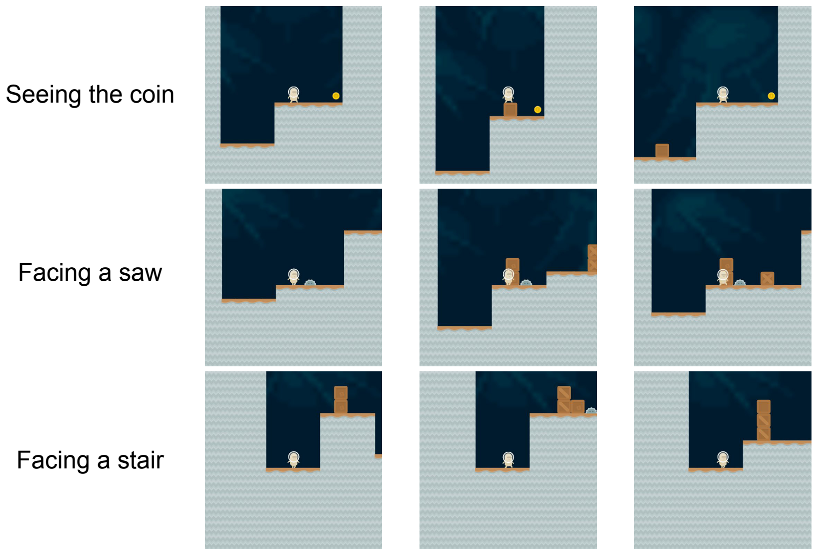


Figure 6: Some sub-classified state examples of VQ-PPO-Reg on CoinRun.

the goal.). However, we can see meaningful clustering results if the states are sub-classified according to different highest probability actions within each cluster. For example, as shown in Figure 6, the states in each row originate from the same cluster and have the same highest probability action. Therefore, the clustering in VQ-RL can still explore the internal state of the models and help us further understand the decision-making process in deep RL.

Robustness Salt and Pepper (SP) is one of the most common image noises Chan et al. (2005). Therefore, we inject SP noise with different degrees to the RGB states in CoinRun to test the robustness of the three models. The results are shown in Table 15. We see that the VQ-RL framework improves the robustness of PPO.

p	PPO	VQ-PPO	VQ-PPO-Reg
1	9.43±2.31	9.50±2.17(+0.7%)	**9.63±1.87(+2.1%)**
10	8.60±3.46	**8.93±3.08(+3.8%)**	8.90±3.12(+3.1%)
20	7.36±4.40	7.66±4.22(+4.1%)	**8.63±3.43(+17%)**
30	6.53±4.75	**7.53±4.31(+15%)**	7.33±4.42(+12%)
40	6.40±4.80	6.56±4.74(+2.5%)	**6.70±4.70(+4.7%)**
50	6.03±4.89	5.80±4.93(-3.9%)	**6.33±4.81(+5.0%)**

Table 15: The results for the noisy CoinRun test. The proportion of noisy points is given by $p \times 0.004$, where p is a configurable parameter. We also include percent improvements from the baseline.

Generalization In the final experiments, we test the generalization of the models for the CoinRun domain. We test the three models trained with the same random parameters on the same 1,000 unseen 'all distributions' episodes to ensure fairness. We performed this experiment three times, labeled as Run1-3. The results for this generalization experiment can be seen in Table 16. We see that the VQ-RL framework slightly improves the generalization of PPO.

Run	PPO	VQ-PPO	VQ-PPO-Reg
1	8.77±3.28	8.58±3.49(-3.2%)	**8.84±3.20(+0.8%)**
2	8.59±3.48	8.53±3.54(-0.7%)	**8.62±3.44(+0.3%)**
3	8.77±3.28	8.66±3.40(-1.3%)	**9.00±3.00(+2.6%)**
Ave.	8.71±3.35	8.59±3.48(-1.4%)	**8.82±3.22(+1.2%)**

Table 16: The results for the generalization CoinRun test, including percent changes from the baseline PPO.

