Chair of Computer Science 6 (Machine Learning and Reasoning)

RWTH AACHEN UNIVERSITY

The present work was submitted to the Chair of Machine Learning and Reasoning.
Diese Arbeit wurde vorgelegt am Lehrstuhl für Maschinelles Lernen und Inferenz.

On Learning Representations in Model-Based Reinforcement Learning

Über das Lernen von Repräsentationen im modellbasierten Reinforcement Learning

Bachelor Thesis
Bachelorarbeit

Presented by / Vorgelegt von

Yingliu Lu
435654

Supervised by / Betreut von Jonas Reiher, M.Sc.

1st Examiner / 1. Prüfer Prof. Hector Geffner, Ph.D.

2nd Examiner / 2. Prüfer Prof. Dr. rer. nat. Christopher Morris

Aachen, January 13, 2025

Abstract

Learning symbolic planning representations from perceptual observations bridges deep Reinforcement Learning (RL) and Classical Planning, and has the potential of addressing the limitations of each paradigm. This thesis explores the feasibility of learning symbolic representations from latent representations learned by a neural network. Specifically, we investigate the possibility of extracting object-centric or symbolic information from the discrete codebook space of the Vector-Quantized Variational Autoencoder (VQ-VAE) of the model-based RL framework Imagination with auto-Regression over an Inner Speech (IRIS).

Two methods are proposed to evaluate whether and in what sense object attributes in Atari games are captured in the discrete latent representation produced by the VQ-VAE. The first combines clustering of the VQ-VAE codebook vectors with an interactive visualization tool, while the second employs random forest regressors and analyzes permutation importance of positions in the discrete representation as features. Findings from the two methods diverge, highlighting limitations of the first method. Results from the second method aligns with our hypothesis, but require further validation.

The visualization tools provide insights into the workings of the discrete representations. We observe that a single codebook vector tends to encode more than one localized changes in the image observation, and its effect can be influence by other codebook vectors. Furthermore, a change in the value of an object attribute is usually represented by the combined effect of several codebook vectors.

Contents

1 Introduction

Deep Reinforcement Learning (RL) is capable of making decisions based on raw sensory inputs alone, but suffers from challenges such as low sample efficiency and limited generalizability, not to mention that the process by which the deep neural networks select the next action remains, for the most part, reactive and opaque to humans. [17, 46]. On the other hand, classical planning methods offer systematic generalization and interpretability but require models of the domain as input [17, 18]. These models are assumed to be known a priori [19, 35], and are usually handcrafted by human, rather than being grounded in perceptual observations from the domain itself [24].

One way to address these limitations is to bridge the two approaches by, first, learning representations from raw perceptual data, and then performing symbolic reasoning based on the learned representations [17]. These representations are object-centric, symbolic and/or relational in nature and ideally can be compiled into languages such as STRIPS [14] and Planning Domain Definition Language (PDDL) [37]. The problem of learning such representations is therefore two-fold:

1. How to extract semantic meaning such as objects and relations between objects from raw perceptual data?
2. How to derive causal and relational knowledge in the form of predicates and action schemas from the extracted representations?

Existing literature on learning symbolic representations from raw perceptual observations either assumes that objects, object attributes, and certain spatial relations are already extracted by some object-recognition system [3, 33], or restricts the prototype system to domains with such simplicity that objects and dynamics can be extracted merely based on neural network activations and the injection of common sense priors [16, 17].

This thesis explores the feasibility of extracting semantic information from the discrete latent representations learned from image observations by a Vector-Quantized Variational Autoencoder (VQ-VAE), which is widely used in representation learning for downstream tasks [42, 44]. The conclusions drawn from this work can inform the design of future frameworks that aim to learn first-order symbolic representations from raw perceptual data without prior knowledge for the task of action selection and planning.

We focus on Imagination with auto-Regression over an Inner Speech (IRIS) [38], a model-based RL framework where a world model, consisting of a VQ-VAE [40, 42] and a Transformer [48], is

trained to simulate the environment: The VQ-VAE encoder maps the high-dimensional input image to a lower-dimensional latent vector of discrete tokens, where each token corresponds to a vector in the learned codebook. The Transformer then models the environment dynamic in the latent space, based on the latent vector and the action selected. The VQ-VAE decoder uses the Transformer outputs to reconstruct the next image observation. The agent learns to act by interacting exclusively with the world model via the reconstructed images and simulated rewards [38].

Originally designed to improve sample efficiency in deep RL, IRIS appears particularly relevant for the task of learning representations from raw perceptual data, especially because of the discrete structure of the latent representations learned by the VQ-VAE, which makes them potentially interpretable, object-centric and/or symbolic in nature. We intend to investigate whether the discrete latent representations learned by the VQ-VAE in IRIS can be used to derive semantically meaningful representations of the states. Specifically, we hypothesize that these latents are useful because they capture objects and their attributes at a symbolic or object-centric level. A formal statement of this hypothesis is provided in Section 3.3. We propose two separate methods to examining the hypothesis: One based on the clustering of the VQ-VAE codebook vectors and visualization of perturbed discrete representations reconstructed by the VQ-VAE decoder, and the other via feature importance of random forest regressors, each of which predicts, based on the discrete representation, the value of a selected object attribute in the reconstructed images. By comparing the results from both approaches, we hope to shed light on the validity of our hypothesis.

The remainder of this thesis is organized as follows: In Chapter 2, we present the related work on deep RL frameworks that rely on world models to simulate the environment, learning object-centric representations, and learning symbolic representations for RL and planning. We then provide the necessary background on planning, model-based RL, and the key components of IRIS, with a focus on its VQ-VAE, in Chapter 3. In Chapter 4, we describe the methods used to test our hypothesis, including the clustering of the VQ-VAE codebook, the visualization of perturbed image frames, and the feature importance analysis of the random forest regressor. Subsequently, we present the results of our experiments in Chapter 5, and discuss the implications of these results in Chapter 6.

2 Related Work

Our aim in this work is to investigate whether the discrete representations learned by the VQ-VAE within the IRIS framework exhibit structured patterns with symbolic significance. In this section, we review related literature in three research areas: Reinforcement Learning applied in the simulation of world models for Atari gameplay, object-centric representation learning from images, and symbolic representation learning for planning and action tasks. Notably, variants of Variational Autoencoder (VAE) frequently appear across these methodologies and research areas.

2.1 Reinforcement Learning Using World Models

Drawing inspiration from how humans form mental models to guide decision-making [15], Ha and Schmidhuber [20] proposed a model-based RL approach, which involves reinforcement learning within a world model env' that simulates the actual environment env, This method follows an iterative process described in Algorithm 1.

Algorithm 1 Learning in the Simulation of World Models, adapted from Kaiser *et al.* [29]

Initialize policy π
Initialize parameters θ of world model env'
Initialize empty set B of experience
while not done **do**
    $B \leftarrow B \cup$ **collect_experience**(env, π) — *Gather real-world experience*
    $\theta \leftarrow$ **update_world_model**(env', B) — *Improve world model predictions*
    $\pi \leftarrow$ **update_behavior**(π, env') — *Update policy function*
Return: Optimized policy π^*

Kaiser *et al.* [29] applied this approach to the Arcade Learning Environment (ALE) benchmark [6], demonstrating that it could achieve performance competitive with contemporary model-free RL methods.

Hafner *et al.* [21] introduced Dreamer, which trains a policy within the latent space of a learned world model using a Recurrent State-Space Model (RSSM). DreamerV2 [22] improved upon Dreamer with discrete latents, achieving human-level performance on 55 tasks of the ALE benchmark. DreamerV3 [23] further extended the framework to diverse domains, including robotics and continuous control tasks.

Alonso *et al.* [2] adopted a diffusion-based world model to better capture visual details, achieving state-of-the-art returns on the ALE benchmark among agents trained exclusively within the simulation of world models.

Among the aforementioned frameworks, only DreamerV2 and DreamerV3 map observations to discrete latents, similar to IRIS. However, their latent representations at each time step are derived from the corresponding hidden state, which incorporates information from past observations and actions. In contrast, IRIS abstracts discrete representation directly from the image observation at a single time step. This suggests that the information encoded in the discrete latents of IRIS may be more interpretable.

Table 2.1 provides a comparison of the architectures of the aforementioned frameworks as well as IRIS, which we will introduce in detail in Chapter 3. Notably, all but Alonso *et al.* [2] employ convolutional encoders and decoders to map observations to and from the latent space.

Policy trained in	Model	Description
Latent Space	Ha and Schmidhuber [20]	A Convolutional Neural Network (CNN) encodes observation into latent vector, a Recurrent Neural Network (RNN) integrates previous hidden state with current latent to predict the next latent and update current hidden state.
	Dreamer [21–23]	A CNN encodes observation and action into latent state, a RSSM predicts reward and next latent state given current latent and action.
Observation Space	Kaiser *et al.* [29]	A convolutional encoder embeds four previous frames, a Long short-term memory (LSTM) approximates discrete latent of the next state, a convolutional decoder predicts the next frame conditioned on the embedding and discrete latent.
	Alonso *et al.* [2]	A score-based diffusion model autoregressively predicts the next observation, conditioned on the episode's past observations and actions.
	Micheli *et al.* [38]	A VQ-VAE encodes observations to discrete latents and reconstructs observations from these latents, a Transformer autoregressively predicts next discrete latent and reward on past discrete latents and actions.

Table 2.1: Comparison of World Models

Such RL paradigm using world models reduces the amount of data required to train agents and has the potential to address safety concerns associated with training in real-world settings [2]; however, its generalizability remains an open question and requires further investigation [23]. Additionally, the action selection process in deep RL is inherently reactive and remains largely uninterpretable to humans [17].

2.2 Learning Object-Centric Representations

Slot attention [34], when combined with an autoencoder, has been used for unsupervised object discovery in images, where each object is represented as a fixed-size vector known as a "slot" [8, 45]. This method segments scenes into distinct, interpretable components without requiring labeled supervision, which is particularly useful in scenarios where objects are not known a priori.

Building on this foundation, several improvements have been proposed to bind objects to specialized slots: Kori *et al.* [31] introduced vector quantization to the slot attention mechanism, alongside learning a prior distribution over the slots. Didolkar *et al.* [12] augmented additional loss terms that measure the so-called "cycle consistency" of slots and visual features extracted by the encoder. They also demonstrated the applicability of their approach in a number of downstream RL tasks, such as Atari gameplay and robotic manipulation in Causal World [1].

Alternatively, scene graphs [28] extract a structured, object-centric, and relational representation of raw sensory inputs. Typically, an image is parsed into a graph where nodes correspond to objects, edges represent relationships between objects, and each object is further characterized by its class and attributes. Such representation facilitates reasoning about scenes.

Various approaches have been proposed for generating scene graphs from images. Xu *et al.* [49] introduced an iterative message-passing model that refines object and relationship predictions by leveraging the contextual information of neighboring nodes and edges. Zellers *et al.* [52] enabled generation of more accurate scene graphs by capturing higher order common substructures, or motifs, within the graph. Yang *et al.* [50] proposed the Graph R-CNN, which integrates object detection with graph-based reasoning to generate scene graphs more efficiently and effectively.

For a comprehensive review of advancements in this field, see the works of Chang *et al.* [10] and Li *et al.* [32].

2.3 Learning Symbolic Representations to Plan and Act

In this section, we first outline the desired properties of the symbolic representations to be learned, before reviewing various methods for learning these representations for the purpose of RL and planning.

Properties of Symbolic Representations

Lorello and Lippi [35] identified several key properties that symbolic representations should ideally possess:

- *Non-ambiguity*: The same symbol should not correspond to different entities,
- *Purity*: Each entity should be mapped to a minimal set of symbols,
- *Symbol Composability*: Symbols can combine with each other to form complex constructs in an open-ended manner [17],
- *Manipulation Composability*: New symbols can be derived by applying operations to existing symbols,
- *Usefulness*: Learned representations should outperform alternatives in downstream tasks,
- *Generalizability*: Symbols should generalize across downstream tasks, environments, space, and time.

These properties guide efforts in learning symbolic representations and highlight several challenges. These include optimizing discretization operations that are not differentiable, selecting appropriate loss functions, deciding whether or how to incorporate prior knowledge, and developing metrics to quantify composability [35].

Learning Symbolic Representations for Reinforcement Learning

Garnelo *et al.* [17] proposed integrating symbolic reasoning into deep RL by transforming image observations into symbolic representations and subsequently applying Q-learning within the symbolic space.

The symbolic representation of the domain is extracted through the following steps:

1. Train a convolutional autoencoder to reconstruct image observations from the game environment;
2. Identify low-level object types, based on the activations in the autoencoder's convolutional layers and the geometric properties computed by the autoencoder;
3. Track objects across observations and assign labels based on common sense priors such as object persistence over time;
4. Learn the dynamics of object interaction by analyzing the local relative positions of objects between successive observations.

Subsequently, a Q-function is learned in a tabular fashion for each interaction between object types. Actions are selected according to the summed-up Q values for all relevant interactions. Although the resulting policy is only locally optimal due to the reliance on common sense priors, it generalizes significantly better than a Deep Q-Network (DQN) in environments with random object placements. This improvement occurs despite both approaches being trained in a setting with grid-like object placement [17].

Garcez *et al.* [16] introduced simplifying modifications to the framework in how the Q-value function is updated and how actions are selected. These changes led to improved performance in environments with random object placements.

Learning Symbolic Representations for Planning

Latent-space Planner (LatPlan) [4] generates a *propositional* representation grounded in image observations for planning tasks. It employs a VAE with Gumbel-Softmax [26] to learn a mapping between raw image observations and binary latent vectors. Each bit in the latent vectors corresponds to a distinct categorical variable, which is directly used as a *propositional symbol*. Once trained, LatPlan takes in two images—corresponding to the initial and goal state— as input and outputs a sequence of reconstructed images that form a plan to transition from the initial state to the goal state. The framework achieves high success rates in games such as 8-puzzle and LightsOut. [4] However, its applicability to more complex and dynamic environments remains an open question.

Improving upon LatPlan, Asai [3] introduced First-Order State Autoencoder (FOSAE), which grounds First-Order Logic (FOL) predicates from feature vectors of objects (assumed to be extracted from raw perceptions by some object-recognition system) without human supervision. The FOSAE encoder consists of multiple Predicate Units (PUs), each trained to identify relevant arguments among input objects and compute a boolean value for the corresponding predicate. This process is done with the help of attention mechanisms [5] combined with Gumbel-Softmax [26]. Similar to LatPlan, the resulting FOL representation is directly compiled into a PDDL model to be solved by a planner.

This method was evaluated on generated instances of 8-puzzles and photo-realistic Blocksworld, All provided instances were successfully solved, with each plan checked for correctness, demonstrating FOSAE's compatibility with planning. It is worth noting, however, that the learned FOL statements thus learned are not necessarily quantifier-free [3]. Additionally, the extracted predicates are anonymous and may not correspond directly to manually defined symbols. Nevertheless, Asai [3] argue that these predicates can be interpreted through visualization using the FOSAE decoder.

The frameworks discussed so far rely solely on deep neural networks to learn representations directly from raw observations. Alternatively, Liberman *et al.* [33] propose a different approach:

learning planning domains from images parsed by object-recognition systems, where an image is represented not as a feature vector, but in a first-order language called Objects in 2D space (O2D). The incremental learning of predicates and action schemas from the structure of the parsed image space can thus be cast as a combinatorial optimization problem. Experiments in domains including Blocks, Hanoi, and Sokoban demonstrate that the learned domains are fully compatible with standard off-the-shelf planners [33].

3 Background

This chapter introduces the concepts, terminologies, and notations used in the thesis. We start by introducing the concepts of classical planning and model-based RL, followed by a description of the key components of the IRIS framework: VQ-VAE with an emphasis on its discretization process, Transformer, and actor-critic.

3.1 Classical Planning

Classical planning is the task of finding, for a given problem description, a sequence of actions (a plan) that, when followed, leads from an initial state to a goal state [19, 43].

We follow Liberman *et al.* [33] in defining a classical planning problem in PDDL [37] as a pair $P = \langle D_{\text{FO}}, I \rangle$ [25, 33]. Here, D_{FO} is a first-order domain, consisting of a set of predicates and a set of action schemas. An atom is a predicate applied to a sequence of terms, whereas a literal is an atom or its negation. An action schema is a parameterized action defined by the action name, a list of all its parameters, its precondition, and its effect, where both the precondition and the effect are sets, i.e., conjunctions of literals [43]. The instance information $I = \langle O, \mathit{Init}, \mathit{Goal} \rangle$ is given by a finite set of objects O, sets of ground literals Init and Goal.

A state s over P is a maximally consistent set of ground literals that represents a truth evaluation over the atoms that can be formed from the predicates in D_{FO} and the objects in I. A ground action a is an instantiation of the corresponding action schema with its parameters replaced by objects in I, and it is applicable in state s if its precondition holds in s. A state s' is the successor $f(a, s)$ of state a given action a if the effect of a holds in s' and the value of atoms not affected by a remains the same in s and s'. And a plan for P is a sequence $a_0, \dots, a_n$ of ground actions such that there exists a state sequence $s_0, \dots, s_{n+1}$ with $s_0 = \mathit{Init}$, $s_{n+1} = \mathit{Goal}$, a_i is applicable in s_i, and $s_{i+1} = f(a_i, s_i)$ for all i [33].

