An Unsupervised Video Game Playstyle Metric via State Discretization (Supplementary Material)

Chiu-Chou Lin[*1] **Wei-Chen Chiu**[†1,2] **I-Chen Wu**[‡1,2,3]

[1]Department of Computer Science, National Yang Ming Chiao Tung University, Taiwan
[2]Pervasive Artificial Intelligence Research (PAIR) Labs, Taiwan
[3]Research Center for IT Innovation, Academia Sinica, Taiwan

Abstract

On playing video games, different players usually have their own playstyles. Recently, there have been great improvements for the video game AIs on the playing strength. However, past researches for analyzing the behaviors of players still used heuristic rules or the behavior features with the game-environment support, thus being exhausted for the developers to define the features of discriminating various playstyles. In this paper, we propose the first metric for video game playstyles directly from the game observations and actions, without any prior specification on the playstyle in the target game. Our proposed method is built upon a novel scheme of learning discrete representations that can map game observations into latent discrete states, such that playstyles can be exhibited from these discrete states. Namely, we measure the playstyle distance based on game observations aligned to the same states. We demonstrate high playstyle accuracy of our metric in experiments on some video game platforms, including TORCS, RGSK, and seven Atari games, and for different agents including rule-based AI bots, learning-based AI bots, and human players.

1 INTRODUCTION

Generally, players of a video game would demonstrate different playstyles [Tychsen and Canossa, 2008, Pirker et al., 2016] to enrich playing experiences. Take a racing game as an example: different players usually have their own preferences on the track positions or the strategies of using accelerations. Basically, the playstyle can be captured by the characteristic of the sets of playing behaviors, where the playing behavior is represented by the pair of game observation (i.e., screens) and its corresponding action taken by the player.

Although it is known that the player modelling has been a common domain in studying the behaviors of players [Yannakakis et al., 2013], the playstyle which we are interested here is able to deliver the intention or preferences of agents or players. If we can directly measure playstyles rather than just capturing some playing behaviors, it becomes useful to develop AI bots for following various playstyles, such as mimicking human players' styles. Therefore, how to measure playstyles becomes important with great potential to both the video game and the AI communities.

Recently, with the rapid growth of deep reinforcement learning (DRL) techniques, the trained AI bots not only achieve super-human performance in several complicated games, e.g., DotA2 [OpenAI, 2018] and StartCraft II [Vinyals et al., 2019], but also attempt to discover new playstyles during their training procedure [Baker et al., 2020]. However, it is hard to have proper metrics to measure the diversity among the playing polices of different AI bots, hence being difficult for game developers to define a specific playstyle or even enrich the playstyles in a new video game. To the best of our knowledge, the existing works for the player behaviors still mainly consider the heuristic rules or study behavior features based on the explicit game-environment support [Tychsen and Canossa, 2008, Pirker et al., 2016, Bontchev and Georgieva, 2018, Yannakakis et al., 2013, Drachen et al., 2012], where they are limited to some pre-defined playstyles. So, it is hard to support a general playstyle metric in the above way. It hence becomes more expected to have a general metric to measure playstyles without any prior knowledge about styles or game specifications.

The issue of designing a general metric for playstyles is actually nontrivial in the sense that different combinations of playing behaviors may lead to too many different game results to make it hard to identify playstyles. In the past,

[*]dsobscure@outlook.com
[†]walon@cs.nctu.edu.tw
[‡]icwu@cs.nctu.edu.tw

Supplement for the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence (UAI 2021).

researches related to playstyles were studied on some applications, such as the categorization of driving behaviors in the driver assistance systems [Brombacher et al., 2017, Shouno, 2018]. Some of these researches discovered different driving behaviors by using discrete representations of the states from sensor signals (derived by Hidden Markov Models (HMMs) [Takano et al., 2008]), and then did clustering for driving styles afterwards. Many works followed to utilize these representations to perform analysis of the time-series data of driving [Taniguchi et al., 2012, 2015, 2016]. As an extension of the analysis of driving behaviors, Shouno [2018] propose a metric for evaluating the similarity between driving styles based on the patterns derived from the t-distributed stochastic neighbor embedding (t-SNE) of image observations.

In addition to analyzing driving behaviors, another possible solution for evaluating the similarity between playstyles from observations is to estimate the distance between distributions of image observations (i.e., game screens) produced during the game playing, via the metric such as Fréchet Inception Distance [Heusel et al., 2017]. Nevertheless, the above solution does not take actions into account. Besides, some playstyles depend on random events in a game, which does not exist in common image distribution comparisons. Therefore, there exists a significant demand for the metric capable of directly predicting playstyles from the rich information composed of the observation-action pairs. This is an issue to be addressed in this paper.

In this paper, we propose a novel metric to predict the video game playstyles based on observations and actions of game episodes without any prior knowledge about style specifications. Our proposed method, inspired by the above driving styles, is built upon a novel scheme of learning discrete representations, called hierarchical state discretization (HSD) in this paper, which can map game observations into latent discrete states, such that playstyles can be exhibited from these discrete states. Discrete representation is reviewed in Section 2, and HSD is described in Section 4. Namely, we measure the playstyle distance based on game observations aligned to the same states as presented in Section 3. Then, we predict playstyles based on the playstyle distance. In our experiments described in Section 5, we demonstrate high playstyle accuracy of our proposed metric on three video game platforms, including TORCS [Wymann et al., 2014], RGSK[1], and seven Atari games [Bellemare et al., 2013], and for different agents including rule-based AI bots, learning-based AI bots, and human players. Finally, we make concluding remarks in Section 6.

[1] https://assetstore.unity.com/packages/templates/racing-game-starter-kit-22615

2 BACKGROUND

The playstyle is closely related to the mapping from the states of the game environment (i.e., the visual observations or game screens in a video game) to the actions taken by the agent/player (which actually being analogous to the *policy*). Hence, the metric of playstyle intuitively would like to compare if two agents intend to take similar actions with respect to (nearly) the same states. However, when the observation space is high-dimensional or continuous, it is typically hard to find the same observations across different game recordings as the basis for further comparing the actions. In order to deal with such a problem, we introduce a method that learns the discrete representations of the observation space, such that the probability of having the same states across recordings significantly increases. A naive solution for reducing the observation space could be from the simple downsampling strategy, for instance, downsampling a given game screen into an image with a much smaller resolution. However, even when we have a low-resolution image after performing downsampling, e.g., 8×8 image size, with assuming there are only 16 possible intensity values for each pixel, the state space will end up with the size of 16^{64}, which is still too large to be feasible for our performing the further metric computation. Another well-known solution for obtaining discrete representations comes from the Vector Quantised-Variational AutoEncoder (VQ-VAE), proposed by [van den Oord et al., 2017], where a well-configured VQ-VAE model can extract latent discrete states with contextual information in video games, in which it fits the needs for our playstyle metric.