A.4 MiniGrid-Empty

The MiniGrid-Empty-Random-6x6-v0 domain is chosen for the model training in the interpretability and robustness experiments in this section. In this domain, an agent moves from a random initial position and orientation to a green target in the lower right corner, using a set of three actions: turning left, turning right, and moving forward. The state of the agent is encoded using a 7×7 field of view (FoV) with three layers of channels, where the first layer encodes object identifiers ranging from 0 to 10, the second layer encodes object colors as integers from 0 to 5, and the third layer encodes whether the block is empty or not (0 or 1). Each result in this section is calculated using 50 episodes.

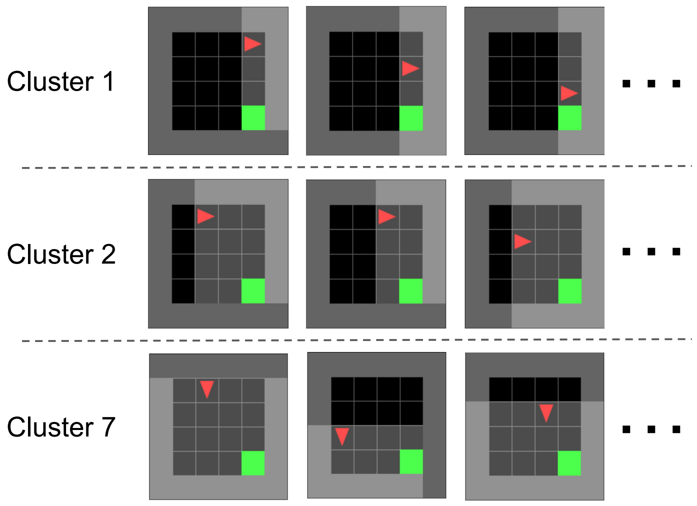


Figure 7: The clustering result of VQ-DQN-Reg on MiniGrid-Empty-Random-6x6-v0. The model divides the state into three clusters according to the actions that need to be performed. In cluster 1, the agent performs the turning right action. Cluster 2 represents moving forward. Cluster 7 is turning left.

Interpretability Figure 7 shows the clustering results of the state of the trained VQ-PPO-Reg model. The model categorizes the state space into three corresponding clusters according to the three actions performed in the policy.

Robustness The robustness of the model is tested by adding random noise to the state (the FoV component) in MiniGrid-Empty-Random-6x6-v0. Specifically, as performing each action, each grid cell in the FoV has a certain probability p_a of being 'attacked', which results in an "illusion", that is, each of the three encoding layers assigns a random value within the valid encoding range for this grid cell. The results are given in Table 17. When p_a is small, the performance of the three models is close. However, as p_a increases, the performance of VQ-PPO decreases rapidly, but VQ-PPO-Reg can still maintain a relative advantage over the other two models.

p_a	PPO	VQ-PPO	VQ-PPO-Reg
0.01	0.96±0.01	**0.97±0.01**	0.96±0.02
0.05	0.94±0.04	**0.95±0.04**	**0.95±0.04**
0.1	0.89±0.08	**0.93±0.06**	0.91±0.08
0.2	0.82±0.18	0.83±0.20	**0.83±0.18**
0.3	0.76±0.25.	0.66±0.34	**0.78±0.22**
0.4	**0.74±0.27**	0.44±0.42	0.72±0.28
0.5	0.60±0.38	0.38±0.42	**0.67±0.34**

Table 17: The results for the noisy Minigrid test.

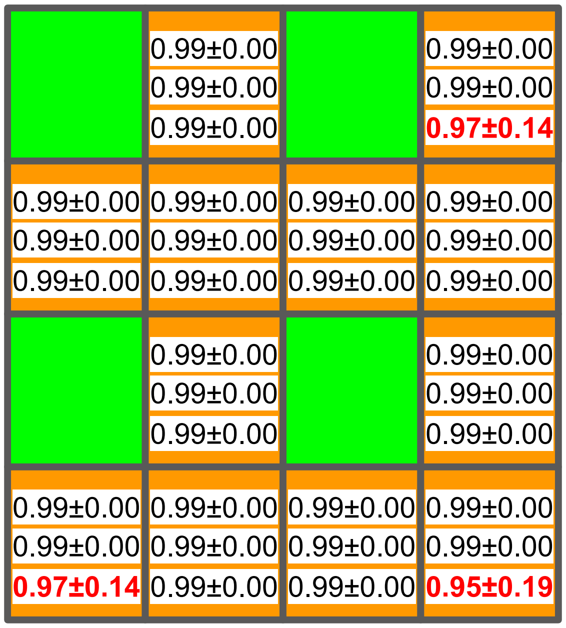


Figure 8: Grid showing the locations of goals in training and test and the generalization test results. In the label for each orange cell, the first, second, and third row correspond to the performance of the VQ-PPO, VQ-PPO-Reg, and PPO methods respectively.