A planner takes such a problem description as input and finds a plan by turning it into a search problem or a logical reasoning problem that can be solved efficiently with the help of heuristics and other techniques [25].

3.2 Model-Based Reinforcement Learning

An environment in Reinforcement Learning is typically modeled as a Markov decision process (MDP) [47]. A MDP can be considered as a generalization of a state model in classical planning [19] and is given by:

- a set of states $\mathcal{S}$,
- a set of actions $\mathcal{A}$,
- transition model: a probability distribution $\Pr(s_{t+1} \mid s_t, a_t)$ over the next states after action a_t is applied in state s_t,
- reward model: a probability distribution $\Pr(r_t \mid s_t, a_t, s_{t+1})$ over the rewards received after action a_t is applied in state s_t,
- the discount factor $\gamma \in [0, 1]$, and
- the time horizon $h \in \mathbb{N} \cup \{\infty\}$, by which fixed time steps $t = 0, 1, 2, \dots, h$ are bounded.

In a Partially Observable Markov Decision Process (POMDP), the agent does not have direct access to the states but only to observations $o \in \mathcal{O}$, and we additionally have a sensor model, i.e., probabilities $\Pr(o_t \mid a_{t-1}, s_t)$ of observing o_t in state s_t given the previous action a_{t-1}.

In RL, the transition and reward models are unknown, and the goal is to learn through interaction with the environment an optimal policy $\pi^* : \mathcal{S} \to \mathcal{A}$ that maximizes the expected utility, i.e., the discounted sum of rewards:

$$\pi^* = \arg\max_{\pi} \sum_{t=0}^{h} \gamma^t \mathbb{E}_{\pi}[r_t] \tag{3.1}$$

The transition and reward models are learned explicitly in model-based RL to be used for planning, policy evaluation, and policy improvement [47]. IRIS is a model-based RL framework, where a world model consisting of a VQ-VAE and a Transformer models the environment as a POMDP with [38]:

- image observations $x_t \in \mathbb{R}^{H \times W \times 3}$,
- discrete actions $a_t \in \{1, \dots, |\mathcal{A}|\}$,
- scalar rewards $r_t \in \mathbb{R}$,
- episode termination signals $d_t \in \{0, 1\}$,
- discount factor $\gamma \in (0, 1)$,
- initial observation distribution ρ_0, and
- environment dynamics $x_{t+1}, r_t, d_t \sim p\big(x_{t+1}, r_t, d_t \mid x_{\leq t}, a_{\leq t}\big)$.

3.3 Vector-Quantized Variational Autoencoder

This thesis centers around the discrete latent representation of the input image observation, i.e., the discretized output of the VQ-VAE encoder. In the following, we introduce the architecture and training objective of the VQ-VAE, as well as how the discrete latent representations are obtained and used in the IRIS framework. After introducing the necessary notation for components of the VQ-VAE, we formally state our hypothesis.

A VQ-VAE a generative model consisting of a codebook $\mathcal{E}$, an encoder E, and a decoder D [40, 42], as illustrated in Figure 3.1.

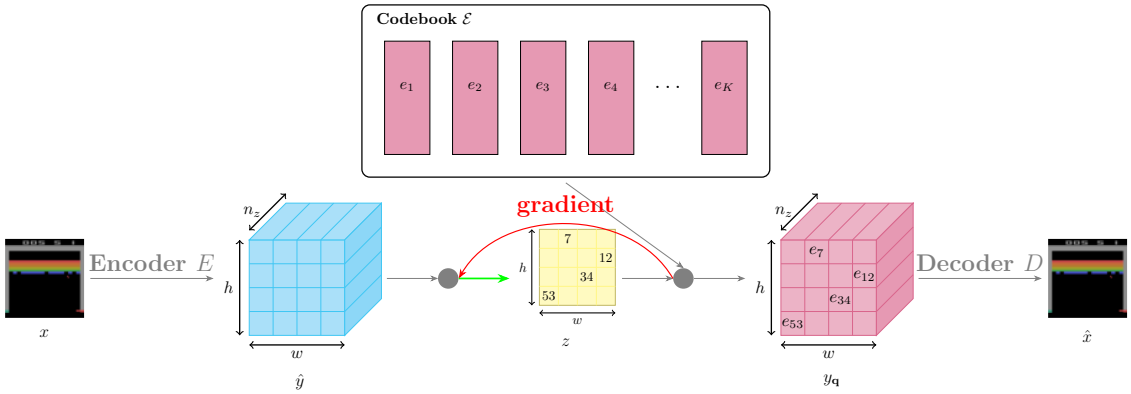


Figure 3.1: VQ-VAE, adapted from Oord *et al.* [40]. The green arrow indicates quantization of the encoder output $\hat{y}$ into a vector z of discrete tokens. During backpropagation, the gradient is directly copied [7] from the decoder to the encoder, as shown by the red arrow, enabling straight-through gradient estimation [7, 51] for discretization.

The codebook $\mathcal{E} = \{e_k\}_{k=1}^{K} \subset \mathbb{R}^{d_z}$ contains K latent vectors of dimension d_z.

The encoder $E : \mathbb{R}^{H \times W \times 3} \to \mathbb{R}^{h \times w \times d_z}$ maps the input image observation $\mathbf{x} \in \mathbb{R}^{H \times W \times 3}$ to latent vector $\hat{y} = E(\mathbf{x}) \in \mathbb{R}^{h \times w \times d_z}$. We discretize $\hat{y}$ to obtain $y_{\mathbf{q}}$ by replacing each element $\hat{y}_{ij} \in \mathbb{R}^{d_z}$ in $\hat{y}$ with its closest neighbor e_k in the codebook in terms of Euclidean distance (see the green arrow in Figure 3.1):

$$y_{\mathbf{q}} = \mathbf{q}(\hat{y}) := \left(\underset{e_k \in \mathcal{E}}{\arg\min} \|\hat{y}_{ij} - e_k\|_2 \right)_{i,j=1}^{h,w} \in \mathbb{R}^{h \times w \times d_z}, \tag{3.2}$$

where $\mathbf{q}(\bullet)$ is the element-wise quantization function. Equivalently, $y_{\mathbf{q}}$ can be represented as a vector of $h \cdot w$ indices $\mathbf{z}$ of the respective codebook vectors:

$$\mathbf{z} = \left(\mathbf{z}^1, \dots, \mathbf{z}^{h \cdot w}\right) = \mathbf{flatten} \circ \mathbf{index}\left(y_{\mathbf{q}}\right) = \mathbf{flatten}\left(\left(\underset{k \in \{1,\dots,K\}}{\arg\min} \|\hat{y}_{ij} - e_k\|_2 \right)_{i,j=1}^{h,w} \right) \in \{1, \dots, K\}^{h \cdot w} \tag{3.3}$$

We call $\mathbf{z}$ the *discrete representation* of the input image observation. These codebook indices $\mathbf{z}^1, \dots, \mathbf{z}^{h \cdot w} \in \{1, \dots, K\}$ are sometimes referred to as *(image) tokens*.

The decoder $D : \mathbb{R}^{h \times w \times d_z} \to \mathbb{R}^{H \times W \times 3}$ takes in $y_{\mathbf{q}}$ and reconstructs the input image $\hat{\mathbf{x}} = D(y_{\mathbf{q}}) = D(\mathbf{q}(E(\mathbf{x})))$.

Now that we have introduced the necessary concepts and notations, we can formally state our hypothesis:
Hypothesis : *For an attribute o_{attr} of an object o depicted in the input image $\mathbf{x}$, there exists a fixed set of positions $i \in \{1, \dots, h \cdot w\}$ in the discrete latent representation $\mathbf{z} = (\mathbf{z}^1, \dots, \mathbf{z}^{h \cdot w})$ of $\mathbf{x}$ whose values are sufficient to determine the value of o_{attr}.*

In IRIS, both the encoder E and the decoder D are implemented as CNNs interspersed with self-attention blocks [38]. The VQ-VAE is trained on previously collected image observations. During training, the parameters in the codebook, the encoder, as well as the decoder are learned by minimizing the following loss function:

$$\mathcal{L}(E, \mathcal{E}, D) = \|\mathbf{x} - \hat{\mathbf{x}}\|_1 + \|\text{sg}[E(\mathbf{x})] - y_{\mathbf{q}}\|_2^2 + \|\text{sg}[y_{\mathbf{q}}] - E(\mathbf{x})\|_2^2 + \mathcal{L}_{\text{perceptual}}(\mathbf{x}, \hat{\mathbf{x}}), \tag{3.4}$$

where $sg[\bullet]$ is the stop gradient operator. This loss function is the sum of four equally weighted terms:

- $\|\mathbf{x} - \hat{\mathbf{x}}\|_1$ is the reconstruction loss, measuring the L_1 distance between the input image observation $\mathbf{x}$ and the reconstructed image $\hat{\mathbf{x}}$,
- $\|\text{sg}[E(\mathbf{x})] - y_{\mathbf{q}}\|_2^2$ is the codebook loss, measuring the squared Euclidean distance between the quantized latent vector $y_{\mathbf{q}}$ and the detached encoder output $\text{sg}[E(\mathbf{x})]$, intended to encourage the chosen codebook vectors to be close to the encoder output,
- $\|\text{sg}[y_{\mathbf{q}}] - E(\mathbf{x})\|_2^2$ is the commitment loss, measuring the squared Euclidean distance between the detached quantized latent vector $\text{sg}[y_{\mathbf{q}}]$ and the encoder output $E(\mathbf{x})$, intended to encourage the encoder output to be close to the chosen codebook vectors, and
- $\mathcal{L}_{\text{perceptual}}(\mathbf{x}, \hat{\mathbf{x}})$ stands for perceptual loss [13, 27], measuring the sum of squared Euclidean distances between the input image observation $\mathbf{x}$ and its reconstruction $\hat{\mathbf{x}}$ at each selected layer of a pre-trained convolutional feature extractor, intended to capture the similarity between $\mathbf{x}$ and $\hat{\mathbf{x}}$ at the feature level.

3.4 Transformer

The Transformer architecture [48] has become a popular choice for sequence modeling tasks due to its parallelizability. Based on the implementation of minGPT [30], the Transformer G in IRIS models the environment dynamics: At each time step t, we use E and $\mathcal{E}$ to encode the episode fragment $(\mathbf{x}_0, a_0, \mathbf{x}_1, a_1, \dots, \mathbf{x}, a_t)$ and obtain a sequence

$$(\mathbf{z}_0^1, \dots, \mathbf{z}_0^{h \cdot w}, a_0, \mathbf{z}_1^1, \dots, \mathbf{z}_1^{h \cdot w}, a_1, \dots, \mathbf{z}_t^1, \dots, \mathbf{z}_t^{h \cdot w}, a_t)$$

of image tokens interleaved with actions. Given this token sequence, the Transformer G autoregressively predicts a sequence of image tokens $(\hat{z}_{t+1}^1, \dots, \hat{z}_{t+1}^{h \cdot w})$ representing the next frame, as well as the reward $\hat{r}_{t+1} \in \mathbb{R}$ and episode termination $\hat{d}_{t+1} \in \{0, 1\}$. In other words, G models the following three distributions in the latent space:

$$\begin{aligned} \text{Transition: } & \hat{\mathbf{z}}_{t+1} \sim p_G(\hat{\mathbf{z}}_{t+1} | \mathbf{z}_{\leq t}, a_{\leq t}) \text{ with } \hat{\mathbf{z}}_{t+1}^k \sim p_G\big(\hat{\mathbf{z}}_{t+1}^k | \mathbf{z}_{\leq t}, a_{\leq t}, \hat{\mathbf{z}}_{t+1}^{<k}\big) \\ \text{Reward: } & \hat{r}_t \sim p_G(\hat{r}_t | \mathbf{z}_{\leq t}, a_{\leq t}) \\ \text{Episode termination: } & \hat{d}_t \sim p_G(\hat{d}_t | \mathbf{z}_{\leq t}, a_{\leq t}) \end{aligned}$$

Note that the image tokens are predicted *autoregressively*, i.e., one token at a time, which means that the prediction of the k-th token at time step $t+1$ is also conditioned on $\hat{\mathbf{z}}_{t+1}^{<k} := (\hat{\mathbf{z}}_{t+1}^1, \dots, \hat{\mathbf{z}}_{t+1}^{k-1})$, the previously predicted tokens at the time step.

3.5 Actor-Critic

Actor-critic methods [47] learn an optimal policy by approximating a policy and a value function at the same time: The critic updates the value function using temporal difference learning, while the actor, informed by the value function, improves the policy using policy gradients to maximize the expected sum of discounted rewards.

In IRIS, both the actor and the critic are implemented as a convolutional block followed by a LSTM unit and a linear output layer [38]. The two components share parameters up to the output layers: the actor outputs a distribution over actions, while the critic estimates the scalar value of the current state. The actor-critic learning objectives and hyperparameters are adopted from DreamerV2 [22], which has delivered strong performance in Atari games.

4 Methods

This thesis aims to investigate the feasibility of extracting semantically meaningful information from the discrete latent representations of image observations learned by a VQ-VAE in the context of environment simulation. In Section 4.1, we outline the data collection process utilized for subsequent analysis. Section 4.2 and Section 4.3 introduce some additional notations and present two distinct approaches for analyzing the collect data, enabling us to assess the validity of our hypotheses (see Section 3.3). Lastly, in Section 4.4, we describe a visualization tool developed for qualitative exploration of the collected data.

4.1 Data Collection

We select a number of environments, and for each selected environment, we train a policy using IRIS. Upon completing the training process, we store the resulting policy π and the VQ-VAE with its encoder E, its codebook $\mathcal{E}$, and its decoder D. Using the trained policy π, we generate $N = 100$ episodes to collect image observations, actions, and rewards. Ground truth information for each image observation is retrieved using the Object-Centric Atari (OCAtari) library, while its discrete representation is derived through the VQ-VAE.

Environment Selection

We select a number of Atari games for analysis based on the following criteria:

1. **Policy Performance**: The policy π trained using IRIS in the environment should achieve competitive returns compared to baseline models used by Micheli *et al.* [38]. High returns indicate that the world model effectively simulates the environment, suggesting that the VQ-VAE encoder E successfully abstracts relevant information from the pixel observations.
2. **Visual Simplicity**: The environment should be visually simple, as filtering image observations based on object attributes is more manageable in such settings. Additionally, it is reasonable to assume that, for fixed h, w, H, and W, simpler images $\hat{\mathbf{x}} \in \mathbb{R}^{H \times W \times 3}$ are easier to reconstruct from discrete representations $\mathbf{z} \in \{1, \dots, K\}^{h \cdot w}$. This assumption is supported by Micheli *et al.* [38], who demonstrated that increasing the number of tokens in the discrete representation enhances the quality of frame reconstruction.

We select the following environments for our investigation:

- `BreakoutNoFrameskip-v4`

- `ChopperCommandNoFrameskip-v4`
- `DemonAttackNoFrameskip-v4`
- `PongNoFrameskip-v4`

Figure 4.1 shows example image observations from each of the selected environments. The performance of the policies trained in these environments are detailed in Section 5.1. For the purpose of illustration, we use `BreakoutNoFrameskip-v4` (hereafter referred to as `Breakout`) as a representative example in subsequent discussions to demonstrate our methods.

(a) `Breakout` (b) `ChopperCommand` (c) `DemonAttack` (d) `Pong`

Figure 4.1: Example image observation in each of the selected Atari games. Names of the games abbreviated for readability.

Policy Training

We use the implementation provided by Micheli *et al.* [38], employing the default hyperparameter configuration to train policies in the selected environments and reproduce the returns reported in their study [38].

Object-Centric Atari

Delfosse *et al.* [11] introduced the OCAtari wrapper, built on top of the ALE framework [6], to enable the integration of object-centric representations into RL. For each image observation, OCAtari maintains a list of depicted objects, where each object is characterized by its category, position, size, and `RGB` values. Additionally, depending on the specific game, certain objects may include other attributes. For instance, in `Boxing`, both the player and the opponent possess properties such as `right_arm_length` and `left_arm_length`.

Object properties can be extracted using two modes: the Visual Extraction Method (VEM) and the RAM Extraction Method (REM). In VEM, objects are identified from pixel frames through color-based filtering combined with prior knowledge about object positions. In contrast, REM interprets object properties directly from the emulator's RAM, enabling the tracking of dynamic attributes such as moving and blinking objects. For this study, we choose the REM mode to ensure consistent extraction of object properties across all games.

During data collection, we substitute the ALE environment with the corresponding OCAtari wrapper to retrieve, for each image observation $\mathbf{x}$, a list $\mathbf{o}$ of depicted objects along with their attributes.

Generating Discrete Representation

To generate the discrete latent representations $\mathbf{z}$ of an image observation $\mathbf{x}$, we initialize and load the VQ-VAE, resize $\mathbf{x}$, pass it through the VQ-VAE encoder E, and discretize the resulting latent vector using the codebook $\mathcal{E}$.

Summary of Collected Data

For each selected environment, we unroll $N = 100$ episodes by following the trained policy π. Each episode consists of a sequence of image observations $\mathbf{x} \in \mathbb{R}^{H \times W \times 3}$, actions $a \in \{1, \dots, |\mathcal{A}|\}$, and rewards $r \in \mathbb{R}$. For each frame $\mathbf{x}$, we retain its corresponding list of objects $\mathbf{o} = \{o^1, \dots, o^{|\mathbf{o}|}\}$ and its discrete latent representation $\mathbf{z} \in \{1, \dots, K\}^{h \cdot w}$. We define the tuple $(\mathbf{x}, \mathbf{o}, \mathbf{z})$ as an `Observation`. In addition, for each game, we store the vectors $e_1, \dots, e_K \in \mathbb{R}^{d_z}$ from the VQ-VAE codebook $\mathcal{E}$.