A typical VQ-VAE model has three components: encoder, decoder, and the embedding space, as shown in Figure 1. In a VQ-VAE model, the encoder first projects the image observation o to a latent feature map z_e. Then, the nearest neighbor method with respect to K shared tensors $E = \{e_1, \cdots, e_K\}$ is used to turn z_e into a discrete representation $\overline{s}$, that is, we replace each cell z_e^i of the z_e feature map with its nearest neighbor in E by the following equation.

$$q(\overline{s} = k|o) = \begin{cases} 1 & \text{for } k_i = \arg\min_j ||z_e^i(o) - e_j||^2, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Basically, the discrete state $\overline{s}$ is a ordered group of indexes to the elements in E, in which $\overline{s}$ can be easily recovered back to the continuous form z_q by creating a feature map (having same size as z_e) with its cells coming from the corresponding tensors in E of each index in $\overline{s}$. Finally, the decoder attempts to reconstruct the observation o from the restored latent code z_q. Derived from VQ-VAE, we develop a novel extension to further reduce the latent discrete space for enabling the computation of our playstyle metric in Section 4.

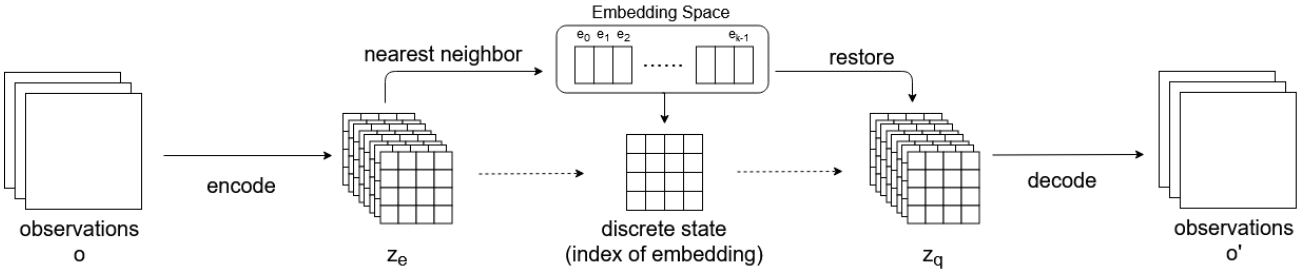

Figure 1: Illustration of VQ-VAE architecture [van den Oord et al., 2017]

3 PLAYSTYLE METRIC

This section presents our definition for playstyle metrics. In Section 3.1, we describe the terminology of the playing dataset and playstyle. Then, we define the playstyle distance in Section 3.2 and discuss its properties in Section 3.3.

3.1 REPRESENTATIONS OF PLAYSTYLES

In this subsection, we present the representation of the playstyles. An agent plays a set of game episodes, which are recorded as game recordings and collected as playing datasets for playstyle evaluation. In reality, an agent may play in several different styles in these game episodes. For example, an agent tends to drive at the speed of 60 in a segment of a game episode but at the speed of 90 in some other segment. However, for simplicity of analysis, in this paper, we assume that an agent always plays in a single playstyle in a set of game recordings, which are collected as a playing dataset; that is, *a playing dataset represents one playstyle*. So, some datasets collected from a single agent are supposed to have the same playstyle or have a high similarity in playstyle. For example, let an AI-bot tend to play at the speed of 60. Thus, all datasets collected from the bot should have high similarity.

A playstyle metric is to measure the similarity of two playstyles. In this paper, we measure the playstyle distance between two datasets (representing two playstyles), which will be described in the next subsection instead. If these two playstyles have a small playstyle distance, we say that the two corresponding playstyles tend to have high similarity.

3.2 PLAYSTYLE DISTANCE

Our metric aims to measure how close two playing datasets A and B are. A dataset, say A, is a set of pairs of observations and their corresponding actions in the action space $\mathcal{A}$. Let O_A denotes its observation set, a set of observations.[2] For an observation o, an action $a \in \mathcal{A}$ is associated with o to indicate how the agent acts to o. If the action space is discrete (e.g., the keyboard inputs), we can use the categorical distribution for modeling the action distribution. If the action space is continuous (e.g., inputs from a scroll wheel), we turn to use the multivariate normal distribution. In our metric, an observation o is mapped to a discrete state $\overline{s}$ via a mapping function ϕ from an observation set O to a set of states $\overline{S}$. In this paper, a mapping function based on state discretization is described in Section 4.

In order to compare the action distributions between two datasets, say A and B, we first derive the intersection $\overline{S}_\phi$ of the discrete states from O_A and O_B respectively, where

$$\overline{S}_\phi(A, B) = \phi(O_A) \cap \phi(O_B). \tag{2}$$

Given a state $\overline{s} \in \overline{S}_\phi$ and its corresponding action distributions π_A and π_B from A and B respectively, define a policy distance on $\overline{s}$ as

$$\overline{d}_\phi(A, B|\overline{s}) = W_2(\pi_A, \pi_B|\overline{s}), \tag{3}$$

where 2-Wasserstein distance (W_2) [Kantorovich, 1939, Vaserstein, 1969] is used to calculate the distance between distributions in this paper, as it is widely used in the metric of generative models, e.g., FID [Heusel et al., 2017]. The idea of using this way is to compare actions on those observations mapped to the same states only.