Generalization To test the generalization of our models, we propose a modified MiniGrid Empty-Random-6x6-v0 domain that we call *Gen-MiniGrid-Empty-Random-v0*. During training, each episode randomly produces the goal from four possible locations (the green cells in Figure 8). In the evaluation, we test the performance of models in the rest of the locations (the orange cells in Figure 8). The corresponding results are also presented in Figure 8. We can see that the three methods have very similar performance, except for the three corner cells, for which the PPO algorithm performs decidedly worse than the other two.

A.5 More Figures

Here we list additional figures.

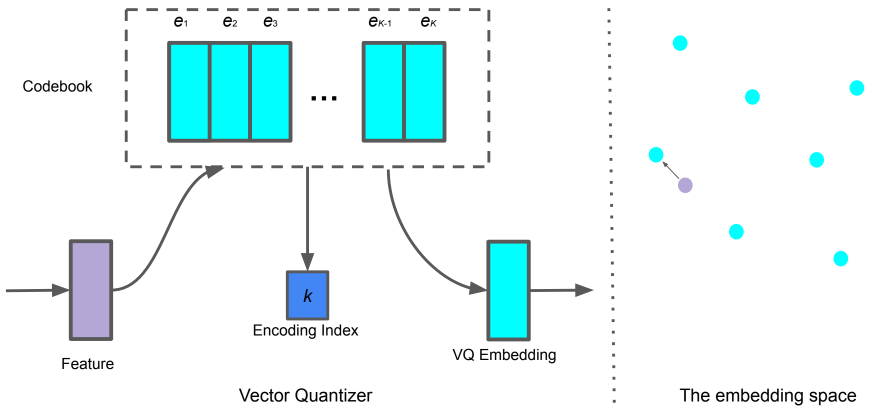


Figure 9: The design of vector quantizer, which returns the closest embedding vector and the corresponding index in the codebook. The right side shows an example of a feature (purple) being clustered to its nearest neighbor embedding vector (cyan).