4.2 Finding Relevant Positions for Object Attributes According to Subsets

In our first approach, we start by defining subsets based on each selected object attribute. To reduce the complexity of our problem, we perform clustering on the codebook vectors. We also build an interactive tool we call "directed" perturbation that can be applied to `Observation` pairs within a subset.

On the one hand, this tool provides us with a qualitative understanding of the effect of changing the discrete representation of one `Observation` to that of the other, position-by-position. On the other hand, with the help of a semi-quantitative evaluation scheme detailed in Appendix A.1, this tool also aids us in evaluating and identifying the most suitable clustering algorithm for our purpose, enabling us to draw meaningful conclusions for this approach.

4.2.1 Defining Subsets

Two arbitrary `Observations` from our collected dataset differ more likely than not in multiple object attributes. To isolate the effect of a single object attribute o^j_{attr}, we define subsets of `Observations` where every object attribute except for o^j_{attr} is kept constant.

A subset $S^{\mathbf{o}}_{\text{attr}}$ of `Observations` is defined by an object attribute o^j_{attr} (e.g., `ball_position` in `Breakout`) and a representative `Observation` with object-centric representation $\mathbf{o}$. This subset

includes all Observations whose object-centric representations share every object attribute with $\mathbf{o}$ except for o^j_{attr}, as formally described below[1]:

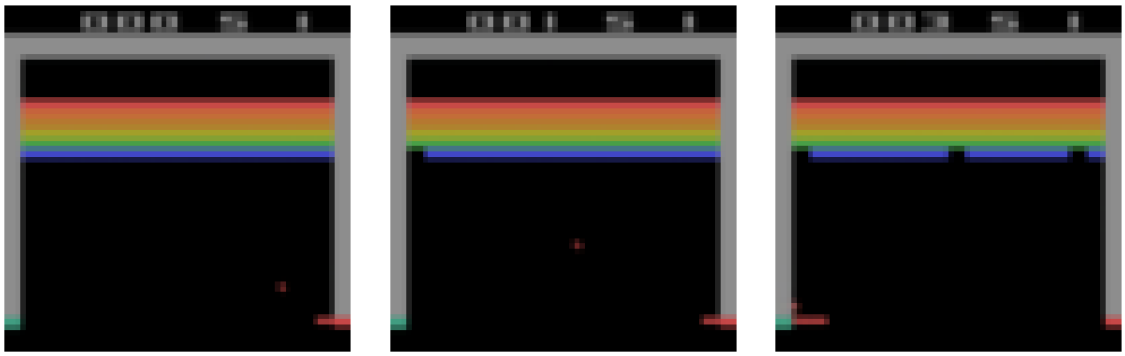
$$S^{\mathbf{o}}_{\text{attr}} = \{(\mathbf{x}, \mathbf{z}, \mathbf{o})\} \cup \{(\mathbf{x}', \mathbf{z}', \mathbf{o}') | o'^j_{\text{attr}} \neq o^j_{\text{attr}} \wedge \forall j' \neq j, \text{attr}' \neq \text{attr} : o'^{j'}_{\text{attr}'} = o^{j'}_{\text{attr}'}\}. \tag{4.1}$$

In this formulation, the subset $S^{\mathbf{o}}_{\text{attr}}$ contains the Observation $(\mathbf{x}, \mathbf{z}, \mathbf{o})$ and all Observations $(\mathbf{x}', \mathbf{z}', \mathbf{o}')$ that differ only in the specified object attribute o^j_{attr}, while maintaining consistency across all other attributes. To be precise, $S^{\mathbf{o}}_{\text{attr}}$ is a multiset, meaning the same Observation can appear multiple times within the same subset.

For each attribute o^j_{attr} of interest, we find such subsets through the following steps:

1. Specify the set $\mathcal{F}$ of all the other *relevant*[2] object attributes that should be kept constant inside a subset.
2. Iterate over the object-centric representation of the Observations in the entire collected dataset, and group them according to the values of the attributes in $\mathcal{F}$ in each Observation.
3. Sort these groups according to the number $|\mathbf{set}(S^{\mathbf{o}}_{\text{attr}})|$ of unique Observations in descending order[3]; the top four groups are the subsets we define for the attribute o^j_{attr}.

Figure 4.2 shows the image observations of the representative Observations for three different subsets defined for ball_position in Breakout. Here, $\mathcal{F}$ consists of player_position and all attributes of the BlockRow objects.



(a) Representative Observation $(\mathbf{x}, \mathbf{z}, \mathbf{o})$ for the subset $S^{\mathbf{o}}_{\text{ball\_pos}}$

(b) Representative Observation $(\mathbf{x}', \mathbf{z}', \mathbf{o}')$ for the subset $S^{\mathbf{o}'}_{\text{ball\_pos}}$

(c) Representative Observation $(\mathbf{x}'', \mathbf{z}'', \mathbf{o}'')$ for the subset $S^{\mathbf{o}''}_{\text{ball\_pos}}$

Figure 4.2: Three representative frames illustrating three subsets defined for the attribute ball_position in Breakout, where $\mathcal{F}$ consists of player_position and all attributes of the BlockRow objects.

[1] We recall that each Observation is a 3-tuple $(\mathbf{x}, \mathbf{z}, \mathbf{o})$.

[2] Some attributes are not visible or are constant for the duration of an episode, they are thus not relevant for our analysis

[3] The operator $\mathbf{set}(\cdot)$ converts a multiset into a set by ensuring each element appears at most once.

For later analysis, we aim to find subsets $S^{\mathbf{o}}_{\text{attr}}$ with large $|\mathbf{set}(S^{\mathbf{o}}_{\text{attr}})|$; thus, the choice of o^j_{attr} and $\mathcal{F}$ matters. For instance, in `Breakout`, we prioritize attributes such as `ball_position` and `player_position` to define the subsets, rather than focusing on the existence of a specific block in a given row. The latter approach would require fixing the positions of the player and the ball, as well as the states of all other blocks, leaving us with very few pair-wise different frames to analyze within that subset.

4.2.2 Clustering Codebook Vectors

We observe that the indices of some codebook vectors never appear in the discrete representations of the collected frames. For instance, in `Breakout`, these codebook vectors are colored blue in the 2D[4] visualization of the codebook space shown in Figure 4.3. For all subsequent discussions, we exclude these "unused" codebook vectors, formally defined as follows:

$$\{e_k \mid \forall \mathbf{z} = \left(z^1, \dots, z^{h \cdot w}\right), \forall i \in \{1, \dots, h \cdot w\} : z^i \neq k\}. \tag{4.2}$$

The resulting codebook, containing only the "used" codebook vectors, is given by:

$$\mathcal{E}' = \{e_k \mid \exists \mathbf{z} = \left(z^1, \dots, z^{h \cdot w}\right), \exists i \in \{1, \dots, h \cdot w\} : z^i = k\} \tag{4.3}$$

This refined codebook has a size of $|\mathcal{E}'| = K'$, where $K' \leq K$.

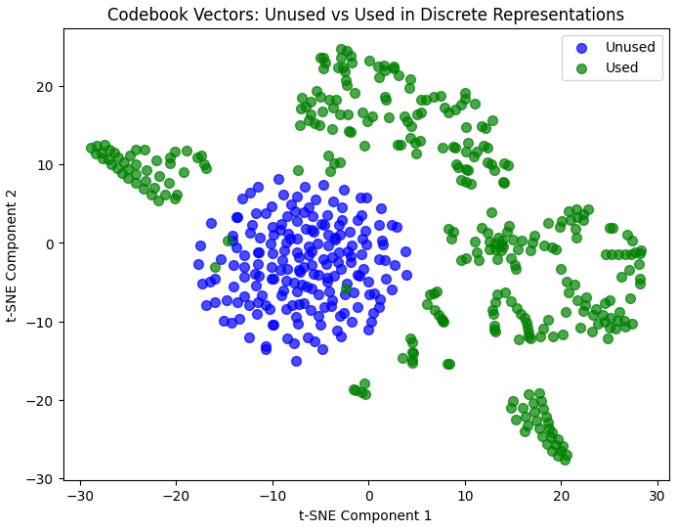


Figure 4.3: Visualization of the codebook vectors for `Breakout`, with dimensionality reduced to 2 using t-SNE. "Unused" codebook vectors are shown in blue, while "used" codebook vectors are shown in green.

However, the number of the remaining codebook vectors still remains quite large. Furthermore, when comparing position-wise the discrete representations $\mathbf{z}_1$ and $\mathbf{z}_2$ of two arbitrary image

[4]The dimensionality of the codebook vectors are reduced from d_z to 2 using t-distributed Stochastic Neighbor Embedding (t-SNE).

observations $\mathbf{x}_1$ and $\mathbf{x}_2$ from $S^{\mathbf{o}}_{\text{attr}}$, we find that changes in codebook vector indices (or tokens) occur at nearly every position. This indicates that the tokens function in a compositional manner: the effect of token k at position i depends on tokens at certain other positions. Consequently, transforming the reconstruction $\hat{\mathbf{x}}_1$ of $\mathbf{z}_1$ into $\hat{\mathbf{x}}_2$ requires cumulative modifications to $\mathbf{z}_1$ across multiple positions until it matches $\mathbf{z}_2$, even when the two frames differ only in o^j_{attr}.

We assume that codebook vectors $e_k, e_{k'}, \ldots$ which are close to each other in the codebook space have similar effects on the reconstruction $\hat{\mathbf{x}}$ of the discrete representation $\mathbf{z}$. By grouping these vectors into a single cluster c_l, we effectively reduce the complexity from the number K' of "used" codebook vectors $e_1, \ldots, e_{K'}$ to the number C of clusters $c_1, \ldots, c_C$, where $C \ll K'$.

When comparing each position i in the discrete representations, only changes in cluster membership are considered relevant for the object attribute o^j_{attr}. As such, we disregard the exact tokens $\mathbf{z}^i_1, \mathbf{z}^i_2$, focusing instead on their corresponding clusters $\textbf{cluster}(\mathbf{z}^i_1), \textbf{cluster}(\mathbf{z}^i_2)$, where the operator $\textbf{cluster}(\cdot)$ maps each codebook index to the cluster to which its corresponding vector belongs, defined as:

$$\textbf{cluster}(k) = l \iff e_k \in c_l. \tag{4.4}$$

We experiment with various clustering algorithms to group the codebook vectors. For algorithms that perform better with lower-dimensional data, we first apply principal component analysis (PCA) to reduce the dimensionality of the codebook vectors from d_z to d_r, where $d_r \ll d_z$, before performing clustering.

In the next section, we introduce an interactive visualization tool that helps us qualitatively compare and evaluate different clustering results.

4.2.3 "Directed" Perturbation Within a Subset

We develop an interactive visualization tool that can be applied to a pair of `Observations` within a subset $S^{\mathbf{o}}_{\text{attr}}$, which we denote as $\texttt{Observation}_1$ and $\texttt{Observation}_2$ in subsequent discussions. This tool allows us to qualitatively compare the results of clustering with the effects of progressively changing the discrete representation $\mathbf{z}_1$ of $\texttt{Observation}_1$ position-by-position to match $\mathbf{z}_2$ of $\texttt{Observation}_2$. This "directed" perturbation, illustrated in Figure 4.4, is conducted as follows:

On the left, we display the reconstructed first frame $\hat{\mathbf{x}}_1$

In the middle, we display all the positions $i = 1, \ldots, h \cdot w$ in the discrete representation $\mathbf{z}_{\text{perturb}}$, showing both $\mathbf{z}^i_1$ and $\mathbf{z}^i_2$. These positions are grouped into three categories:

1. **No change in codebook index**: Positions i where the codebook index remains the same, i.e.,

$$\mathbf{z}^i_1 = \mathbf{z}^i_2, \tag{4.5}$$

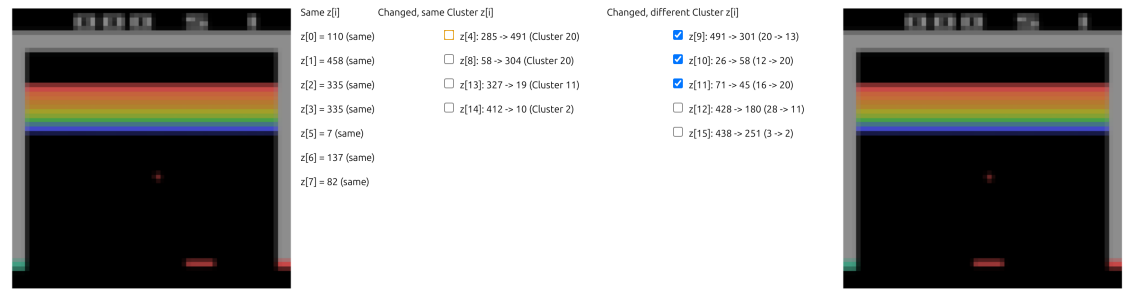


Figure 4.4: Visualization of the interactive "directed" perturbation of the discrete representations of `Observation` pairs within a subset.

2. **Change within the same cluster**: Positions i where the token changes but remains inside the same cluster, i.e.,

$$\mathbf{z}_1^i \neq \mathbf{z}_2^i \wedge c(\mathbf{z}_1^i) = c(\mathbf{z}_2^i), \tag{4.6}$$

3. **Change across clusters**: Positions i where both the token and the cluster to which it belongs changes, i.e.,

$$\mathbf{z}_1^i \neq \mathbf{z}_2^i \wedge c(\mathbf{z}_1^i) \neq c(\mathbf{z}_2^i). \tag{4.7}$$

A checkbox is placed at each position i where $\mathbf{z}_1^i \neq \mathbf{z}_2^i$. When the box is checked, the token at that position is updated to the codebook index from the second discrete representation ($\mathbf{z}_2^j$); otherwise, it retains the value from the first discrete representation ($\mathbf{z}_1^j$). At all other positions j, the token remain fixed and unchangeable, i.e., $\mathbf{z}_{\text{perturb}}^j = \mathbf{z}_1^j = \mathbf{z}_2^j$. We define this mechanism as follows:

$$\mathbf{z}_{\text{perturb}}^i = \begin{cases} \mathbf{z}_2^i & \text{if } \mathbf{z}_1^i \neq \mathbf{z}_2^i \text{ and box checked,} \\ \mathbf{z}_1^i & \text{otherwise.} \end{cases} \tag{4.8}$$

On the right, we display the reconstruction $\hat{\mathbf{x}}_{\text{perturb}}$ of the perturbed discrete representation $\mathbf{z}_{\text{perturb}}$ determined by the state of the checkboxes in the middle:

$$\hat{\mathbf{x}}_{\text{perturb}} = D\Big((\mathbf{flatten} \circ \mathbf{index})^{-1}(\mathbf{z}_{\text{perturb}})\Big), \tag{4.9}$$

where $\mathbf{index}(\bullet)$ is the element-wise operation that maps each codebook vector in $\mathcal{E}'$ to its corresponding index, and $\mathbf{flatten}(\bullet)$ is the operation that flattens the $h \times w$ matrix to an $h \cdot w$-dimensional vector (see Section 3.3). If all the boxes are checked, the reconstructed image corresponds to the reconstruction $\hat{\mathbf{x}}_2$ of the second image observation. Conversely, if none are checked, the reconstruction matches $\hat{\mathbf{x}}_1$ (identical to the image on the left).

Cluster Visualization in Codebook Space with "Directed" Perturbation

An optional visualization displays the clusters in the codebook space, with the codebook vectors projected to 2D or 3D using t-SNE, as illustrated in Figure 4.5. The vectors $e_1, \ldots, e_{K'}$ take on different colors according to their cluster membership and distinct shapes based on their category, defined using the help of $\mathbf{z}_1$ and $\mathbf{z}_2$: [5]

1. **Unique in $\mathbf{z}_1$**: Codebook vectors e_k where there exists a position i such that $\mathbf{z}_1^i = k \neq \mathbf{z}_2^i$,
2. **Unique in $\mathbf{z}_2$**: Codebook vectors e_k where there exists a position i such that $\mathbf{z}_1^i \neq k = \mathbf{z}_2^i$,
3. **Common**: Codebook vectors e_k where there exists a position i such that $\mathbf{z}_1^i = \mathbf{z}_2^i = k$,
4. **Other**: All other vectors in the codebook $\mathcal{E}'$.

For each position i where $\mathbf{z}_1^i \neq \mathbf{z}_2^i$, a line is drawn between the vectors with indices $\mathbf{z}_1^i$ and $\mathbf{z}_2^i$. This visualization facilitates a qualitative comparison of the spatial relationships between the vectors in the codebook space and their corresponding effects on the reconstructed observations.