This paper presents two methods to calculate the playstyle distance between two datasets A and B with the mapping function ϕ. The first is to average the distance uniformly over all the intersected states as

$$d_\phi(A, B) = \frac{\sum_{\overline{s} \in \overline{S}_\phi(A,B)} \overline{d}_\phi(A, B|\overline{s})}{|\overline{S}_\phi(A, B)|}, \tag{4}$$

where every state $\overline{s}$ in $\overline{S}_\phi(A, B)$ has the same importance for the playstyle metric. Obviously, it is symmetric in the sense of $d_\phi(A, B) = d_\phi(B, A)$. The second is to calculate the expected distance according to the observation distribution of O_A and O_B over $\overline{S}_\phi$, defined as

$$d_\phi(A, B) = \frac{d_\phi(A|B)}{2} + \frac{d_\phi(B|A)}{2}, \tag{5}$$

[2] An observation is usually represented by a tensor in the way of how a player observes the game, e.g., consecutive 4 images of the game screen is a common observation in the video games [Mnih et al., 2015].

where

$$d_\phi(X|Y) = \mathbb{E}_{o\sim O_Y, \phi(o)\in\overline{S}_\phi(X,Y)}[\overline{d}_\phi(X, Y|\phi(o))]. \quad (6)$$

This method is symmetric too, but the expected distance weighs more for those with high observation occurrences in the game playing. To distinguish both methods, let us illustrate the following case for two states: first, many observation occurrences on both A and B, say n_A observation occurrences in A and n_B observation occurrences in B, are mapped to a state, say $s_1 \in \overline{S}_\phi$; second, only one observation occurrence on each of A and B are mapped to a set, say $s_2 \in \overline{S}_\phi$. In this case, the policy distance for s_1 weighs n_A times more for $d_\phi(A|B)$ (respectively n_B times more for $d_\phi(B|A)$) in the second method than the first, while the policy distance for s_1 weighs one for both methods. An example in more detail is given in the appendix. For the above two, if $\overline{S}_\phi = \varnothing$, the playstyle distance is set to undefined which will be further described in the section for experiments, as there is no common states across two playing datasets to measure their playstyle similarity. For simplicity of discussion, the rest of this paper are based on the second only, since the experiment results described in the appendix shows the second performs better than the first.

In practical implementation, we add an additional condition to $\overline{S}_\phi$ with a threshold t to discard the rarely visited states that tends to be unstable during the metric computation. With the threshold t, we use the following, instead of $\overline{S}_\phi$.

$$\overline{S'}_\phi(A, B, t) = \{\overline{s}|\overline{s} \in \overline{S}_\phi(A, B), F_\phi(\overline{s}, A) \geq t, F_\phi(\overline{s}, B) \geq t\}, \quad (7)$$
where $F_\phi(s, X) = |\{o|o \in O_X, \phi(o) \in \{s\}\}|$.

3.3 DISCUSSION

As motivated previously in Section 1, prior works on modeling player behaviors usually require domain-specific knowledge to extract relevant features or define target behaviors, which is hard to be generalized to other video games. Our playstyle metric in the previous subsection addresses the following two issues. The first one is *generality*. In our metric, we have no assumption on the targeting playstyle label, except for states and actions. In other word, the metric is capable of differentiating several playstyles in a game from various playing datasets, such that the metric does not need any predefined or specific features to the targeted playstyles, such as the playing speeds, or positions for racing cars.

The second one is *consistency*. As described in the previous subsections, two datasets with the same playstyle (or from the same agent) tends to have small playstyle distance in comparison to those with different playstyles, in terms of the metric. While playstyle similarity or distance is subjective in some sense, we assume that there exists an ideal or oracle metric to measure playstyle distance. A metric of estimating playstyle distance $d^*(A, B)$ is said to be consistent to an ideal similarity metric, if the following is satisfied.

Consistency Given any three playing datasets A, B and C, the similarity between A and C is higher than that between B and C in terms of the oracle metric, if and only if $d^*(A, C) < d^*(B, C)$.

In this paper, we argue that the expected playstyle distance in (6) in our metric distinguishes the distance relatively after a sufficient large number of samples. Namely, we argue that the following property is satisfied for any three playing datasets A, B and C, where the similarity between A and C is higher than that between B and C.

Argument Given a sufficient large number of samples, the expected playstyle distance in (6) satisfies the following: $d_\phi(A|C) < d_\phi(B|C)$ and $d_\phi(C|A) < d_\phi(C|B)$.

The argument $d_\phi(A|C) < d_\phi(B|C)$ implies that the expected distance from C to A is closer than that from C to B according to C's observation distribution. Next, the argument $d_\phi(C|A) < d_\phi(C|B)$ is similar, but in a different aspect where the observation distribution is reversed, namely, the targeting datasets A and B use their own observation distribution to calculate the expected distance to C. We can consider the targeting datasets A and B have consensus on the relation of distance value. With these arguments, we derive that after a sufficient large number of samples the playstyle distance given in (6) is consistent as follows.

$$\begin{aligned} d_\phi(A, C) &= d_\phi(A|C)/2 + d_\phi(C|A)/2 \\ &< d_\phi(B|C)/2 + d_\phi(C|B)/2 \\ &= d_\phi(B, C) \end{aligned} \quad (8)$$

4 HIERARCHICAL STATE DISCRETIZATION

As described in Section 3.2, we need a proper discrete state mapping function ϕ to find the intersection states between playing datasets. Thus, we develop a novel discrete representation learning method for enabling our proposed playstyle metric in the high dimensional observation space. To start with, we first provide the description on the design of our discrete representation learning in Subsection 4.1. Next, we introduce the gradient copy trick in VQ-VAE in Subsection 4.2. Finally, the training procedure of our model is defined in Subsection 4.3.

4.1 ARCHITECTURE DESIGN

A well-configured VQ-VAE model is able to extract the contextual information from raw observations in the video games [van den Oord et al., 2017]; however, the size of state

space in a typical VQ-VAE model could be still too large for our playstyle metric computation. To better control the size of state space, while keeping the capacity of the latent representation during the model optimization, we propose a multi-hierarchy stacked structure of the VQ-VAE model, named as *Hierarchical State Discretization* (**HSD**), where its model architecture is shown in Figure 2.

The convolutional encoder of our HSD in the base hierarchy (hierarchy 0, closest to the original observation space) follows the original design in a VQ-VAE model, while we add extra fully-connected encoders above the base hierarchy for controlling the discrete state space. Basically, a fully-connected encoder (in higher hierarchy other than the base one) takes the latent representation obtained by the previous hierarchy as its input and outputs a latent code, which is then folded to the representation composed of B cells of 1-D feature map and gone through the vector quantization procedure afterward as VQ-VAE does. The size of the discrete state space during such procedure is K^B, where K is the number of candidate tensors used for quantization in the embedding space at this hierarchy.