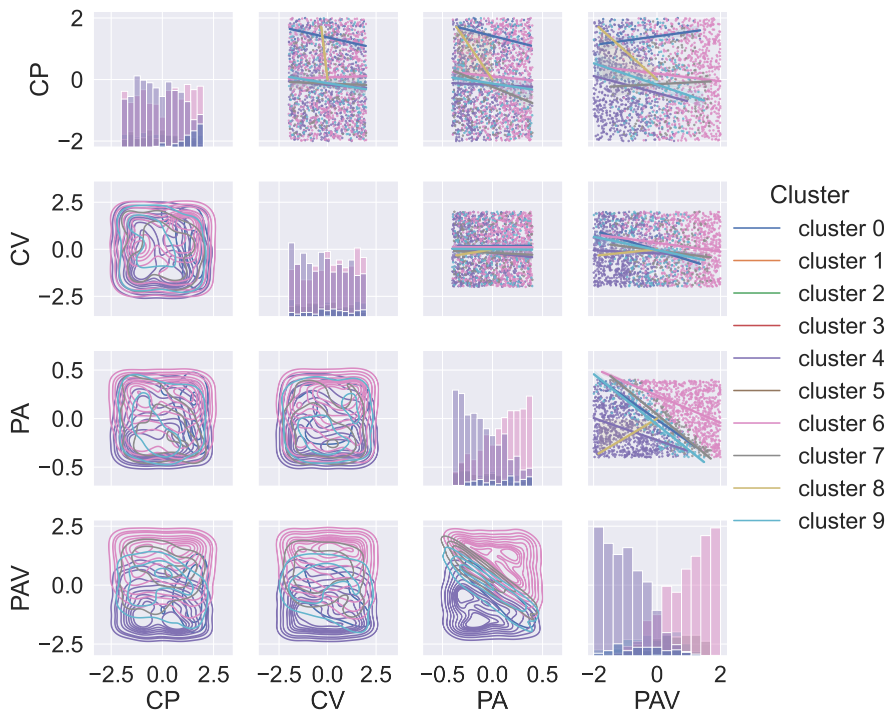


Figure 10: The state clusters for VQ-DQN in the CartPole domain. Plots above the diagonal show the distribution of clusters and the corresponding linear regression lines, plots along the diagonal are histograms, and plots below the diagonal are contour plots of the Gaussian kernel density estimate (KDE) plots of clusters. We see that PA and PAV are the primary state components that distinguish the clusters. In each state component, each cluster presents a continuous distribution in a specific interval.

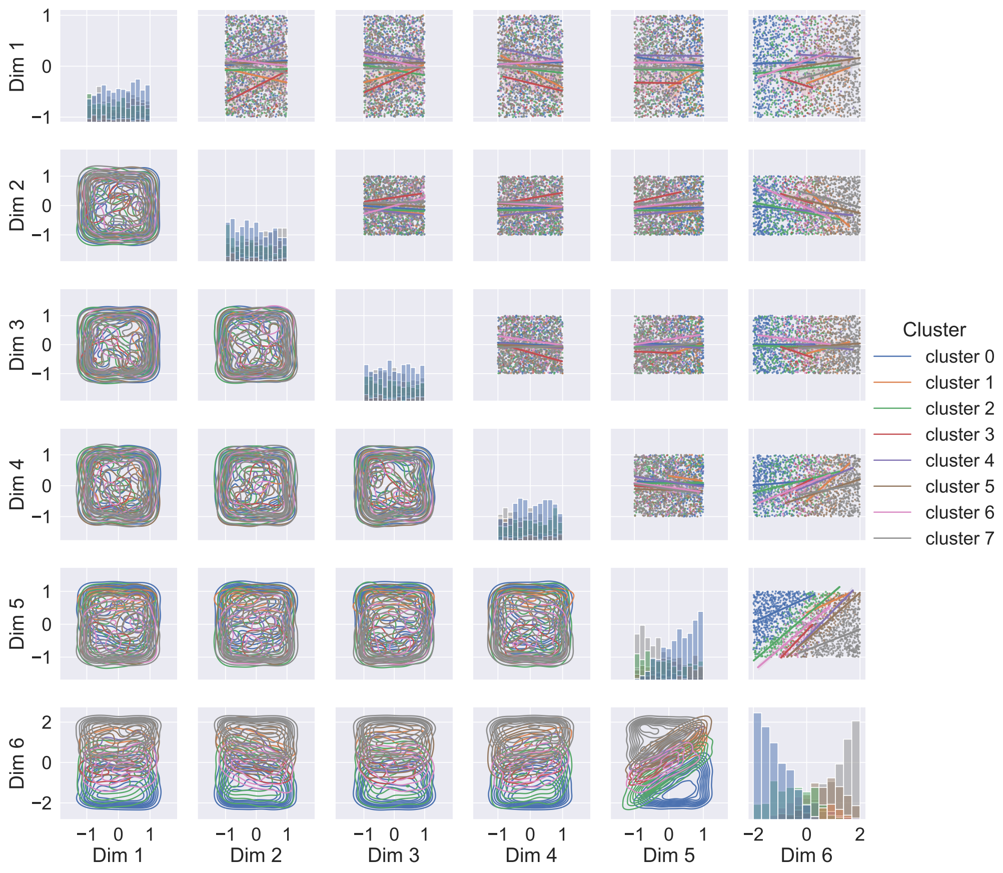


Figure 11: The state clusters for VQ-A2C-Reg in the Acrobot domain. Plots above the diagonal show the distribution of clusters and the corresponding linear regression lines, plots along the diagonal are histograms, and plots below the diagonal are contour plots of the Gaussian kernel KDE plots of clusters. We see that Dim 6 (angular velocity of the second link) is the primary state component that distinguishes the clusters. In each state component, each cluster presents a continuous distribution in a specific interval.