Cluster Evaluation with "Directed" Perturbation

According to our assumptions, an ideal clustering would group codebook vectors into the same cluster if they have similar effect on the reconstructed frame. To qualitatively evaluate how well a clustering result aligns with these assumptions, we interact with the aforementioned checkboxes and observe the effect of toggling them on the reconstruction $\hat{\mathbf{x}}_{\text{perturb}}$:

- **No Change Possible**: For positions i where $\mathbf{z}_1^i = \mathbf{z}_2^i$, no checkboxes are present since the codebook indices are identical.
- **No or Minimal Change Expected**: For positions i where $\mathbf{z}_1^i \neq \mathbf{z}_2^i$ and $c(\mathbf{z}_1^i) = c(\mathbf{z}_2^i)$, toggling the checkbox should result in either no change or minimal change in $\hat{\mathbf{x}}_{\text{perturb}}$.
- **Noticeable Change Expected**: For positions i where $\mathbf{z}_1^i \neq \mathbf{z}_2^i$ and $c(\mathbf{z}_1^i) \neq c(\mathbf{z}_2^i)$, toggling the checkbox should cause a noticeable change in $\hat{\mathbf{x}}_{\text{perturb}}$.

We conduct this comparison across multiple subsets $S_{\text{attr}}^{\mathbf{o}}, S_{\text{attr}}^{\mathbf{o}'}, \ldots$, each defined by the same object attribute o_{attr}^j but associated with pairwise distinct representative `Observations` $(\mathbf{x}, \mathbf{z}, \mathbf{o})$, $(\mathbf{x}', \mathbf{z}', \mathbf{o}'), \ldots$. This approach is intended to ensure that the expected clustering result remains robust and generalizable across different regions of the observation space.

Once the clustering result most aligned with the "directed" perturbations has been selected (as detailed in Appendix A.1), we can assume the following from this point onward: changes in the discrete representation across cluster boundaries are relevant to the object attribute o_{attr}^j

[5] We assign a category to each codebook vector by iterating over all positions i in the discrete representation and checking the corresponding codebook index k. If there exists a token k with $\mathbf{z}_1^i = \mathbf{z}_1^j = k$ for some $i < j$, then k is assigned to the category according to position j, since whatever category k belongs to at position i is overwritten in our implementation. A misclassification of Category 1 or 2 as Category 3 is possible, but does not have a significant impact on the visualization, since the vector would still be connected by a line to its counterpart, making its category evident.

Figure 4.5: Visualization of clustered codebook vectors projected to 2D using t-SNE. Colors indicate cluster membership, while shapes denote categories. Lines connect vectors $\mathbf{z}_1^i$ and $\mathbf{z}_2^i$ for positions i where $\mathbf{z}_1^i \neq \mathbf{z}_2^i$. Hovering mouse over individual elements displays additional details, such as cluster membership, codebook index, and position in the discrete representation. The corresponding "directed" perturbation with reconstructed images is shown in Figure 4.4.

of interest. Building on this assumption, we proceed to analyze the relevance of individual positions in the discrete representation to the attribute o_{attr}^j:

For each subset $S_{\text{attr}}^{\mathbf{o}}$ and each position i in the discrete representation, we count the number of cross-cluster changes $n^i_{o_{\text{attr}}^j,\mathbf{o}}$ for `Observation` pairs within $S_{\text{attr}}^{\mathbf{o}}$. These counts are visualized in a histogram, where the i-th bin corresponds to the value of $n^i_{o_{\text{attr}}^j,\mathbf{o}}$.

To evaluate position relevance across all subsets $S^{\mathbf{o}}_{\text{attr}}, S^{\mathbf{o}'}_{\text{attr}}, \ldots$ defined by the same attribute o^j_{attr}, we sum up the counts of cross-cluster changes at each position i over all the subsets. This aggregated count, $n^i_{o^j_{\text{attr}}}$, is defined as:

$$n^i_{o^j_{\text{attr}}} = \sum_{\mathbf{x}} n^i_{o^j_{\text{attr}},\mathbf{o}}. \tag{4.10}$$

The aggregated counts are visualized in another histogram, where the i-th bin represents the total count $n^i_{o^j_{\text{attr}}}$ for position i. In each histogram, the size of the i-th bin size serves as a proxy for the relevance of position i to o^j_{attr}, either specific to the corresponding subset or across all subsets.

We acknowledge that part of this approach—namely, selecting the appropriate clustering algorithm (detailed in Appendix A.1), is mostly qualitative in nature, requiring manual intervention and relying on human perception. Furthermore, the subsets we define are inherently arbitrary and may not comprehensively capture the full range of `Observation`s for the given attribute. To complement these limitations, in the next section, we propose a quantitative method that approaches our hypothesis from a different angle.

4.3 Random Forest Regressors for Object Attributes

To provide an additional quantitative perspective on the findings obtained using the method described in Section 4.2, we turn our attention to feature importance of random forest regressors.

For each object attribute o^j_{attr} of interest, we prepare the following datasets:

1. **Individual Subsets**: Subsets $S^{\mathbf{o}}_{\text{attr}}, S^{\mathbf{o}'}_{\text{attr}}, \ldots$ defined by o^j_{attr} and pairwise distinct representative frames $\mathbf{x}, \mathbf{x}', \ldots$.
2. **Union of Subsets**: The union of all the subsets $S^{\mathbf{o}}_{\text{attr}}, S^{\mathbf{o}'}_{\text{attr}}, \ldots$.
3. **Complete Dataset**: The entire collected dataset for the given environment.

Each of the above datasets is split into training and test sets, with the training set comprising 80% of the data and the test set containing the remaining 20%. A random forest regressor is then trained to predict the attribute o^j_{attr} using the training set. The regression is performed using the following data:

- **Features**: The discrete representations $\mathbf{z}$ of all image observations $\mathbf{x}$ in the dataset. Each position $i = \{1, \ldots, h \cdot w\}$ is treated as an individual feature, resulting in $h \cdot w$ features in total. These features are categorical, as the values of $\mathbf{z}^i$ correspond to indices of vectors in the codebook.
- **Target**: The value(s) of o^j_{attr}. If o^j_{attr} is a tuple (e.g., `ball_position`, represented as a 2-tuple), we split it into multiple real-valued targets. Each target value is normalized to a real number between 0 and 1.

To assess the importance of each position in the discrete representation for predicting the object attribute o^j_{attr}, we compute feature importance over the test set, using the `permutation_importance` function from the `scikit-learn` library [41]. For each position i, the values of the i-th feature are shuffled across all frames in the test set, and the resulting increase in prediction error is measured. A greater increase in error indicates that feature i plays a more significant role in determining o^j_{attr}. Using these values, we construct a bar chart where the i-th bin represents the feature importance of position i in the regression task.

This process produces a bar chart for each of the datasets described above. Finally, we compare these bar charts to their counterparts obtained using the method presented in Section 4.2—except for the entire dataset, which is unique to this approach. This comparison allows us to assess whether the results align and draw further conclusions about our hypothesis.

4.4 Exploring Meaning of Individual Codebook Vectors

As discussed in the previous section, one of our key assumptions is that the effect of token $k = \mathbf{z}^i$ at position i in the discrete latent representation $\mathbf{z}$ on the reconstructed image $\hat{\mathbf{x}} = D\Big((\mathbf{flatten} \circ \mathbf{index})^{-1}(\mathbf{z})\Big)$ is influenced by tokens at other positions. And it is only through specific combinations that these tokens at their individual positions collectively contribute to a concrete change in the value of a particular object attribute in $\hat{\mathbf{x}}$.

Nonetheless, if certain positions i are identified as particularly relevant for an object attribute o^j_{attr}, a promising next step is to explore the "meaning" of the specific tokens k that appear either most frequently or most rarely at these significant positions. The rationale is that we suspect, for the position i and within subset $S^{\mathbf{o}}_{\text{attr}}$ with duplicates removed, tokens that occur highly frequently probably encode some notion of baseline for the value of attribute o^j_{attr}, while the most rarely occurring tokens might represent special cases or deviation from this baseline.

An obstacle, however, is that, because the codebook indices k are categorical values, there is no inherent "zero" or `None` value in the codebook space to represent the absence of a token at positions $j \neq i$. Assigning the index of "unused" codebook vectors to $\mathbf{z}^j$ could still introduce unknown confounding noise into the reconstruction. To address this limitation, we use a discrete representation $\mathbf{z}_{\text{ref}}$ as reference, and define the effect $\hat{\mathbf{x}}_{k,i}(\mathbf{z}_{\text{ref}})$ of token k at position i as a function of the reference discrete representation $\mathbf{z}_{\text{ref}}$. We test the following two options for $\mathbf{z}_{\text{ref}}$:

1. Any discrete representation $\mathbf{z}$ of an `Observation` in $S^{\mathbf{o}}_{\text{attr}}$, where $\mathbf{z}^i \neq k$;
2. An artificially constructed discrete representation consisting of arbitrary "used" or "unused" tokens.

To calculate the effect of token k at position i with the help of the reference discrete representation $\mathbf{z}_{\text{ref}}$, we modify $\mathbf{z}_{\text{ref}}$ by replacing the value $\mathbf{z}^i_{\text{ref}}$ with k:

$$\mathbf{z}^j_{\text{ref,perturb}} = \begin{cases} \mathbf{z}^j_{\text{ref}} & \text{if } j \neq i, \\ k & \text{if } j = i. \end{cases} \tag{4.11}$$

For visualization, we reconstruct the perturbed discrete representation $\hat{\mathbf{x}}_{\text{ref,perturb}}$ using the VQ-VAE decoder and take the absolute value of the difference of the RGB values between $\hat{\mathbf{x}}_{\text{ref,perturb}}$ and the reconstructed reference frame $\hat{\mathbf{x}}_{\text{ref}}$:

$$\begin{aligned} \hat{\mathbf{x}}_{k,i}(\mathbf{z}_{\text{ref}}) &= |\hat{\mathbf{x}}_{\text{ref,perturb}} - \hat{\mathbf{x}}_{\text{ref}}| \\ &= |D\Big((\mathbf{flatten} \circ \mathbf{index})^{-1}\big(\mathbf{z}_{\text{ref,perturb}}\big)\Big) - D\Big((\mathbf{flatten} \circ \mathbf{index})^{-1}(\mathbf{z}_{\text{ref}})\Big)|. \end{aligned} \tag{4.12}$$

We anticipate that the visualized effect of token k at position i will, at most, provide a partial depiction of the object o^j with attribute o^j_{attr}. Attributes of other objects, along with elements such as the background or pixel patterns representing scores obtained by the player or lives remaining, may also appear in the reconstruction. This aligns with the expectation that features extracted by the VQ-VAE are interdependent and compositional, meaning each individual token likely encodes partial information about multiple object attributes.

4.5 Summary

In this chapter, we outlined the processes for generating and analyzing data to investigate the relationship between the discrete representation of image observations and object attributes depicted in the reconstructed images. We presented two distinct methods, each providing a different perspective on this problem, and proposed an approach for exploring the meaning of individual codebook vectors. In the next chapter, we present the results obtained from these approaches.

5 Results

In this chapter, we present the results of our investigation into the discrete latent representations abstracted from pixel observations by the learned VQ-VAE of the IRIS world model. We first describe the data we collected, followed by results we obtained via the two approaches described in Section 4.2 and Section 4.3. Finally, we report on what we have observed from the effect of individual codebook vectors on the reconstructed image.

5.1 Policy Training and Returns Reproduction

As discussed in Section 4.1, we select the following environments for our investigation:

- `BreakoutNoFrameskip-v4`
- `ChopperCommandNoFrameskip-v4`
- `DemonAttackNoFrameskip-v4`
- `PongNoFrameskip-v4`

We use the single-GPU implementation[1] of IRIS, released by Micheli *et al.* [38], to train a policy on each of these selected environments. For the training, we adopted the same hyperparameters as in the original implementation, with the exception of the batch size for the VQ-VAE[2], which we halve from 256 to 128 to fit the available memory.

Although this modification has, to some extent, a negative impact on the performance of resulting policy, we still achieve comparable returns to what is reported by Micheli *et al.* [38], as shown in Table 5.1. Notably in `ChopperCommand` and `DemonAttack`, our trained policies achieved better returns than reported by Micheli *et al.* [38].

Returns	`Breakout`	`ChopperCommand`	`DemonAttack`	`Pong`
Micheli *et al.*	83.70	1565.00	2034.40	14.60
Our experiment	72.23	1904.00	2187.35	12.03

Table 5.1: Comparison of returns from paper and experiment in selected environments. The scores are calculated as averaged return over 100 episodes. Names of environments are shortened for readability.

[1] `https://github.com/eloialonso/iris`

[2] which corresponds to `training.tokenizer.batch_num_samples` in `config/trainer.yaml`

5.2 Gathering Data

To collect data for analysis, we modify some available functionalities in the original implementation[3] and replace the `gymnasium` environment with the corresponding OCAtari wrapper.

For each environment, we gather the following:

- $K = 512$ codebook vectors of dimension $d_z = 512$
- over 100 episodes:
  - image observations $\mathbf{x} \in \mathbb{R}^{210\times160\times34}$
  - discrete representations $\mathbf{z} \in \{0, \dots, 511\}^{4\cdot4}$
  - object-centric representations $\mathbf{o}$ as lists of objects, where each object is represented by its category, `RGB` value, position, size, and other additional properties depending on the object in the game.

The OCAtari wrapper does not provide the values of the scores the player (and the enemy, if existent) achieves at each time step in the object-centric representation. They are, however, major components in the image observations of `Pong`, and thus important object attributes for defining subsets. Therefore, we modified the OCAtari wrapper for `Pong` to additionally store the player score and the enemy score for each time step.

5.3 Relevant Positions for Object Attributes

In this section, we present the results we obtained by following each of the two approaches detailed in Section 4.2 and Section 4.3.

Choosing Object Attributes

For each game, we choose one or more object attributes for later analysis, as shown in Table 5.2.

Game	Object Attribute	Description
`Breakout`	`ball_position`	x- and y-coordinates of the `Ball` object
`ChopperCommand`	`player_position`	x- and y-coordinates of the `Player` object
`DemonAttack`	`player_position`	x-coordinate of the `Player` object
`Pong`	`player_position`	y-coordinate of the `Player` object
`Pong`	`enemy_position`	y-coordinate of the `Enemy` object

Table 5.2: Object attributes we select for investigation and their descriptions for different games.

[3] primarily, `eval.py` and `Collector.collect()`

[4] We store image observations in their original resolution, but the figures in this thesis are downscaled images that are used as input for the VQ-VAE as well as the trained policy. The latter can be easily obtained from the former by applying a resizing operation.

5.3.1 Results According to Subsets and Clustering

In this subsection, we describe the subsets we defined for each game and object attribute, and list the clustering algorithms we select for each attribute using the defined subsets in the method described in Section 4.2. Lastly, we present results specific to the defined subsets about the relevance of positions in the discrete representation to each selected object attribute.

Subset Definition

As detailed in Section 4.2.1, we specify, for each attribute o^j_{attr}, a set $\mathcal{F}$ of all the other relevant object attributes that should be kept constant inside a subset, and calculate the subsets according to the object-centric representations of the `Observations` collected. Table 5.3 shows $\mathcal{F}$ for each attribute of interest. Each $\mathcal{F}$ yields four subsets for subsequent analysis.

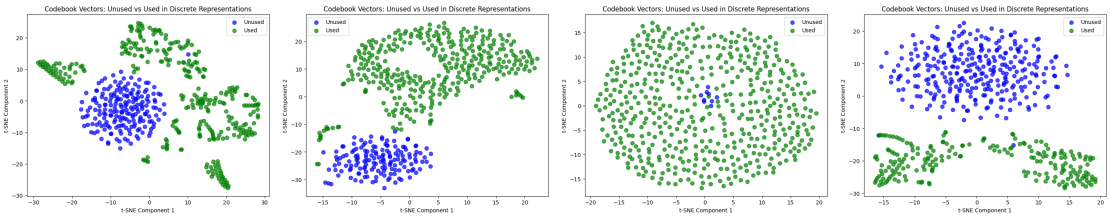
Game	Object Attribute	$\mathcal{F}$
`Breakout`	`ball_position`	`player_position` and all attributes of `BlockRow` objects
`ChopperCommand`	`player_position`	`player_orientation` and all attributes of `Shot`, `Bomb`, `Truck`, `EnemyHelicopter`, and `EnemyPlane` objects, if visible
`DemonAttack`	`player_position`	all attributes of `Enemy` objects
`Pong`	`player_position`	`ball_position` and `enemy_position`
`Pong`	`enemy_position`	`ball_position` and `player_position`

Table 5.3: The corresponding set $\mathcal{F}$ of other object attributes used to defined subsets for each selected object attribute of interest.

We note that in `DemonAttack`, every time the agent successfully eliminates all the existing `Enemy` object, the new `Enemy` objects take on a different visual representation (presumably the game progresses to a new "level"). This visual difference is, however, not reflected in the object-representations. We, therefore, first separated the image observations into different levels where the `Enemy` objects share the same visual representation, and then perform the aforementioned process of grouping `Observations` into subsets.

Clustering Codebook Vectors Informed by 'Directed' Perturbation

We observe that the number of codebook vectors that actually appear in the discrete representations across the entire collected dataset varies significantly between games, as shown in Figure 5.1. This indicates different levels of visual complexity in the games. For the subsequent clustering, we remove the "unused" codebook vectors, which are colored blue in Figure 5.1. As can be seen in the figure, the "used" codebook vectors distribute in distinct ways in the codebook space for different games. Especially for `DemonAttack`, these vector are much more evenly distributed than in the other games, which might be at least in part due to its visual complexity.



(a) `Breakout` (b) `ChopperCommand` (c) `DemonAttack` (d) `Pong`

Figure 5.1: Visualization of the VQ-VAE codebook vectors for different Atari games, with dimensionality reduced to 2 using t-SNE. "Unused" codebook vectors are shown in blue, while "used" codebook vectors are shown in green.