In the base hierarchy, in addition to the typical decoder taking care of the observations, we have another decoder related to the policy for encouraging discrete representations to better capture the information of actions, which is also quite important for our playstyle metric computation. Every hierarchy above the base one has a fully-connected decoder after the vector quantization procedure, which reconstructs the latent representations in the lower hierarchy. Moreover, we propose a weighted sum block to combine the latent representation of hierarchy h and the decoder output of hierarchy $h+1$ as the input for the decoder of hierarchy h, where the coefficients for such weighted sum are α^h (sampled from uniform distribution between 0 and 1) and $1-\alpha^h$ respectively, as shown in Figure 2. Such mechanism of using weighted sum blocks helps encouraging the decoder of the base hierarchy to utilize the outputs from all the hierarchies, thus the model training (i.e. gradients propagated from the objective in the base hierarchy) can smoothly contribute to all the subnetworks (i.e. decoders and encoders) in our HSD. For learning our HSD model, in order to enable the gradient propagation stopped by the nearest neighbor method of vector quantization, we follow the gradient copy trick proposed in VQ-VAE and extend it for our HSD architecture, which will be detailed in the next subsection.

It is worth noting that, there exists another well-known hierarchical VQ-VAE, the VQ-VAE-2 framework proposed by [Razavi et al., 2019], which is however quite different from our HSD in two perspectives: (1) VQ-VAE-2 is aiming to generate high-quality images with the autoregressive prior in the latent representation, while our HSD instead attempts to discover the intersection states between playing datasets; (2) VQ-VAE-2 jointly uses the discrete latent maps obtained from multiple hierarchies to generate the output image, thus its main purpose is to *increase* the latent states for producing delicate image details, while our HSD is proposed to *reduce* the number of latent states. Moreover, the architectures of VQ-VAE-2 and our HSD are distinct from each other.

4.2 GRADIENT COPY TRICK

There are two losses in gradient computation of a VQ-VAE model: L_{rec} is the reconstruction error between the observations o and their corresponding reconstruction o', and L_{vq} is the vector quantization (VQ) loss between the latent representation z_e and its nearest code z_q in the embedding space (cf. Figure 1). We adopt the Huber function [Huber, 1992] D for computing the reconstruction error $D(o, o')$ as well as the VQ loss $D(z_e, z_q)$:

$$L_{rec} = \mathbb{E}[D(o, o')], \quad L_{vq} = \mathbb{E}[D(z_e, z_q)] \tag{9}$$

Let $\theta_{encoder}$ be the weights of the encoder in a VQ-VAE model. The gradients of $\theta_{encoder}$ are computed by the chain rule as Equation 10 with a gradient copy trick (i.e. the gradients are copied from decoder input z_q to encoder output z_e [van den Oord et al., 2017]) to make the backpropagation going through the nearest neighbor path, where β is a coefficient to control the updating speed of the embedding vectors, with a default value set to 0.25 suggested by VQ-VAE.

$$\nabla\theta_{encoder} = \frac{\partial L_{rec}}{\partial z_q} \times \frac{\partial z_e}{\partial \theta_{encoder}} + \beta \times \frac{\partial L_{vq}}{\partial \theta_{encoder}} \tag{10}$$

4.3 TRAINING HSD MODEL

Here we introduce the objective functions for training our proposed hierarchical state dicretization (HSD) framework. First, we adopt the same reconstruction loss L_{rec} as original VQ-VAE model on the observations. Second, for further encouraging the features extracted from our HSD is also aware of the information of policy/action, we add an extra loss function L_π defined as follows.

$$L_\pi = \begin{cases} \mathbb{E}[H(\pi_{label}, \pi_{decoder})] & \text{if } \mathcal{A} \text{ is discrete,} \\ \mathbb{E}[D(\pi_{label}, \pi_{decoder})] & \text{if } \mathcal{A} \text{ is continuous,} \end{cases} \tag{11}$$

where H stands for the cross entropy, $\pi_{decoder}$ and π_{label} are the action prediction produced by the policy decoder of HSD and the corresponding groundtruth label, respectively. Last, we revise the vector quantization loss of VQ-VAE to a hierarchical fashion L^h_{vq} with a hierarchy index h as follows to better fit our HSD framework:

$$L^h_{vq} = \mathbb{E}[D(z^h_e, z^h_q)] \tag{12}$$

As mentioned previously, the decoder in the base hierarchy (i.e., hierarchy index 0) consists of the observation decoder and the policy decoder, where their weights are θ_{rec} and θ_π respectively. The weights of the encoders, the embedding,

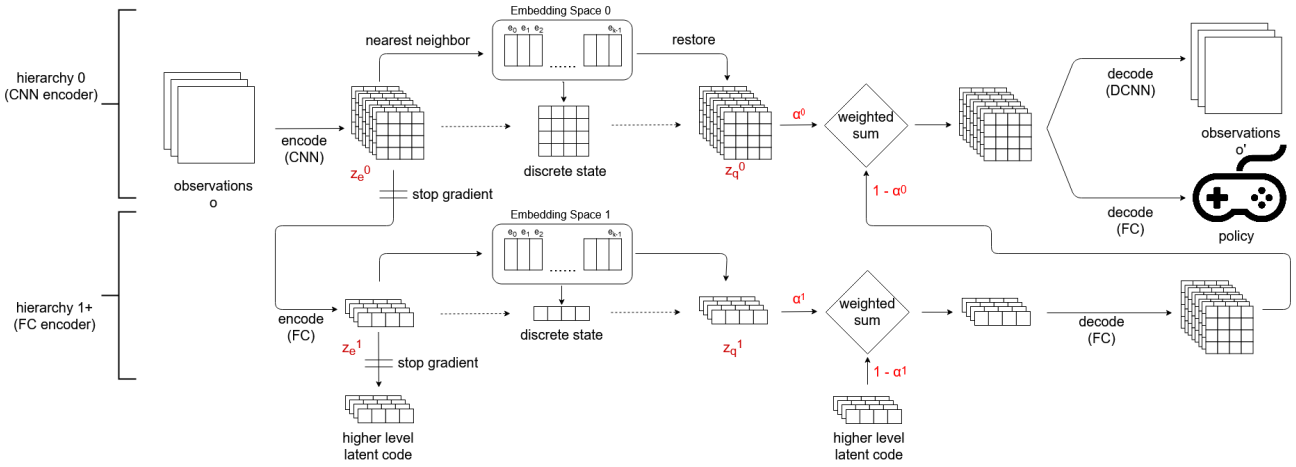

Figure 2: Architecture of our proposed hierarchical state discretization (HSD) model