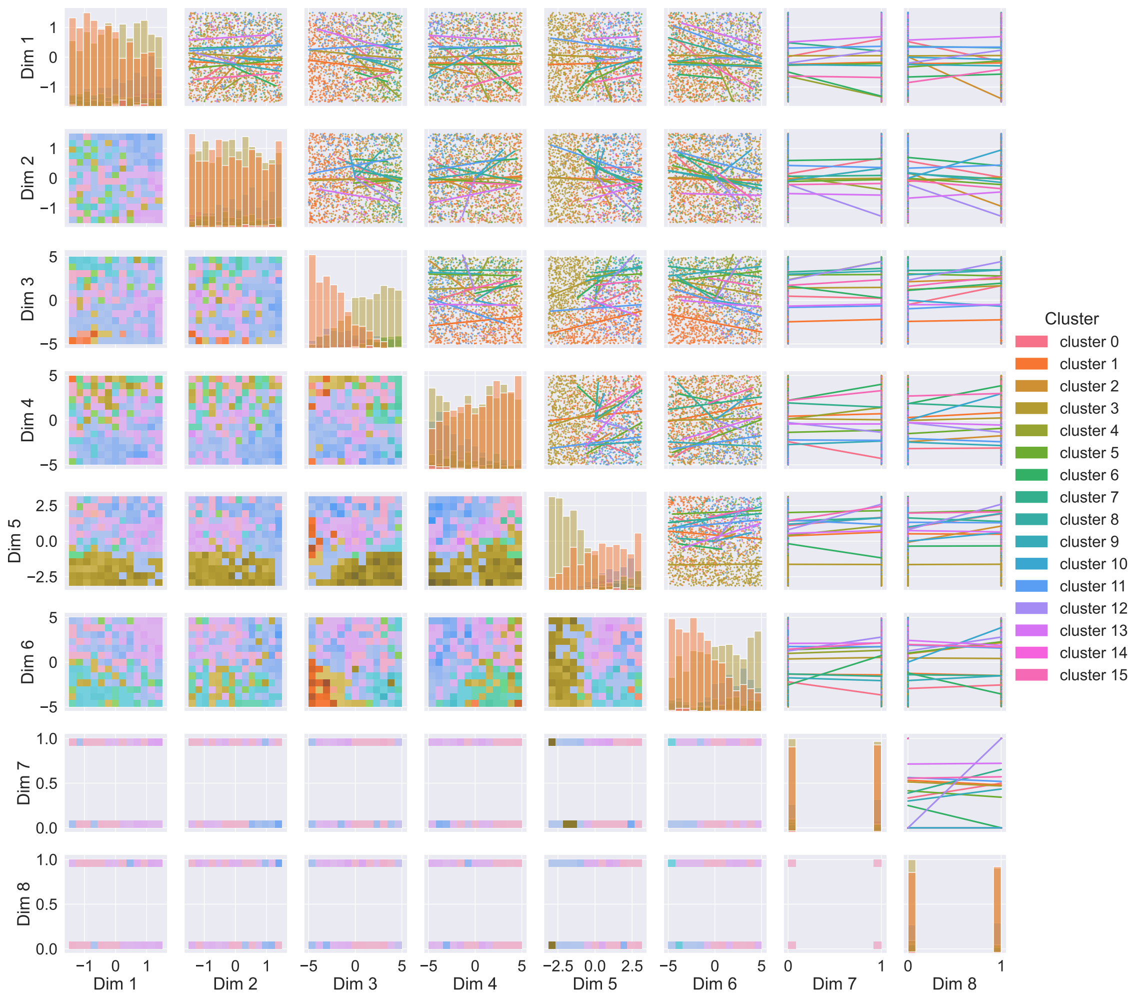


Figure 12: The state clusters for VQ-PPO-Reg in the LunarLander domain are visualized as follows: Plots above the diagonal depict the distribution of clusters along with corresponding linear regression lines, while plots along the diagonal represent histograms. Plots below the diagonal are also histograms. Due to the non-Gaussian distribution of state clusters, KDE was not used; instead, histograms were employed for visualization. Unlike the continuous component values in other game states, LunarLander's Dim 7 and Dim 8 represent two booleans indicating whether each leg is in contact with the ground or not. We observe that Dim 3 (the linear velocity of the lander in the x-direction) is the primary state component that distinguishes the clusters. Each cluster exhibits a continuous distribution within a specific interval across all state components.