We have the following clustering algorithms from the `scikit-learn` library [41] as candidates to group the codebook vectors into clusters:

- `AgglomerativeClustering`,
- `SpectralClustering`,
- `HDBSCAN`, and
- `GaussianMixture`.

Based on properties of these algorithms [9, 36, 39], we apply `SpectralClustering` to the codebook vectors of reduced[5] dimensionality d_r and the other algorithms to the codebook vectors of original dimensionality $d_z = 512$. For those algorithms that require specifying the number of clusters C, we choose C to be 30 for `Pong`, 37 for `Breakout`, 37 for `ChopperCommand`, and 25 for `DemonAttack` based on trial and error in preliminary analysis. We evaluate each clustering result with the help of the visualization of 'directed' perturbation of the discrete representations of `Observation` pairs inside individual subsets, as detailed in Section 4.2.

According to the scores we obtained using the evaluation method described in Appendix A.1, we select the following clustering algorithm, as shown in Table 5.4, for the selected games object attributes.

Game	**Object Attribute**	**Clustering Algorithm**	Dim. Reduction
`Breakout`	Ball position	`Agglomerative`	No
`ChopperCommand`	Helicopter position	`Spectral`	Yes
`DemonAttack`	Laser canon position	`Spectral`	Yes
`Pong`	Player position	`GaussianMixture`	No
`Pong`	Enemy paddle position	`GaussianMixture`	No

Table 5.4: Selected clustering algorithms for each game and object attribute based on evaluation scores according to methods detailed in Appendix A.1. Dim. reduction refers to whether the clustering is performed on the codebook vectors of reduced dimensionality. Name of some clustering algorithms shortened for readability: `Agglomerative` stands for `AgglomerativeClustering`, and `Spectral` for `SpectralClustering`.

[5] We choose d_r through trial and error. d_r for each environment see Appendix A.1.

Relevant Positions

After determining which clustering of the codebook vector is most suitable, we construct, based on the clustering results, histograms of cluster change counts at each position of the discrete latent representations within subsets (See Appendix A.2.1), as well as for their sum over all subsets (See Appendix A.2.2).

Table 5.5 summarizes the positions in the discrete representations with above average number of cluster change counts for each selected object attribute, according to the histograms in Appendix A.2.

Game	Object Attribute	Positions with Frequent Cluster Change
`Breakout`	`ball_position`	$\mathbf{z}_{12}, \mathbf{z}_{13}, \mathbf{z}_{14}, \mathbf{z}_{15}$
`ChopperCommand`	`player_position`	$\mathbf{z}_{1}, \mathbf{z}_{2}, \mathbf{z}_{3}, \mathbf{z}_{4}, \mathbf{z}_{5}$
`DemonAttack`	`player_position`	$\mathbf{z}_{1}, \mathbf{z}_{4}, \mathbf{z}_{5}, \mathbf{z}_{6}, \mathbf{z}_{7}, \mathbf{z}_{11}, \mathbf{z}_{12}$
`Pong`	`player_position`	$\mathbf{z}_{3}, \mathbf{z}_{7}, \mathbf{z}_{11}, \mathbf{z}_{15}$
`Pong`	`enemy_position`	$\mathbf{z}_{4}, \mathbf{z}_{5}, \mathbf{z}_{8}, \mathbf{z}_{9}, \mathbf{z}_{12}, \mathbf{z}_{13}$

Table 5.5: Positions $i \in \{0, \dots, 15\}$ in discrete representations $\mathbf{z}$ with above average counts of cluster change, comparisons done within individual subsets, as detailed in Section 4.2.3

We encountered difficulty in finding the suitable clustering algorithm for `DemonAttack`. We recall that the "used" codebook vectors in this game are particularly evenly distributed, as shown in Figure 4.3. Therefore, we highlight our uncertainty regarding the results we derived (see Figures A.3 and A.8) for this attribute.

Discussion

We attempt to group codebook vectors into clusters in such a way that those with different effects on the reconstructed image are separated. And we assume that by doing so those vectors with similar effects are grouped together. Under this assumption, we interpret counts of cluster change at each position in the discrete representation as a measure of the relevance of that position to the object attribute of interest.

However, we did not account for the fact that the image observation used as input to the VQ-VAE encoder are resized into a resolution of 64×64. As a result, for example, the `Ball` object in `Breakout` is no longer a rectangle of 2×4 pixels, each with the same `RGB` value, but rather has a less regular and fuzzier shape with pixels of different `RGB` values that differ from `Observation` to `Observation`. We do not know how much this affects the abstraction of discrete representations $\mathbf{z}$ from image observations $\mathbf{x}$, but as we observe when interacting with the "directed" perturbation visualization, there exist a proportion of codebook vectors at certain positions whose effect on the reconstructed image has to do with minor changes in the *appearance* of the target object—in our example, in the shape and `RGB` values of the `Ball`

object. And the problem lies in the fact that these slight changes in object appearance do not always correspond to a change in the value of the object attribute `ball_position`.

We observe that such minor details accounts for a not-so-insignificant proportion of changes in the reconstructed image. This, together with our assumption about changes in reconstruction being captured as changes between clusters, means that at least a portion of the cluster change counts at certain positions in the discrete representation do not reflect an actual change in the value of o^j_{attr}. This constitutes a major limitation of this method. It makes us question to what extent the cluster change counts at each position indeed reflect the relevance of that position to the value of o^j_{attr}. It also raises the question of whether there exists another way to measure the similarity between codebook vectors other than clustering.

We also note other limitations of this method. For example, we did not determine the number of clusters C in a principled and systematic way, but rather through trial and error. The same holds for the choice of the reduced dimensionality d_r for `SpectralClustering`. Furthermore, it was not possible to evaluate the clustering results in an entirely quantitative way (See Appendix A.1.).

5.3.2 Results According to Permutation Importance

In this subsection, we describe the results obtained using the method described in Section 4.3. For each selected object attribute, we fit a `RandomForestRegressor` from the `scikit-learn` library [41] using each of the individual subsets, the union of all the subsets, as well as the entire dataset, as detailed in Section 4.3. Each set of data is split into a training set and a test set with `test_size=0.2`.

Permutation importance of each position in the discrete representation as a feature for the regressor are depicted in bar charts (Appendix A.3.1 shows the permutation importance for `RandomForestRegressors` trained on a single subset, Appendix A.3.2 for `RandomForestRegressors` trained on the union of all subsets, and Appendix A.3.3 for `RandomForestRegressors` trained on the entire dataset).

For `RandomForestRegressors` trained on the entire dataset and on the union of subsets, Table 5.6 provides an overview of the positions in the discrete representations with high permutation importance for predicting a given object attribute.

Discussion

Permutation importance of a feature is given by the decrease in the predictor's performance when the values of that feature are randomly permuted. A high permutation importance of a position in the discrete representation implies that the position is important for predicting the object attribute. However, to determine with certainty which minimal set of positions are

Game	**Object Attribute**	**Pos.** (Entire Dataset)	**Pos.** (Subset Union)
`Breakout`	`ball_position`	$\mathbf{z}_8$	$\mathbf{z}_8$
`ChopperCommand`	`player_position`	$\mathbf{z}_0$	$\mathbf{z}_9$
`DemonAttack`	`player_position`	$\mathbf{z}_{15}$	$\mathbf{z}_{13}$
`Pong`	`player_position`	$\mathbf{z}_3, \mathbf{z}_7$	$\mathbf{z}_1$
`Pong`	`enemy_position`	$\mathbf{z}_{12}$	$\mathbf{z}_8$

Table 5.6: Positions $i \in \{0, \dots, 15\}$ in $\mathbf{z}$ in the discrete representation with high permutation importance for `RandomForestRegressors` trained to predict values of an object attribute, for details see Section 4.3. The `RandomForestRegressors` used for these results are trained and evaluated on the entire dataset or the union of subsets defined above.

sufficient to predict the object attribute, we need further experimentation to train predictors on subsets of positions in the discrete representation.

We also note that the positions with high permutation importance for the regressors trained on the entire dataset and on the union of subsets differ from each other. This is likely because the object attributes to be predicted are not sufficiently diverse within the subsets, and the regressors trained on the union of subsets are not able to generalize well to the entire dataset.

5.3.3 Comparison of Results

The results obtained from the two approaches detailed in Section 4.2 and Section 4.3 diverge significantly from each other. For example, the positions in the discrete representations that are deemed relevant for predicting the object attributes `ball_position` in `Breakout` are $\mathbf{z}_{12}, \mathbf{z}_{13}, \mathbf{z}_{14}$, and $\mathbf{z}_{15}$ according to the clustering approach and $\mathbf{z}_8$ based on the permutation importance approach. Similar discrepancies persists for other object attributes in other games, with a slight exception for `player_position` and `enemy_position` in `Pong`, where the positions identified by the second approach form a subset of the positions identified by the first approach.

We think a reason for this misalignment is that the two methods approach the problem of determining the relevance of positions in the discrete representation to object attributes from different perspectives.

The first approach is not as straight forward as the second, with several assumptions and intermediate steps that might render the process less robust and more prone to error than intended. For example, we still do not have a quantitative and rigorous way to evaluate whether a clustering result aligns with our purpose of grouping codebook vectors with similar effects on the reconstructed image. In addition, the resizing of the input image observations to a lower resolution makes it more difficult to establish a correspondence between changes in reconstructed images and changes in the value of object attributes, introducing additional uncertainty to the results.

Nonetheless, interacting with the "directed" perturbation visualization provides us with a qualitative understanding of the compositional nature of the effects of individual codebook vectors on the reconstructed image, and the fact that these effects are usually localized but not necessarily confined to a single object in the image, about which we will further elaborate in Section 5.4. We also suspect that this is another reason for the misalignment of results from the two approaches.

The second approach, on the other hand, is more straightforward and easier to interpret. It is also better aligned with our eventual goal of extracting semantically meaningful information from the discrete representations, but the results still require further experimentation to be validated.

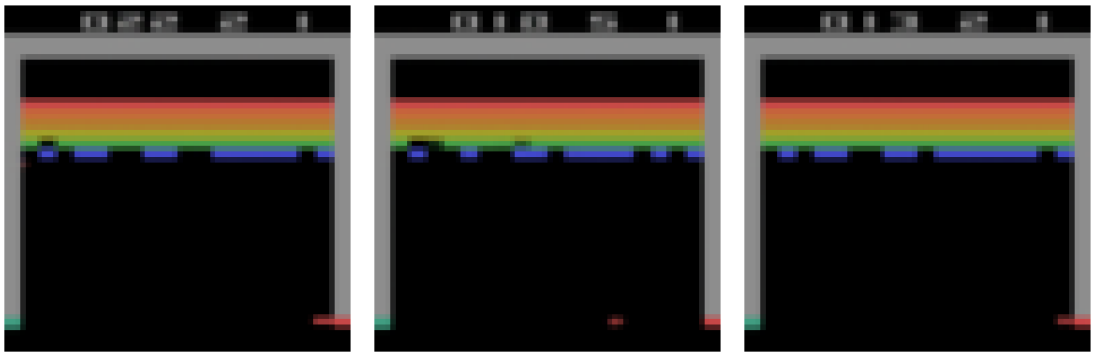
5.4 Effect of Individual Codebook Vectors

Comparing Table 5.5 and Table 5.6, the results from the two approaches described in Section 4.2 and Section 4.3 diverge with each other to such an extent that we cannot reach a definitive conclusion about whether certain positions i are particularly relevant for an object attribute o^j_{attr}.

In spite of this, we proceed to investigate the effect of codebook indices k^i at positions i that are identified to be important in predicting each of the selected object attributes, according to Table 5.6. We select k^i to be the tokens that are either most frequently or most rarely assigned to the position i in the discrete representations of `Observations` in a subset $S^{\mathbf{o}}_{\text{attr}}$, with the rationale that the former could encode some notion of common baseline value for the object attribute o^j_{attr}, while the latter could correspond to rarer signals. For details of how we calculate and visualize the effect of placing k^i at position i, see Section 4.4.

In the process, we observe that, even when we artificially construct a reference discrete representation $\mathbf{z}_{\text{ref}}$ using random codebook indices, its reconstruction $\hat{\mathbf{x}}_{\text{ref}}$ still appears to be a plausible image observation of the game. An example $\hat{\mathbf{x}}_{\text{ref}}$ is shown in Figure 5.2a, with $\mathbf{z}_{\text{ref}}$ consisting of the first 16 "used"[6] tokens. This is the case even when some or even all of the positions in $\mathbf{z}_{\text{ref}}$ have "unused" tokens k' as value, as shown in Figures 5.2b and 5.2c. Figure 5.2b shows the reconstruction of $\mathbf{z}_{\text{ref}}$ with the first 16 tokens as values, among which 12 are "used" and 4 are "unused". Figure 5.2c depicts the reconstruction of $\mathbf{z}_{\text{ref}}$ consisting of the first 16 "unused" tokens. This suggests that the distribution $p(\mathbf{x}|\mathbf{z})$ that the decoder has learned to approximate is primarily responsible for the majority of the objects depicted in the reconstructed images, not the individual or a combination of tokens at certain positions in the discrete representation.

[6] As we recall from Section 4.2.2, "unused" tokens are the indices of codebook vectors, which have never appeared in the discrete representations of any image observations in the collected data; and "used" tokens otherwise.



(a) Reconstruction of $\mathbf{z}_{\text{ref}}$ artificially constructed using the first 16 "used" tokens only.

(b) Reconstruction of $\mathbf{z}_{\text{ref}}$ artificially constructed using the first 16 tokens.

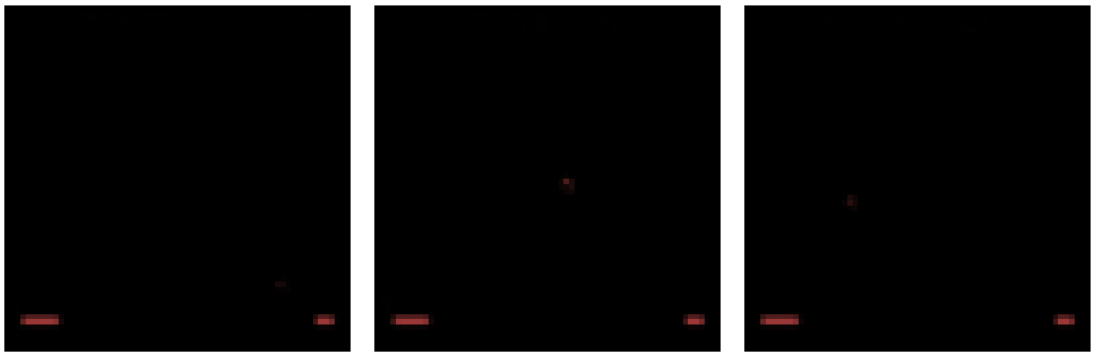
(c) Reconstruction of $\mathbf{z}_{\text{ref}}$ artificially constructed using the first 16 "unused" tokens.

Figure 5.2: Images reconstructed from artificially constructed discrete representation $\mathbf{z}$. Environment is `Breakout`.

Now, we compare the influence of the choice of reference discrete representation $\mathbf{z}_{\text{ref}}$ on the effect $\hat{\mathbf{x}}_{k,i}(\mathbf{z}_{\text{ref}})$ of individual token k^i at position i, which we defined as a function of $\mathbf{z}_{\text{ref}}$ in Section 4.4. When $\mathbf{z}_{\text{ref}}$ is the discrete representation of an `Observation` in $S^{\mathbf{o}}_{\text{attr}}$, the effect is visually distinct and localized, as shown in Figure 5.3a and Figure 5.3b; whereas when $\mathbf{z}_{\text{ref}}$ is artificially constructed from arbitrary tokens, the reconstructed effect is not as directly interpretable, as depicted in Figure 5.4.

Given the above observation, we opt for discrete representations from subsets $S^{\mathbf{o}}_{\text{ball\_pos}}$ as reference $\mathbf{z}_{\text{ref}}$. Comparing $\hat{\mathbf{x}}_{k,i}(\mathbf{z}_{\text{ref}})$ of the same token k^i at the same position i with different $\mathbf{z}_{\text{ref}}$, we observe that the effect of token k^i at position i is localized in small regions, i.e. objects, and one single token can simultaneously affect multiple such regions and objects. For example, in Figure 5.3a, the `Player` object appears at two different regions—one of which is the position where it used to be in, whereas the other is where it is currently located (both regions have roughly the same `RGB` value, since we calculate the effect as absolute difference)[7]. And in Figure 5.3b, token $k^i = 191$ at position $i = 8$ affects an additional region representing the `Ball` object. This single occurrence of change indicates that position $i = 8$ is not solely responsible for the position of the `Ball` object, since otherwise we would have seen two `Ball` objects in Figure 5.3b, just like we see two `Player` objects. We also note that the effect is rather visually distinct. Furthermore, the location (and form) of the changes varies as $\mathbf{z}_{\text{ref}}$ changes, indicating that the effect of a token at its given position is influenced by tokens at other positions. For instance, compare `ball_positions` in Figure 5.3b and Figure 5.3c. These observations are consistent with what we have seen when toggling the checkboxes in the visualization tool for 'directed' perturbation.

[7]We acknowledge that the effect is change in `player_position`, despite the fact that the subset is defined for `ball_posiion`. So, we additionally applied the permutation importance analysis for `player_position`, according to which position $i = 8$ is also the most important feature for predicting `player_position`.