	Pixel	LRD	HSD	Continuous
25 Styles	4.00	4.00	**69.73**	time out
5 Speed Styles	20.00	20.00	**91.20**	27.60
5 Noise Styles	20.00	20.00	**81.33**	23.20

Table 1: TORCS Playstyle Accuracy (%). Please refer to Section 5.1 for detailed description.

and the decoders with the hierarchy index h are then denoted as θ^h_{enc}, θ^h_{embed}, and θ^h_{dec} respectively. The gradients of each component are computed in Equation 13-Equation 14.

$$\nabla\theta^h_{enc} = \frac{\partial L_{rec}}{\partial z^h_q} \times \frac{\partial z^h_e}{\partial \theta^h_{enc}} + \frac{\partial L_\pi}{\partial z^h_q} \times \frac{\partial z^h_e}{\partial \theta^h_{enc}} + \beta \times \frac{\partial L^h_{vq}}{\partial \theta^h_{enc}} \tag{13}$$

$$\nabla\theta^h_{embed} = \frac{\partial L^h_{vq}}{\partial \theta^h_{embed}}, \quad \nabla\theta^h_{dec} = \frac{\partial L_{rec}}{\partial \theta_{dec}} + \frac{\partial L_\pi}{\partial \theta_{dec}}, \quad \nabla\theta_{rec} = \frac{\partial L_{rec}}{\partial \theta_{rec}}, \quad \nabla\theta_\pi = \frac{\partial L_\pi}{\partial \theta_\pi} \tag{14}$$

where the gradient copy trick is also adopted for the gradient computation of θ^h_{enc} .

5 EXPERIMENTS

This subsection evaluates our proposed playstyle metric on several video game platforms, including TORCS, RGSK, and seven Atari games. As described in Section 3, we assume that a playing dataset represents one playstyle and that an agent always plays in a single playstyle in a dataset. So, in our experiment, let different agents p produce their own datasets C_p as candidates, whose collection is denoted by $\mathcal{C} = \{C_p\}$. Then, let these agents produce additional datasets T_p as targets. For T_p, find the most matching playstyle C among the collection $\mathcal{C}$ in the following formula.

$$C = \arg\min_{C' \in \mathcal{C}} d_\phi(T_p, C'). \tag{15}$$

If C is C_p, also played by the agent p, then it is said to be a correct prediction. Then, the *playstyle prediction accuracy* is defined as the ratio of correct predictions over the total number of trials, namely 100 trials for all T_p in our experiments. The accuracy is used to evaluate the performance of the playstyle metric.

In Subsection 5.1, we analyze the playstyles of rule-based AI players in the racing game, TORCS. Then, in Subsection 5.2, we recruit human players to play the racing game, RGSK, for evaluating the metric upon the practical playstyles. Finally, we adopt several learning-based AI players to play different Atari games in Subsection 5.3 to study the efficacy of our proposed metric on a wide range of video games.

5.1 TORCS

In this subsection, our experiment is to analyze the playstyles of the racing game, TORCS [Wymann et al., 2014], by the rule-based AI agents developed in [Yoshida et al., 2017], which are able to control the acceleration and steering angle of the race car to maintain a target speed in a given default track. Besides, since the randomness in the single-player mode of TORCS is low, we also inject noises sampled from a normal distribution into the action values for generating diverse games.

Now, let us produce playing datasets $C_{s,n}$ as candidates, which contains a set of samples, namely 1024 samples in our experiment, from game episodes played by the above rule-based agent with the speed s and a noise level n. From

Section 3, each $C_{s,n}$ represents a playstyle. In this experiment, we use 5 different target speeds (i.e. 60, 65, 70, 75, and 80) and 5 noise levels (denoted by $n1$, $n2$, $n3$, $n4$ and $n5$) for the action value. For each noise level, the action noises are sampled from two zero-mean Gaussian distribution related to the steers and accelerations; and the standard deviation (of the steers and accelerations respectively) for 5 noise levels are (0.01,0.005), (0.02,0.01), (0.03,0.015), (0.04,0.02), and (0.05,0.025). Thus, we produce 25 datasets as candidates in total. For playstyle prediction, we also produce the corresponding 25 datasets $T_{s,n}$ as targets.

In this experiment, we use three different discrete representations from the training of HSD as described in Section 4, where the HSD model includes one and only one hierarchy above the base hierarchy, and the discrete state space is set to 2^{20}. Table 1 shows a playsytle accuracy, 69.73%, averaged from the three HSD representations. This table also shows the playstyle accuracy for less datasets in the following two experiments. First, given a noise level n at random, consider the five speed datasets, $C(s, n)$, where s are the above five speeds. Second, similar to the first, given a speed s at random, consider the five noise datasets, $C(s, n)$, where n are the above five noise levels. The results shows higher playstyle accuracy, 91.20% for five speeds, and 81.33% for five noise levels.

For comparison, we present the following three different metrics. First, if we still use discrete representation, we consider two metrics with different ϕ. The first, called *Pixel*, straightforwardly uses the raw pixel values of 4 consecutive screen observations as the states, namely $\overline{s} = \phi(o) = o$. The second, called *LRD*, uses low-resolution downsampling that resizes the raw screens (of observations) from (64,64) to (8,8) with bilinear downsampling and also shrinks the intensity value range from 0-255 to 0-15 by dividing the intensity value by 16 and discarding the remainder. The last metric, called *Continuous*, does not use discrete representation. It simply follows a common image distance metric used in generative models, namely, FID [Heusel et al., 2017], which simply compares the distance of two image datasets in terms of the classification latent distribution, trained from ImageNet [Deng et al., 2009] for identifying the similarity between real images and generative images. However, since the model used for FID is for ImageNet and cannot directly be used in our game environment, we use the latent distribution of our HSD model, as those of z_e^1 in Figure 2.

For the three metrics, we experiment in the same way as that for HSD. Table 1 show that playstyle accuracies for these methods are obviously much lower than that for HSD. Note that the performances of Pixel and LRD are low since there are no intersection states in the experiment. For Continuous, it runs out of time for the one with 25 styles, we can observe that HSD clearly outperforms from five speed styles and noise styles. The metric Continuous does not have high accuracy since the observation distribution does not reflect actions which are important in terms of playstyle.

5 Speed Styles	2^{16}	2^{20}	2^{24}
$t = 1$	89.33	89.60	**91.47**
$t = 2$	89.73	**91.20**	89.60
$t = 4$	90.13	**90.27**	86.67
5 Noise Styles	2^{16}	2^{20}	2^{24}
$t = 1$	**74.13**	72.67	73.60
$t = 2$	78.27	**81.33**	78.67
$t = 4$	84.53	**85.07**	82.40

Table 2: The comparison of using different size of the HSD state space and threshold in evaluating the playstyle accuracy (%) in TORCS. Please refer to Section 5.1 for detailed description.

	Nitro	Surface	Position	Corner
Pixel	16.67	16.67	16.67	16.67
LRD	33.33	16.67	33.17	42.50
HSD	**93.11**	**99.83**	**99.67**	**99.56**

Table 3: RGSK Playstyle Accuracy (%). Please refer to Section 5.2 for detailed description.

We further study the proper size of state space and the threshold t of our metric in Table 2. It is clear that using a large threshold tends to lead to high accuracy in noise styles, which inherently include big randomness. However, the different thresholds do not make large difference in speed styles. From the table, we observe that 2^{20} is an appropriate size of state space since it performs well in most cases. Thus, in the rest of our experiments we use $t = 2$ and 2^{20} state space as a default setting. More comparisons and experiment details can be found in our appendix.

5.2 RGSK

In this experiment, we further investigate our playstyle metric with human players, based on another racing game, RGSK, in which its development pack is available on the Unity Asset Store. For RGSK, we use Unity ML-Agents Toolkit [Juliani et al., 2018] to sample data, pairs of observations and actions, from human players. Each of human players is requested to maintain a consistent playstyle during game playing, which is sampled as a dataset representing the playstyle. For example, a player are requested to use the N_2O boosting system, called Nitro in this paper, to accelerate the car in his/her own style. In this experiment, six players, denoted by p_1 to p_6, are requested to play in their own playstyles of using Nitro, where six datasets C_1 to C_6

	Asterix	Breakout	MsPacman	Pong	Qbert	Seaquest	SpaceInvaders
Pixel	67.10	59.05	70.95	34.20	30.00	14.95	30.35
LRD	92.20	59.85	**96.75**	**94.15**	83.70	42.70	56.10
HSD	**98.65**	**93.83**	93.85	91.55	**94.58**	**93.87**	**79.72**

Table 4: Atari Playstyle Accuracy (%). Please refer to Section 5.3 for detailed description.

from the six players are collected respectively as candidates.

Table 3 shows that the playstyle accuracy for HSD reaches 93.11%, which is much better than those for Pixel and LRD. The experiment settings are nearly the same as TORCS, except for the following differences: a screen is 72×128, instead of 64×64; the action space is discrete with 27 actions, a combination of three steering directions, three acceleration levels and three driving direction controls (i.e. driving forward, backward, and neutrally).

In addition to Nitro, we do experiments for other playstyles, where the players are requested to play consistently in their own styles in the following three style dimensions, called Surface, Position and Corner. For Surface, players are requested to maintain their own playstyles consistently on driving on either the road surface or the grass surface. For Position, players are requested to maintain their own playstyles consistently on driving the car in the inner or outer of the track. For Corner, players are requested to maintain their own playstyles consistently on passing a corner via drifting or slowing down with a break. Table 3 also includes the performances of these playstyle dimensions. From the table, the playstyle accuracy for HSD reaches above 99% for these playstyle dimensions, which again shows consistent outperformance in comparison to those for Pixel and LRD. The aforementioned experiments also demonstrate the generality of our metric on different playstyle dimensions, as our metric does not need any prior knowledge about which kinds of playstyles to measure.

5.3 ATARI 2600

Finally, we measure the playstyles of learning-based AI agents for seven Atari games [Bellemare et al., 2013]. These agents are trained by four available RL algorithms, i.e. DQN [Mnih et al., 2015], C51 [Bellemare et al., 2017], Rainbow [Hessel et al., 2018], and IQN [Dabney et al., 2018]), provided by a deep reinforcement learning framework Dopamine [Castro et al., 2018, Such et al., 2019]. For each of these algorithms, five models are trained from initially different random seeds. Thus, there are 20 models in total serving as 20 agents, each of which is presumed to play in a unique playstyle.

The experiment measures the playstyle accuracy in a similar way to those in the previous sections. Namely, 20 agents are used to generate respectively 20 datasets as candidates. The experimental settings are similar to those in the previous subsections, except for the following modifications. Each screen for observation is 84×84 grayscale images, and the action space is discrete.

Table 4 shows the performances of the three playstyle metrics, Pixel, LRD and HSD. We observe a phenomenon as follows. Since observations have low randomness in Atari games, the intersection of states of Atari games for both Pixel and LRD are larger than those in the previous subsections, thus leading to the case that these two metrics have much better performances than those in the previous subsections (i.e. TORCS and RGSK). Nevertheless, even under such phenomenon, our playstyle metric HSD remains to outperform the other two for five games, while performing slightly lower than LRD but still quite competitive for the other two games (i.e. MsPacman and Pong).

6 CONCLUSION

We propose a novel playstyle metric for video games to evaluate playstyle distances and predict playstyles, based on a novel scheme of learning hierarchical discrete representations. To our knowledge, such a metric is the first of its kind without prior knowledge of the games. Our experiment results shown in Section 5 also demonstrate high playstyle prediction accuracy of this metric on video game platforms, including TORCS, RGSK,and seven Atari games. In fact, our work also leaves some open problems for the future work. For example, measure the playsytles based on segments of trajectories, not just pairs of observations and actions, investigate the transitivity of playstyle distances, and judge whether an AI acts like a human. Besides, we make an assumption that each player has a single playstyle for the simplicity of analysis. It is possible to extend our metric to a more general case, which contains more than one playstyle per person/dataset.