(a) With $\mathbf{z}$ of an existing $\mathbf{x}$ from subset $S^{\mathbf{o}}_{\text{ball\_pos}}$ as reference.

(b) With $\mathbf{z}$ of an existing $\mathbf{x}$ from a different subset $S^{\mathbf{o}'}_{\text{ball\_pos}}$ as reference.

(c) With $\mathbf{z}$ of an existing $\mathbf{x}$ from a third subset $S^{\mathbf{o}''}_{\text{ball\_pos}}$ as reference.

Figure 5.3: Visualization of the effect of token $k^i = 191$ at position $i = 8$ in the discrete representation with different baseline discrete representations. Environment is `Breakout`. For details of the effect calculation see Section 4.4.

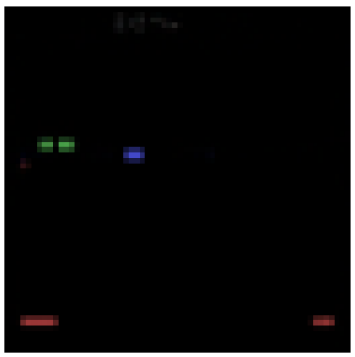


Figure 5.4: With a $\mathbf{z}$, artificially constructed using "used" tokens, as reference.

6 Conclusion

Learning symbolic planning representations from perceptual data constitutes a way to address the limitations of current deep RL methods in terms of generalization and sample efficiency, as well as the problem for symbolic planning methods of their reliance on hand-crafted domain models and problem descriptions.

6.0.1 Contributions and Conclusions

In this thesis, we presented approaches that explore the feasibility of learning symbolic representations from latent representations learned by a neural network. Specifically, we investigated the possibility of extracting semantic information from the discrete codebook space of the VQ-VAE of the model-based RL framework IRIS, which is used to train policies to play Atari games in a sample-efficient manner.

We presented two methods that determine the relevance of positions in the discrete representations, learned by the VQ-VAE model, to certain object attributes in selected Atari game environments. The first method centers around clustering the vectors in the VQ-VAE codebook. The results are evaluated with the help of the subsets we defined for each attribute and a visualization tool named "directed" perturbation. The cluster change counts at each position in the discrete representation are used as a proxy for the relevance of that position to the object attribute of interest. The second method uses a random forest regressor to predict each chosen object attribute using the discrete representation as categorical features. The relevance of each position in the discrete representation is measured by its permutation importance according to the random forest regressor.

We observe discrepancies in the two results and discuss the causes and implications of such misalignment. We acknowledge several limitations of the first method, such as the lack of a quantitative method for evaluating clustering results, and the unaccounted-for effect of resizing the input image on the validity of our basic assumptions for this method. Due to these limitations, results from the first method do not tell us much about our hypotheses. Nonetheless, the "directed" perturbation visualization tool advanced our understanding of the combined effect of tokens (codebook vector indices) on the output of the VQ-VAE decoder. On the other hand, the results of the second method are more straightforward to interpret and indicate the validity of our hypotheses, but further experimentation are required for a more definitive conclusion.

We introduced a way to visualize the effect of single tokens at different positions in the discrete representation on images reconstructed the VQ-VAE decoder. This provides us a qualitative understanding of the workings of the discrete representations: The combined effect of several tokens are required to represent a change in the value of a single object attribute. And, although the effect of a single token appears localized, it tends to affect more than one regions, thus more than one object attributes at the same time. Furthermore, we suspect it is the probability distribution $p(\mathbf{x}|\mathbf{z})$ approximated by the VQ-VAE decoder that is mainly responsible for the reconstruction of most of the object features depicted in the reconstructed image. And the individual tokens merely encode slight changes to the visual representations of the object attributes on top of this baseline reconstruction.

6.0.2 Future Work

One way to improve the first method we proposed, would be to do away with clustering the codebook vectors and instead use a distance metric such as the Euclidean distance between the codebook vectors to determine similarity. This would avoid situations where vectors at cluster boundaries are assigned to different clusters despite being close to each other. Instead of counting the number of cluster changes at each position in $\mathbf{z}$, we could keep track of the number of times the distance between $\mathbf{z}_1^i$ and $\mathbf{z}_2^i$ is above a certain threshold and infer the position's relevance accordingly.

For validation of the second method, we suggest training object attribute predictors using different subsets of the positions in $\mathbf{z}$ as features and comparing the performance of these predictors. For instance, a small increase in prediction error in the absence of position i as feature implies that position i is of little importance for determining the value of the object attribute. However, we note that this is still some distance away from actually learning symbolic representations from the discrete latent space.

For other future works, we find it meaningful to apply similar methods to object-recognition systems that employ slot attention [8], which are designed to learn disentangled representations of objects in images. Furthermore, we would advise against using a vanilla VQ-VAE to learn the latent representations in such a system, given our findings and what we deduced from our observations.

A Appendix

A.1 Semi-quantitative evaluation of clustering results

In this section, we detail the process used to evaluate the clustering results of the VQ-VAE codebook vectors, which enables the selection of the most suitable clustering algorithm for the approach described in Section 4.2.

For $\texttt{Observation}_1$ and $\texttt{Observation}_2$ from the same subset $S^{\mathbf{o}}_{\text{attr}}$, we evaluate the clustering result based on the following three aspects:

1. `major`: Whether the major changes in attribute o^j_{attr} between $\hat{\mathbf{x}}_1$ and $\hat{\mathbf{x}}_2$ are reflected by the cross-cluster changes between $\mathbf{z}_1$ and $\mathbf{z}_2$, i.e., in positions i where $\mathbf{z}^i_1 \neq \mathbf{z}^i_2$ and $c(\mathbf{z}^i_1) \neq c(\mathbf{z}^i_2)$.
2. `minor`: Whether the minor changes in attribute o^j_{attr} between $\hat{\mathbf{x}}_1$ and $\hat{\mathbf{x}}_2$ are also captured by the cross-cluster changes between $\mathbf{z}_1$ and $\mathbf{z}_2$.
3. `noise`: Whether any cross-cluster changes between $\mathbf{z}_1$ and $\mathbf{z}_2$ do not correspond to noticeable change in attribute o^j_{attr} between $\hat{\mathbf{x}}_1$ and $\hat{\mathbf{x}}_2$.

For each aspect, we assign a value y (yes) or n (no) based on our observation when interacting with the 'directed' perturbation, As a result, each `Observation` pair under the given clustering result is associated with a tuple $(v_{\texttt{major}}, v_{\texttt{minor}}, v_{\texttt{noise}})$. We assign a numerical value to each tuple as follows:

major	minor	noise	Value
y	y	y	3
y	y	n	4
y	n	y	1
y	n	n	2
n	*	*	0

Table A.1: Numerical values assigned to tuples $(v_{\texttt{major}}, v_{\texttt{minor}}, v_{\texttt{noise}})$ for evaluating clustering results. The symbol * stands for any value.

Given the size of $S^{\mathbf{o}}_{\text{attr}}$ and the manual nature of this evaluation process, we limit the assessment for each clustering result and each subset to comparisons between the first `Observation` and all other `Observations` in the subset. The values assigned to these pairs are averaged. The clustering algorithm that yields the highest average value is deemed the most suitable for our purposes.

A.1.1 Evaluation results

The following tables present the evaluation of the clustering results. Names of the algorithms are abbreviated as follows: `agglo` for `AgglomerativeClustering`, `hdbscan` for `HDBSCAN`, `gmm` for `GaussianMixture`, and `spectral + pca` for `SpectralClustering` with PCA dimensionality reduction. The algorithm with the highest average value (underlined in each table) is selected as the most suitable clustering algorithm for the given subset and attribute.

Algorithm	subset0	subset1	subset2	subset3	average
agglo	**3.46**	**3.68**	**3.23**	**3.52**	**3.47**
hdbscan	**3.46**	2.92	2.08	3.00	2.86
gmm	3.11	2.82	2.92	3.10	2.99
spectral + pca	2.97	2.29	2.38	2.43	2.52

Table A.2: Evaluation of clustering results across subsets for different algorithms for `ball_position` in `Breakout`. Bold values indicate the best performance. Dimension reduced to $d_r = 40$ for the spectral clustering algorithm.

Algorithm	subset0	subset1	subset2	subset3	average
agglo	**3.54**	**3.41**	3.05	3.12	3.28
hdbscan	2.98	3.02	3.24	2.87	3.03
gmm	3.19	3.24	2.99	3.32	3.19
spectral + pca	3.44	3.21	**3.38**	**3.47**	**3.38**

Table A.3: Evaluation of clustering results across subsets for different algorithms for `player_position` in `ChopperCommand`. Bold values indicate the best performance. Dimension reduced to $d_r = 40$ for the spectral clustering algorithm.

Algorithm	subset0	subset1	subset2	subset3	average
agglo	1.22	1.14	1.35	1.07	1.20
hdbscan	1.31	**1.21**	1.19	1.26	1.24
gmm	1.18	1.13	1.39	1.32	1.26
spectral + pca	**1.43**	1.20	**1.54**	**1.47**	**1.41**

Table A.4: Evaluation of clustering results across subsets for different algorithms for `player_position` in `DemonAttack`. Bold values indicate the best performance. Dimension reduced to $d_r = 40$ for the spectral clustering algorithm.

Algorithm	subset0	subset1	subset2	subset3	average
agglo	3.57	3.51	3.24	3.38	3.42
hdbscan	3.17	**3.59**	3.24	**3.63**	3.41
gmm	**3.74**	3.54	**3.38**	3.50	**3.54**
spectral + pca	3.26	3.41	3.14	2.13	2.98

Table A.5: Evaluation of clustering results across subsets for different algorithms for `player_position` in Pong. Bold values indicate the best performance. Dimension reduced to $d_r = 32$ for the spectral clustering algorithm.

Algorithm	subset0	subset1	subset2	subset3	average
agglo	3.65	3.51	3.19	3.38	3.43
hdbscan	3.13	**3.65**	3.24	3.50	3.38
gmm	**3.70**	3.54	**3.38**	**3.63**	**3.56**
spectral + pca	3.35	3.41	3.10	2.13	2.99

Table A.6: Evaluation of clustering results across subsets for different algorithms for `enemy_position` in Pong. Bold values indicate the best performance. Dimension reduced to $d_r = 32$ for the spectral clustering algorithm.

A.2 Results According to Clustering and 'Directed' Perturbation Within Subsets

A.2.1 Individual Subsets

Figure A.1: Histogram for `ball_position` in `Breakout` of cluster change counts at each position in discrete representations of two `Observations` from the same subset, counted within each individual subsets.

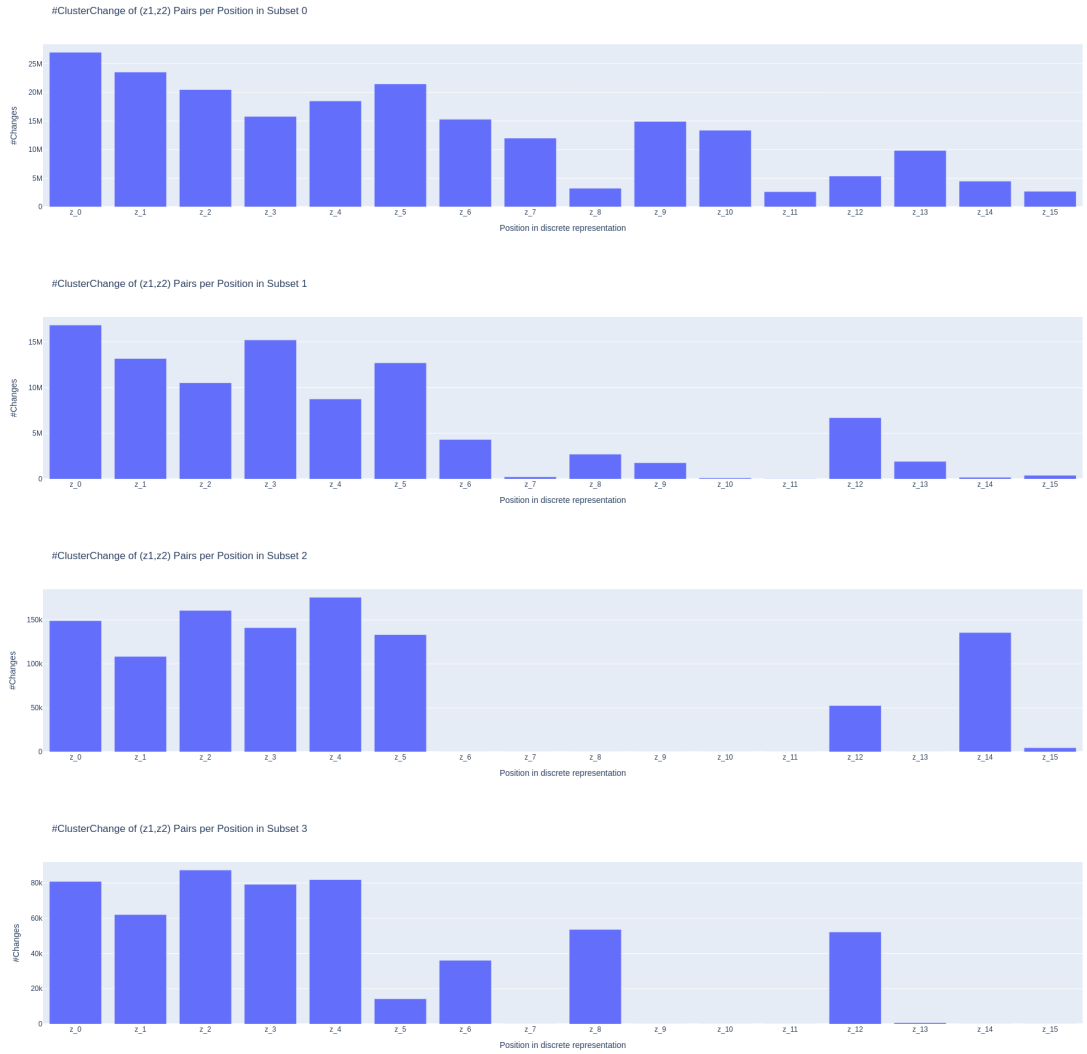


Figure A.2: Histogram for `player_position` in `ChopperCommand` of cluster change counts at each position in discrete representations of two `Observations` from the same subset, counted within each individual subsets.

Figure A.3: Histogram for `player_position` in `DemonAttack` of cluster change counts at each position in discrete representations of two `Observations` from the same subset, counted within each individual subsets.

Figure A.4: Histogram for `player_position` in `Pong` of cluster change counts at each position in discrete representations of two `Observations` from the same subset, counted within each individual subsets.

Figure A.5: Histogram for `enemy_position` in `Pong` of cluster change counts at each position in discrete representations of two `Observations` from the same subset, counted within each individual subsets.

A.2.2 Sum over All Available Subsets

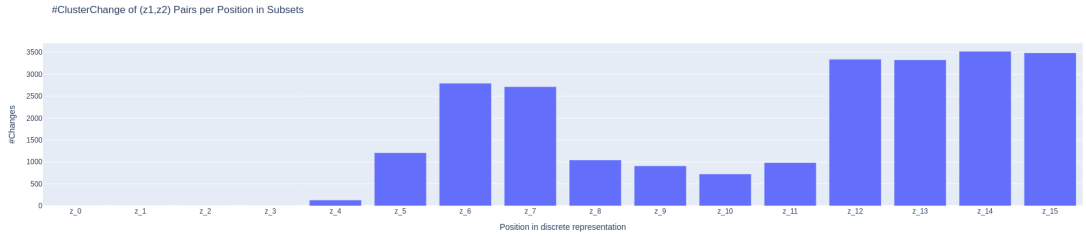


Figure A.6: Histogram for `ball_position` in `Breakout` of cluster change counts at each position in discrete representation of two `Observations` from the same subset, counts summed up over all available subsets.

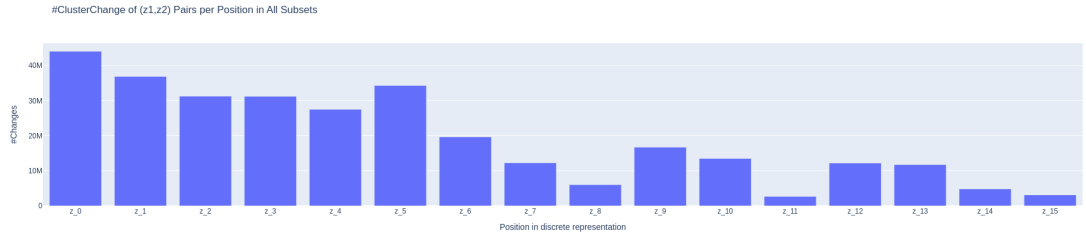


Figure A.7: Histogram for `player_position` in `ChopperCommand` of cluster change counts at each position in discrete representation of two `Observations` from the same subset, counts summed up over all available subsets.

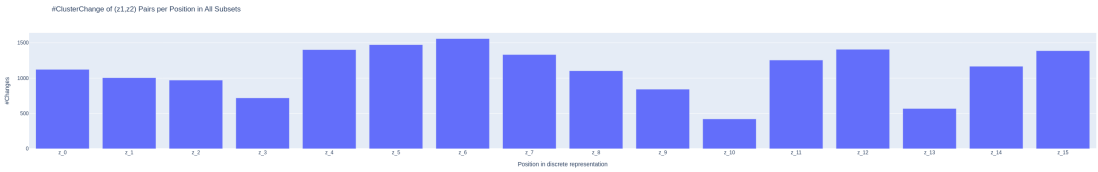


Figure A.8: Histogram for `player_position` in `DemonAttack` of cluster change counts at each position in discrete representation of two `Observations` from the same subset, counts summed up over all available subsets.