Acknowledgements

This research is partially supported by the Ministry of Science and Technology (MOST) of Taiwan under Grant Number 109-2634-F-009-019 and 110-2634-F-009-022 through Pervasive Artificial Intelligence Research (PAIR) Labs, and the computing resource partially supported by National Cen-

ter for High-performance Computing(NCHC) of Taiwan. The authors also thank anonymous reviewers for their valuable comments.

Author Contributions

Briefly list author contributions. This is a nice way of making clear who did what and to give proper credit.

H. Q. Bovik conceived the idea and wrote the paper. Coauthor One created the code. Coauthor Two created the figures.

Acknowledgements

Briefly acknowledge people and organizations here.

All acknowledgements go in this section.

References

Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. In *International Conference on Learning Representations (ICLR)*, 2020.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 2013.

Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2017.

Boyan Bontchev and Olga Georgieva. Playing style recognition through an adaptive video game. *Computers in Human Behavior*, 2018.

Patrick Brombacher, Johannes Masino, Michael Frey, and Frank Gauterin. Driving event detection and driving style classification using artificial neural networks. In *IEEE International Conference on Industrial Technology (ICIT)*, 2017.

Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G. Bellemare. Dopamine: A research framework for deep reinforcement learning. *ArXiv:1812.06110*, 2018.

Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2018.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE International Conference on Computer Vision (ICCV)*, 2009.

Anders Drachen, Rafet Sifa, Christian Bauckhage, and Christian Thurau. Guns, swords and data: Clustering of player behavior in computer games in the wild. In *IEEE Conference on Computational Intelligence and Games (CIG)*, 2012.

Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In *International Conference on Machine Learning (ICML)*, 2018.

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

Peter J Huber. Robust estimation of a location parameter. In *Breakthroughs in Statistics*. Springer, 1992.

Arthur Juliani, Vincent-Pierre Berges, Esh Vckay, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents. *ArXiv:1809.02627*, 2018.

Leonid V Kantorovich. The mathematical method of production planning and organization. *Management Science*, 1939.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

OpenAI. OpenAI Five. `https://blog.openai.com/openai-five/`, 2018.

Johanna Pirker, Simone Griesmayr, Anders Drachen, and Rafet Sifa. How playstyles evolve: Progression analysis and profiling in Just Cause 2. In *International Conference on Entertainment Computing (ICEC)*, 2016.

Ali Razavi, Aäron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with VQ-VAE-2. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Osamu Shouno. Deep unsupervised learning of a topological map of vehicle maneuvers for characterizing driving styles. In *Conference on Intelligent Transportation Systems (ITSC)*, 2018.

Felipe Petroski Such, Vashisht Madhavan, Rosanne Liu, Rui Wang, Pablo Samuel Castro, Yulun Li, Jiale Zhi, Ludwig Schubert, Marc G. Bellemare, Jeff Clune, and Joel Lehman. An Atari model zoo for analyzing, visualizing, and comparing deep reinforcement learning agents. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

Wataru Takano, Akihiro Matsushita, Keijiro Iwao, and Yoshihiko Nakamura. Recognition of human driving behaviors based on stochastic symbolization of time series signal. In *IEEE/RJS International Conference on Intelligent Robots and Systems (IROS)*, 2008.

Tadahiro Taniguchi, Shogo Nagasaka, Kentarou Hitomi, Naiwala P. Chandrasiri, and Takashi Bando. Semiotic prediction of driving behavior using unsupervised double articulation analyzer. In *Intelligent Vehicles Symposium, (IEEE IV)*, 2012.

Tadahiro Taniguchi, Shogo Nagasaka, Kentarou Hitomi, Kazuhito Takenaka, and Takashi Bando. Unsupervised hierarchical modeling of driving behavior and prediction of contextual changing points. *IEEE Transactions on Intelligent Transportation Systems*, 2015.

Tadahiro Taniguchi, Shogo Nagasaka, Kentarou Hitomi, Naiwala P. Chandrasiri, Takashi Bando, and Kazuhito Takenaka. Sequence prediction of driving behavior using double articulation analyzer. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2016.

Anders Tychsen and Alessandro Canossa. Defining personas in games using metrics. In *International Academic Conference on the Future of Game Design and Technology (FuturePlay)*, 2008.

Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

Leonid Nisonovich Vaserstein. Markov processes over denumerable products of spaces, describing large systems of automata. *Problemy Peredachi Informatsii*, 1969.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 2019.

Bernhard Wymann, Christos Dimitrakakis, Andrew Sumner, Eric Espié, and Christophe Guionneau. Torcs: The open racing car simulator. `http://www.torcs.org`, 2014.

Georgios N. Yannakakis, Pieter Spronck, Daniele Loiacono, and Elisabeth André. Player modeling. In *Artificial and Computational Intelligence in Games*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013.

Naoto Yoshida, Alexander Khimulya, Billy Zhou, and Arun Kumar. Gym-TORCS. `https://github.com/ugo-nama-kun/gym_torcs`, 2017.

APPENDIX

A A STUDY OF SOME VARIANTS OF PLAYSTYLE DISTANCE

In our proposed playstyle metric, we suggest using 2-Wasserstein distance metric in computing the distance of action distributions and using the expected version in gathering the distances over the overlapping states. Here, we investigate some variants of calculating the playstyle distance on the racing games. We first give in Subsection A.1 the comparison of using a uniform version and expected version as our playstyle distance (cf. Equation (4) and (5) in the main paper); then in Subsection A.2, we conduct the study of using different distances between probability distributions to compute the policy distance.

A.1 UNIFORM DISTANCE OR EXPECTED DISTANCE

As described in Section 3, we propose to use the uniform version (cf. Equation (4)) or the expected version (cf. Equation (5)) to compute playstyle distances, which are modelled from slightly different aspects. In Table 5, we provide the experimental results for the two versions under the setting of using 2^{20} state space with different threshold t in $\overline{S'_\phi}$, for making comparison on the playstyle accuracy in TORCS, while in Table 6 the results in the RGSK racing game are presented. We can observe that, the expected version is apparently better in TORCS, and the difference between these two versions of playstyle distance is marginal in RGSK.