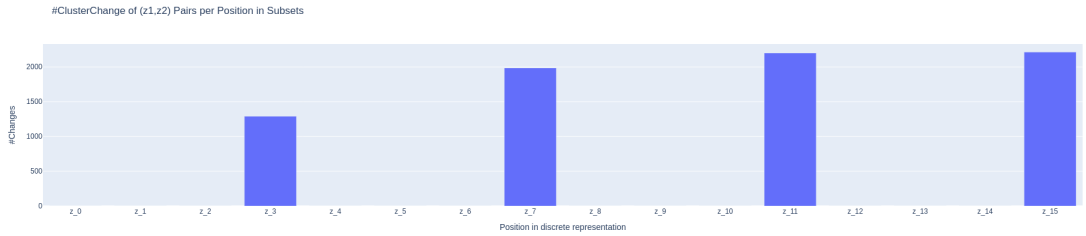


Figure A.9: Histogram for `player_position` in `Pong` of cluster change counts at each position in discrete representation of two `Observations` from the same subset, counts summed up over all available subsets.

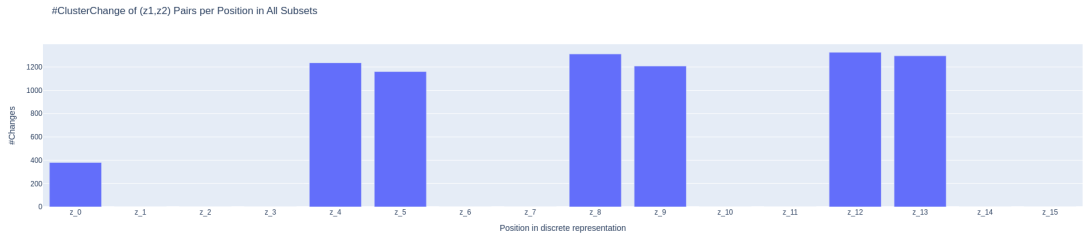


Figure A.10: Histogram for `enemy_position` in `Pong` of cluster change counts at each position in discrete representation of two `Observations` from the same subset, counts summed up over all available subsets.

A.3 Results According to Permutation Importance

A.3.1 Regressors Trained on Individual Subsets

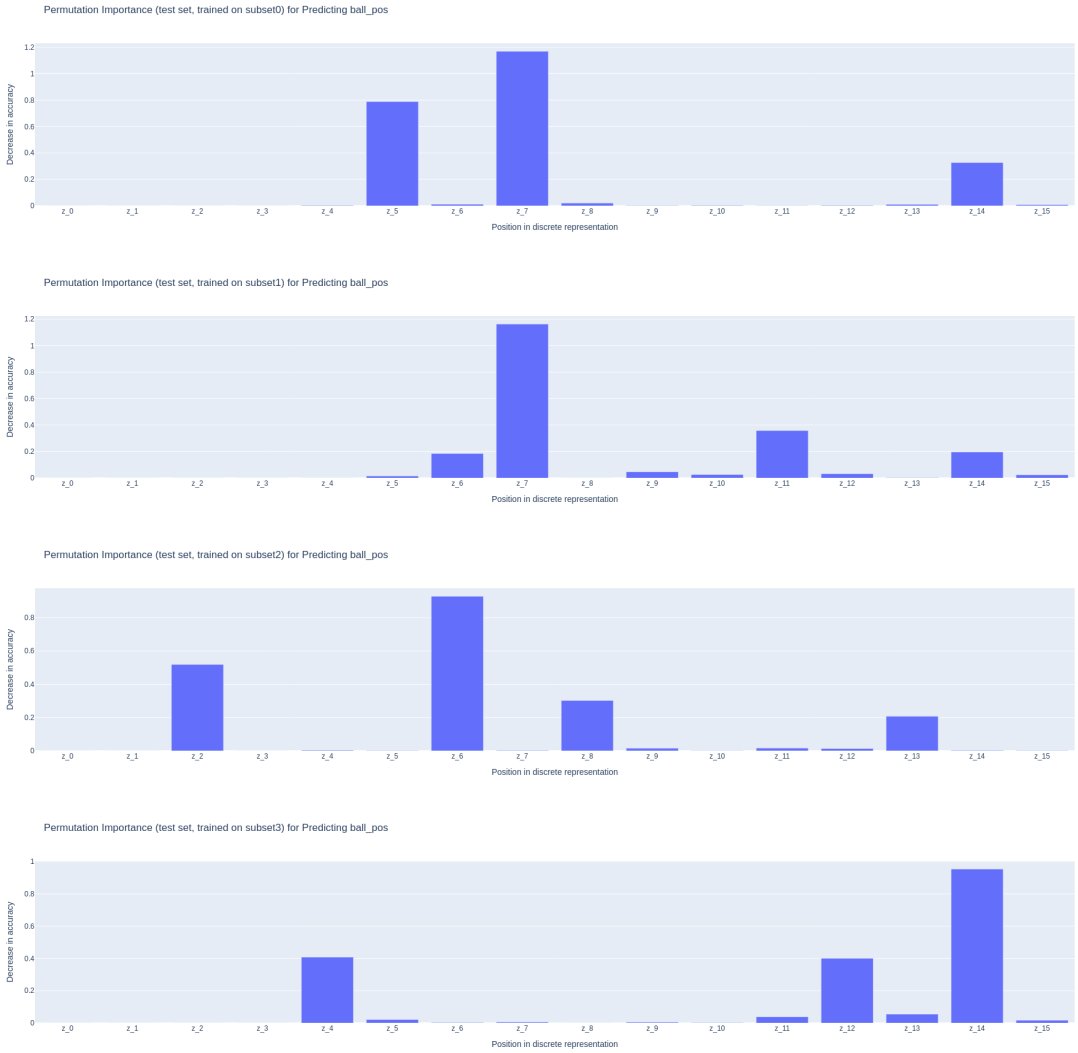


Figure A.11: Bar chart for `ball_position` in `Breakout` of permutation importance of each position in discrete representation, computed within each individual subsets.

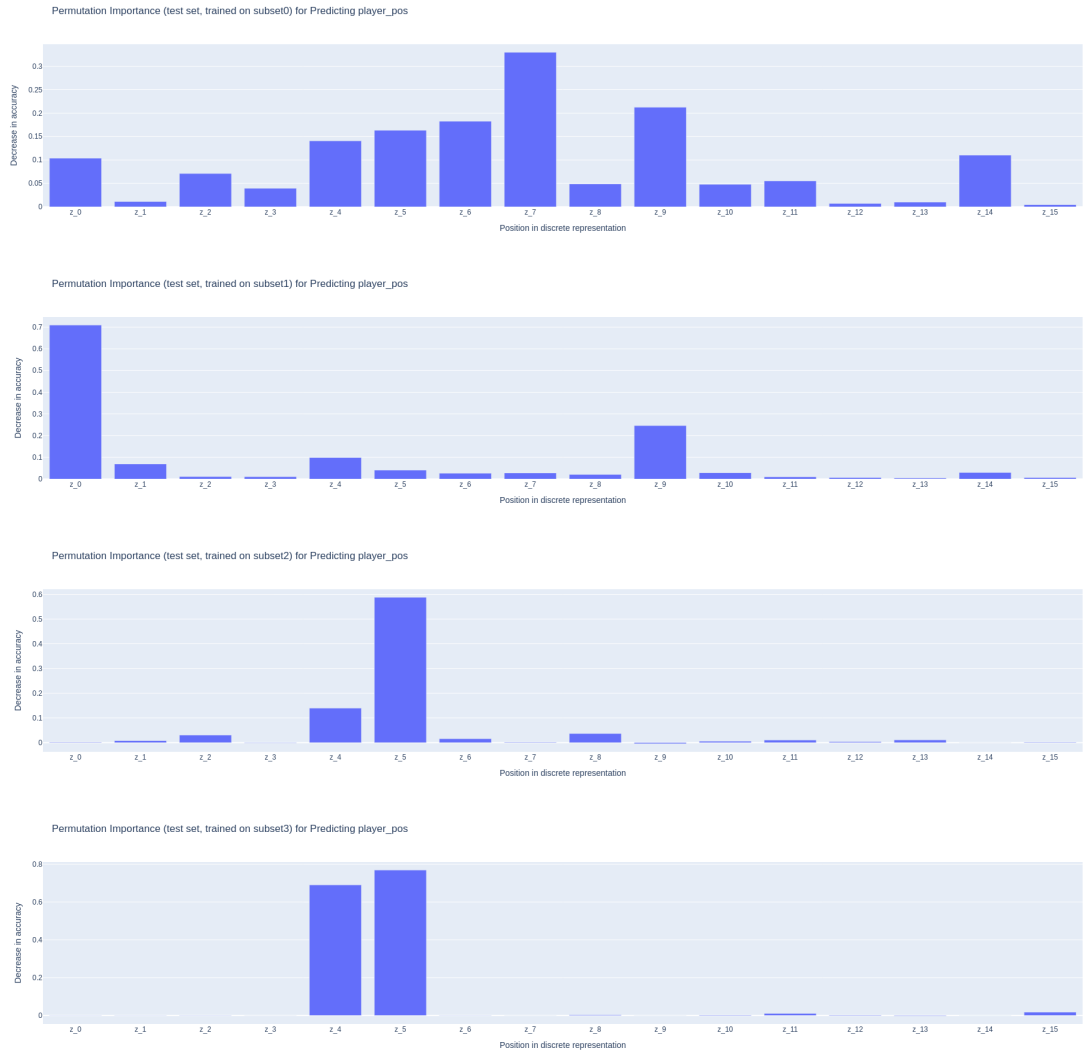


Figure A.12: Bar chart for `player_position` in `ChopperCommand` of permutation importance of each position in discrete representation, computed within each individual subsets.

Figure A.13: Bar chart for `player_position` in `DemonAttack` of permutation importance of each position in discrete representation, computed within each individual subsets.

Figure A.14: Bar chart for `player_position` in Pong of permutation importance of each position in discrete representation, computed within each individual subsets.

Figure A.15: Bar chart for `enemy_position` in `Pong` of permutation importance of each position in discrete representation, computed within each individual subsets.

A.3.2 Regressors Trained on Union of All Available Subsets

computed

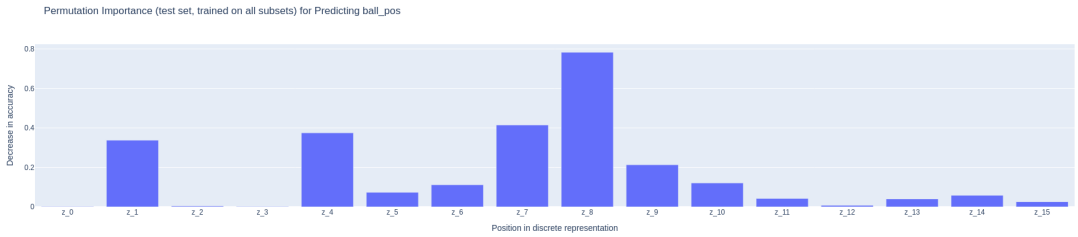


Figure A.16: Bar chart for `ball_position` in `Breakout` of permutation importance of each position in discrete representation, computed over union of all available subsets.

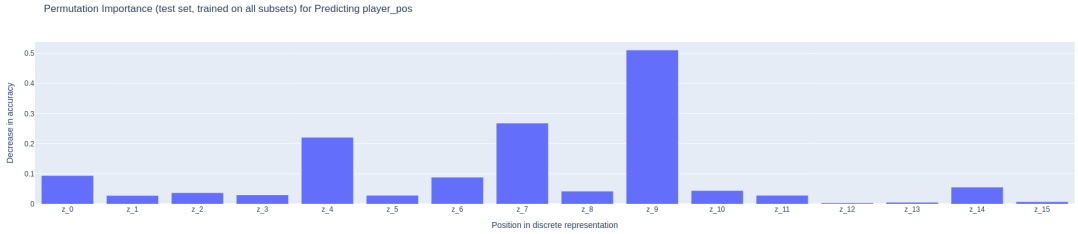


Figure A.17: Bar chart for `player_position` in `ChopperCommand` of permutation importance of each position in discrete representation, computed over union of all available subsets.

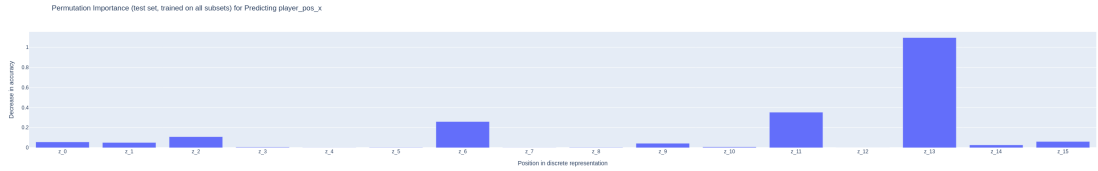


Figure A.18: Bar chart for `player_position` in `DemonAttack` of permutation importance of each position in discrete representation, computed over union of all available subsets.

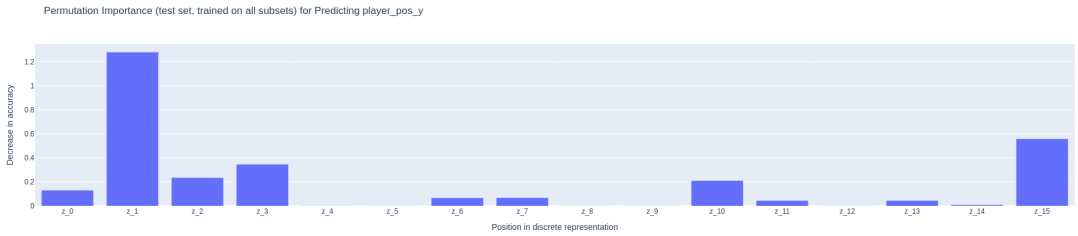


Figure A.19: Bar chart for `player_position` in `Pong` of permutation importance of each position in discrete representation, computed over union of all available subsets.

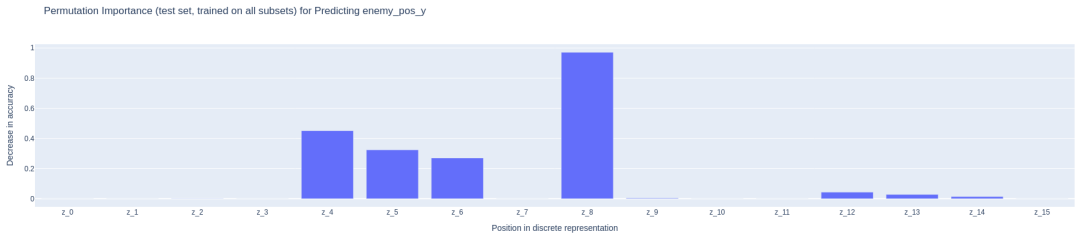


Figure A.20: Bar chart for `enemy_position` in `Pong` of permutation importance of each position in discrete representation, computed over union of all available subsets.

A.3.3 Regressors Trained on Entire Dataset

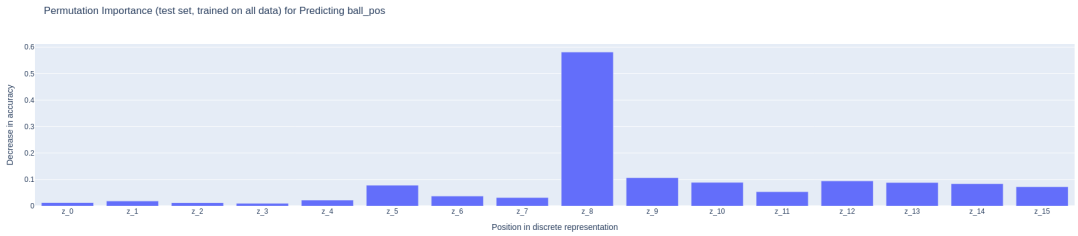


Figure A.21: Bar chart for `ball_position` in `Breakout` of permutation importance of each position in discrete representation, computed over the entire dataset.

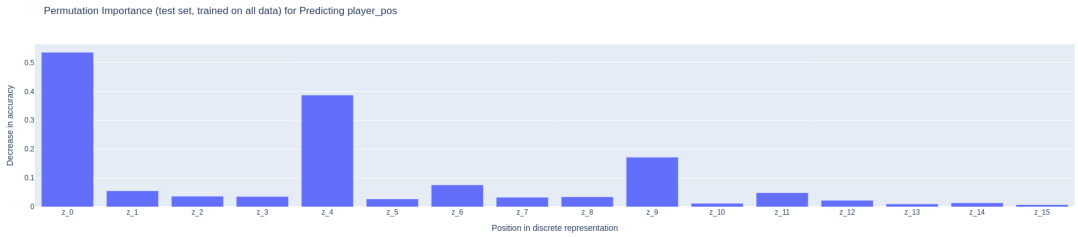


Figure A.22: Bar chart for `player_position` in `ChopperCommand` of permutation importance of each position in discrete representation, computed over the entire dataset.

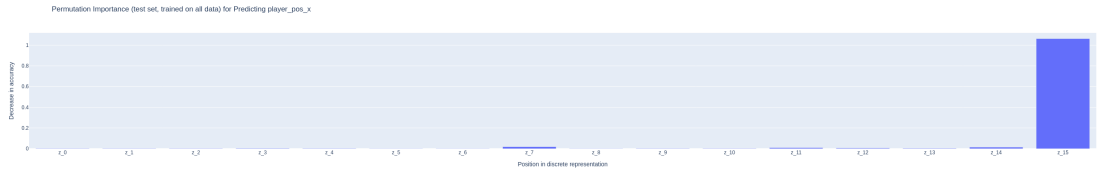


Figure A.23: Bar chart for `player_position` in `DemonAttack` of permutation importance of each position in discrete representation, computed over the entire dataset.

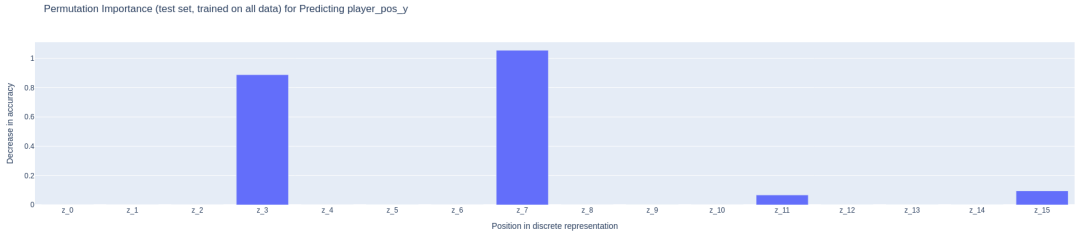


Figure A.24: Bar chart for `player_position` in `Pong` of permutation importance of each position in discrete representation, computed over the entire dataset.

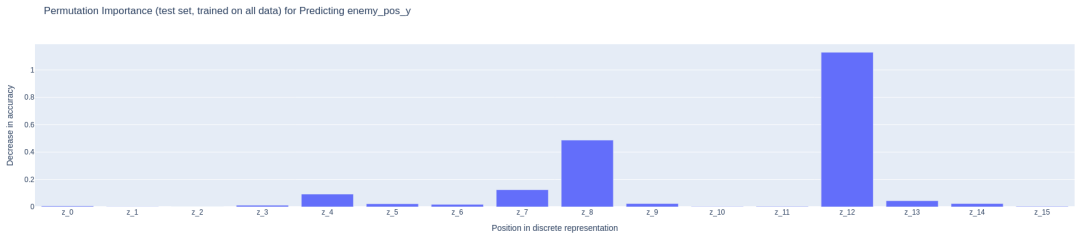


Figure A.25: Bar chart for `enemy_position` in `Pong` of permutation importance of each position in discrete representation, computed over the entire dataset.

List of Acronyms

ALE	Arcade Learning Environment
CNN	Convolutional Neural Network
DQN	Deep Q-Network
FOL	First-Order Logic
FOSAE	First-Order State Autoencoder
IRIS	Imagination with auto-Regression over an Inner Speech
LatPlan	Latent-space Planner
LSTM	Long short-term memory
MDP	Markov Decision Process
O2D	Objects in 2D space
OCAtari	Object-Centric Atari
PCA	principal component analysis
PDDL	Planning Domain Definition Language
POMDP	Partially Observable Markov Decision Process
PU	Predicate Unit
REM	RAM Extraction Method
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RSSM	Recurrent State-Space Model
RV	Random Variable
t-SNE	t-distributed Stochastic Neighbor Embedding
VAE	Variational Autoencoder
VEM	Visual Extraction Method
VQ-VAE	Vector-Quantized Variational Autoencoder

List of Symbols

General

$(\cdot)$	arbitrary (scalar) variable
$(\bullet)$	arbitrary vector variable
$(\cdot, \cdot, \cdot)$	tuple of multiple variables
$\mathbb{R}$	set of real numbers

Planning

P	planning problem
D_{FO}	first-order domain of a planning problem
I	instance information of a planning problem
$Init$	initial state of a planning problem
$Goal$	goal state of a planning problem

Model-Based Reinforcement Learning

env	environment
env'	world model, or simulated environment
π	policy
π^*	optimal policy

Markov Decision Process

t	time step index
$\mathcal{S}$	set of all states
$\mathcal{A}$	set of all actions
$\mathcal{R}$	set of all possible rewards
s, s_t	state at time t, realization of random variable (RV) S^t
s', s_{t+1}	next state at time $t+1$, realization of RV S^{t+1}
a, a_t	action at time t, realization of RV A^t
a', a_{t+1}	next action at time $t+1$, realization of RV A^{t+1}
r, r_{t+1}	reward at time $t+1$, realization of RV R^{t+1}

Vector-Quantized Variational Autoencoder

E	encoder
D	decoder
n_z	dimension of codebook vectors

K	number of codebook vectors
K'	number of used codebook vectors
e_k	codebook vector with index k
$\mathcal{E}$	codebook
$\mathcal{E}'$	codebook without unused codebook vectors
$\mathbf{x}$	encoder input
$\hat{\mathbf{y}}$	encoder output
$\mathbf{q}(\bullet)$	element-wise quantization function
$\mathbf{y_q}$	result of quantizing encoder output $\hat{\mathbf{y}}$; decoder input
$\mathbf{index}(\bullet)$	element-wise function that maps each codebook vector e_k in $\mathbf{y_q}$ to its index k
$\mathbf{flatten}(\bullet)$	function that maps a $h \times w$ matrix to a $h \cdot w$-dimensional vector
$\hat{\mathbf{x}}$	decoder output

Imagination with auto-Regression over an Inner Speech (IRIS)

$\mathbf{x}$, $\mathbf{x}_t$	image observation at time t
$\hat{\mathbf{x}}$, $\hat{\mathbf{x}}_t$	reconstruction of image observation at time t
$\mathbf{z}$, $\mathbf{z}_t$	discrete latent representation of an image observation at time t
$\mathbf{z}^i$, $\mathbf{z}_t^i$	token at position i in the discrete representation of an image observation at time t
$\mathbf{o}$, $\mathbf{o}_t$	object-centric representation of an image observation at time t
$\mathbf{o}^j$, $\mathbf{o}_t^j$	j-th object in the object-centric representation of an image observation at time t
$\mathbf{o}_{\text{attr}}^j$, $\mathbf{o}_{t,\text{attr}}^j$	attribute of the j-th object in the object-centric representation of an image observation at time t
$S_{\text{attr}}^{\mathbf{o}}$	multiset of `Observations` defined by object attribute o_{attr}^j and representative `Observation` $(\mathbf{x}, \mathbf{z}, \mathbf{o})$
$\mathcal{F}$	the set of object attributes for the definition of a subset of `Observations`
$\mathbf{set}(\cdot)$	operator that maps a multiset to a set
C	number of clusters
c_l	cluster with index l
$\mathbf{cluster}(\cdot)$	operator that maps a token to its assigned cluster
$n^i_{o_{\text{attr}}^j,\mathbf{o}}$	number of cross-cluster changes for each subset $S_{\text{attr}}^{\mathbf{o}}$ and each position i in the discrete representation

List of Figures

List of Tables

List of Algorithms

List of References

[1] O. Ahmed, F. Träuble, A. Goyal, A. Neitz, Y. Bengio, B. Schölkopf, M. Wüthrich, and S. Bauer, *Causalworld: A robotic manipulation benchmark for causal structure and transfer learning*, 2020. arXiv: 2010.04296 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2010.04296.

[2] E. Alonso, A. Jelley, V. Micheli, A. Kanervisto, A. Storkey, T. Pearce, and F. Fleuret, *Diffusion for world modeling: Visual details matter in atari*, 2024. arXiv: 2405.12399 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2405.12399.

[3] M. Asai, *Unsupervised grounding of plannable first-order logic representation from images*, 2019. arXiv: 1902.08093 [cs.AI]. [Online]. Available: https://arxiv.org/abs/1902.08093.

[4] M. Asai and A. Fukunaga, *Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary*, 2017. arXiv: 1705.00154 [cs.AI]. [Online]. Available: https://arxiv.org/abs/1705.00154.

[5] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, 2016. arXiv: 1409.0473 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1409.0473.

[6] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, Jun. 2013, ISSN: 1076-9757. DOI: 10.1613/jair.3912. [Online]. Available: http://dx.doi.org/10.1613/jair.3912.

[7] Y. Bengio, N. Léonard, and A. Courville, *Estimating or propagating gradients through stochastic neurons for conditional computation*, 2013. arXiv: 1308.3432 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1308.3432.

[8] O. Biza, S. van Steenkiste, M. S. M. Sajjadi, G. F. Elsayed, A. Mahendran, and T. Kipf, *Invariant slot attention: Object discovery with slot-centric reference frames*, 2023. arXiv: 2302.04973 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2302.04973.

[9] R. J. G. B. Campello, D. Moulavi, and J. Sander, "Density-based clustering based on hierarchical density estimates," in *Advances in Knowledge Discovery and Data Mining*, J. Pei, V. S. Tseng, L. Cao, H. Motoda, and G. Xu, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 160–172, ISBN: 978-3-642-37456-2.

[10] X. Chang, P. Ren, P. Xu, Z. Li, X. Chen, and A. Hauptmann, "A comprehensive survey of scene graphs: Generation and application," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 1, pp. 1–26, 2021.

[11] Q. Delfosse, J. Blüml, B. Gregori, S. Sztwiertnia, and K. Kersting, *Ocatari: Object-centric atari 2600 reinforcement learning environments*, 2024. arXiv: 2306.08649 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2306.08649.

[12] A. Didolkar, A. Goyal, and Y. Bengio, *Cycle consistency driven object discovery*, 2023. arXiv: 2306.02204 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2306.02204.

[13] P. Esser, R. Rombach, and B. Ommer, *Taming transformers for high-resolution image synthesis*, 2021. arXiv: 2012.09841 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2012.09841.

[14] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3, pp. 189–208, 1971, ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(71)90010-5. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0004370271900105.

[15] D. J. Foster, "Replay comes of age.," *Annual review of neuroscience*, vol. 40, pp. 581–602, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:46885097.

[16] A. d'Avila Garcez, A. R. R. Dutra, and E. Alonso, *Towards symbolic reinforcement learning with common sense*, 2018. arXiv: 1804.08597 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1804.08597.

[17] M. Garnelo, K. Arulkumaran, and M. Shanahan, *Towards deep symbolic reinforcement learning*, 2016. arXiv: 1609.05518 [cs.AI]. [Online]. Available: https://arxiv.org/abs/1609.05518.

[18] H. Geffner, *Model-free, model-based, and general intelligence*, 2018. arXiv: 1806.02308 [cs.AI]. [Online]. Available: https://arxiv.org/abs/1806.02308.

[19] H. Geffner and B. Bonet, *A Concise Introduction to Models and Methods for Automated Planning: Synthesis Lectures on Artificial Intelligence and Machine Learning*, 1st. Morgan & Claypool Publishers, 2013, ISBN: 1608459691.

[20] D. Ha and J. Schmidhuber, *Recurrent world models facilitate policy evolution*, 2018. arXiv: 1809.01999 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1809.01999.

[21] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, *Dream to control: Learning behaviors by latent imagination*, 2020. arXiv: 1912.01603 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1912.01603.

[22] D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba, *Mastering atari with discrete world models*, 2022. arXiv: 2010.02193 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2010.02193.

[23] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap, *Mastering diverse domains through world models*, 2024. arXiv: 2301.04104 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2301.04104.

[24] S. Harnad, "The symbol grounding problem," *Physica D: Nonlinear Phenomena*, vol. 42, no. 1–3, pp. 335–346, Jun. 1990, ISSN: 0167-2789. DOI: 10.1016/0167-2789(90)90087-6. [Online]. Available: http://dx.doi.org/10.1016/0167-2789(90)90087-6.

[25] P. Haslum, N. Lipovetzky, D. Magazzeni, C. Muise, R. Brachman, F. Rossi, and P. Stone, *An Introduction to the Planning Domain Definition Language* (Synthesis Lectures on Artificial Intelligence and Machine Learning). Morgan & Claypool Publishers, 2019, ISBN: 9781627057370. [Online]. Available: https://books.google.de/books?id=bA6QDwAAQBAJ.

[26] E. Jang, S. Gu, and B. Poole, *Categorical reparameterization with gumbel-softmax*, 2017. arXiv: 1611.01144 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1611.01144.

[27] J. Johnson, A. Alahi, and L. Fei-Fei, *Perceptual losses for real-time style transfer and super-resolution*, 2016. arXiv: 1603.08155 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1603.08155.

[28] J. Johnson, R. Krishna, M. Stark, L.-J. Li, D. Shamma, M. Bernstein, and L. Fei-Fei, "Image retrieval using scene graphs," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3668–3678.

[29] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, A. Mohiuddin, R. Sepassi, G. Tucker, and H. Michalewski, *Model-based reinforcement learning for atari*, 2024. arXiv: 1903.00374 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1903.00374.

[30] A. Karpathy, *Mingpt: A minimal pytorch re-implementation of the openai gpt (generative pretrained transformer) training*, https://github.com/karpathy/minGPT, 2020.

[31] A. Kori, F. Locatello, F. D. S. Ribeiro, F. Toni, and B. Glocker, *Grounded object centric learning*, 2024. arXiv: 2307.09437 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2307.09437.

[32] H. Li, G. Zhu, L. Zhang, Y. Jiang, Y. Dang, H. Hou, P. Shen, X. Zhao, S. A. A. Shah, and M. Bennamoun, "Scene graph generation: A comprehensive survey," *Neurocomputing*, vol. 566, p. 127 052, 2024.

[33] A. O. Liberman, B. Bonet, and H. Geffner, *Learning first-order symbolic planning representations that are grounded*, 2022. arXiv: 2204.11902 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2204.11902.

[34] F. Locatello, D. Weissenborn, T. Unterthiner, A. Mahendran, G. Heigold, J. Uszkoreit, A. Dosovitskiy, and T. Kipf, *Object-centric learning with slot attention*, 2020. arXiv: 2006.15055 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2006.15055.

[35] L. S. Lorello and M. Lippi, "The challenge of learning symbolic representations.," in *NeSy*, 2023, pp. 44–61.

[36] U. von Luxburg, *A tutorial on spectral clustering*, 2007. arXiv: 0711.0189 [cs.DS]. [Online]. Available: https://arxiv.org/abs/0711.0189.

[37] D. McDermott, M. Ghallab, A. E. Howe, C. A. Knoblock, A. Ram, M. M. Veloso, D. S. Weld, and D. E. Wilkins, "Pddl-the planning domain definition language," 1998. [Online]. Available: https://api.semanticscholar.org/CorpusID:59656859.

[38] V. Micheli, E. Alonso, and F. Fleuret, *Transformers are sample-efficient world models*, 2023. arXiv: 2209.00588 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2209.00588.

[39] D. Müllner, *Modern hierarchical, agglomerative clustering algorithms*, 2011. arXiv: 1109.2378 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1109.2378.

[40] A. van den Oord, O. Vinyals, and K. Kavukcuoglu, *Neural discrete representation learning*, 2018. arXiv: 1711.00937 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1711.00937.

[41] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, A. Müller, J. Nothman, G. Louppe, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, *Scikit-learn: Machine learning in python*, 2018. arXiv: 1201.0490 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1201.0490.

[42] A. Razavi, A. van den Oord, and O. Vinyals, *Generating diverse high-fidelity images with vq-vae-2*, 2019. arXiv: 1906.00446 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1906.00446.

[43] S. Russell and P. Norvig, *Artificial Intelligence, Global Edition A Modern Approach*. Pearson Deutschland, 2021, p. 1168, ISBN: 9781292401133. [Online]. Available: https://elibrary.pearson.de/book/99.150005/9781292401171.

[44] J. Seo and J. Kang, *Raq-vae: Rate-adaptive vector-quantized variational autoencoder*, 2024. arXiv: 2405.14222 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2405.14222.

[45] G. Singh, F. Deng, and S. Ahn, *Illiterate dall-e learns to compose*, 2022. arXiv: 2110.11405 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2110.11405.

[46] S. Ståhlberg, B. Bonet, and H. Geffner, "Learning General Policies with Policy Gradient Methods," in *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning*, Aug. 2023, pp. 647–657. DOI: 10.24963/kr.2023/63. [Online]. Available: https://doi.org/10.24963/kr.2023/63.

[47] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, second edition. MIT Press, Nov. 13, 2018, ISBN: 978-0-262-03924-6.

[48] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, 2023. arXiv: 1706.03762 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1706.03762.

[49] D. Xu, Y. Zhu, C. B. Choy, and L. Fei-Fei, *Scene graph generation by iterative message passing*, 2017. arXiv: 1701.02426 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1701.02426.

[50] J. Yang, J. Lu, S. Lee, D. Batra, and D. Parikh, "Graph r-cnn for scene graph generation," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 670–685.

[51] P. Yin, J. Lyu, S. Zhang, S. Osher, Y. Qi, and J. Xin, *Understanding straight-through estimator in training activation quantized neural nets*, 2019. arXiv: 1903.05662 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1903.05662.

[52] R. Zellers, M. Yatskar, S. Thomson, and Y. Choi, *Neural motifs: Scene graph parsing with global context*, 2018. arXiv: 1711.06640 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1711.06640.