	5 Speed Styles	5 Noise Styles
Uniform ($t = 2$)	76.00	65.20
Uniform ($t = 4$)	83.47	74.40
Expected ($t = 2$)	91.20	81.33
Expected ($t = 4$)	90.27	85.07

Table 5: The comparison of TORCS playstyle accuracy (%) between using the uniform version (cf. Equation (4)) or the expected version (cf. Equation (5)) as our playstyle distance.

A.2 DIFFERENT DISTRIBUTION DISTANCE METRICS

In our playstyle metric, we use 2-Wasserstein distance (W_2) as the distance metric for action distribution. For simplicity of discussion, we use $\pi_A(\overline{s})$ to denote the action distribution of a dataset A under a specific state $\overline{s}$. For the policy in the discrete action space, we treat the categorical probability distribution as a simple vector according to the action probability and directly compute the L_2 distance between two action vectors as follows.

$$W_2(\pi_A, \pi_B|\overline{s}) = \|\pi_A(\overline{s}) - \pi_B(\overline{s})\|_2, \tag{16}$$

In video game AI, an action-value is usually presented as an action index, and there is no specific relation among these indices in terms of their numerical difference. Hence, the way we present the action distribution still meets the definition in the Wasserstein metric, which indicates that the distances based on each of the action values are of the same importance. For the policy in the continuous action space, we use the multivariate normal distribution to formulate the action distributions, and the W_2 distance is defined by the formula [Vaserstein, 1969] as follows.

$$\begin{aligned} &W_2(\pi_X, \pi_Y|\overline{s}) = \|m_X(\overline{s}) - m_Y(\overline{s})\|_2 + \\ &trace(C_X(\overline{s}) + C_Y(\overline{s}) - 2(C_Y(\overline{s})^{1/2} C_X(\overline{s}) C_Y(\overline{s})^{1/2})^{1/2}), \\ &\text{where } \pi_X(\overline{s}) = \mathcal{N}(m_X(\overline{s}), C_X(\overline{s})), \\ &\text{and } \pi_Y(\overline{s}) = \mathcal{N}(m_Y(\overline{s}), C_Y(\overline{s})). \end{aligned} \tag{17}$$

$m(\overline{s})$ and $C(\overline{s})$ are the mean vector and the covariance matrix respectively of the multivariate normal distribution under a specific state $\overline{s}$.

In machine learning, Kullback–Leibler divergence (mostly being abbreviated as KL divergence) is a common metric (sometimes, we call it "distance") to measure the difference between two probability distributions. Its computation upon the discrete probability distributions is as follows:

$$D_{KL}(P\|Q) = \sum_{x \in X} P(x) \log(\frac{P(x)}{Q(x)}) \tag{18}$$

We therefore also experiment to use KL divergence instead of W_2 in the RGSK playstyle comparison, which has a discrete action space. Since KL divergence is an asymmetric metric, we also try its average variant $M_{KL}(P, Q)$ as follows.

$$M_{KL}(P, Q) = \frac{D_{KL}(P\|Q) + D_{KL}(Q\|P)}{2} \tag{19}$$

In addition to W_2 and KL divergence, we also include W_1 metric, which uses L_1 norm in Wasserstein metric instead of L_2 norm, in our comparison. As the results based on all the aforementioned metrics (i.e. W_2, KL divergence, and W_1) being shown In Table 7, where the state space is 2^{20} with the threshold $t = 2$, we observe that using Wasserstein metric has better performance than KL divergence, and W_1 is sightly better than W_2 in this case. Please note that, even though the better performance of W_1, we adopt the W_2 as the distribution distance used for our playstyle metric computation due to its common application as well as the capability of reflecting the covariance between different action dimensions.

	Nitro	Surface	Position	Corner
Uniform ($t = 2$)	92.33	99.28	99.33	99.50
Uniform ($t = 4$)	86.94	96.44	95.56	95.61
Expected ($t = 2$)	93.11	99.83	99.67	99.56
Expected ($t = 4$)	88.78	98.39	98.33	98.44

Table 6: The comparison of RGSK playstyle accuracy (%) between using the uniform version (cf. Equation (4)) or the expected version (cf. Equation (5)) as our playstyle distance.

	W_1	W_2	D_{KL}	M_{KL}
Nitro	**95.28**	93.11	37.44	76.78
Surface	**100.00**	99.83	74.28	96.06
Position	**99.83**	99.67	58.06	91.94
Corner	**99.83**	99.56	56.33	87.33

Table 7: The comparison in terms of playstyle accuracy (%) among adopting different metrics for the action distributions. The video game environment used in this experiment is based on RGSK.

B AN EXAMPLE OF CALCULATING THE PLAYSTYLE DISTANCE

We give an example for calculating the playstyle distance, with the scenario as shown in Figure 3, where the action space is discrete and we adopt the W_2 distance as the distribution metric.

First, we have to compute the policy distances over intersection states (in the red border regions) as follows.

$$\begin{aligned}
\overline{d}_\phi(A, B|\overline{s_1}) &= \|(1, 0, 0) - (0, 0.5, 0.5)\|_2 \approx 1.225, \\
\overline{d}_\phi(A, B|\overline{s_2}) &= \|(0, 0.5, 0.5) - (0, 1, 0)\|_2 \approx 0.707, \\
\overline{d}_\phi(A, B|\overline{s_3}) &= \|(0, 0, 1) - (0, 0.5, 0.5)\|_2 \approx 0.707.
\end{aligned} \tag{20}$$

Next, we can calculate the playstyle distance as following the Equation (4):

$$\begin{aligned}
d_\phi(A, B) &= \frac{d_\phi(A|B)}{2} + \frac{d_\phi(B|A)}{2} \\
&\approx \frac{1.225 \times 0.5 + 0.707 \times 0.333 + 0.707 \times 0.167}{2} \\
&+ \frac{1.225 \times 0.4 + 0.707 \times 0.2 + 0.707 \times 0.4}{2} \\
&\approx \frac{0.966}{2} + \frac{0.914}{2} \\
&= 0.940
\end{aligned} \tag{21}$$

Finally, when we use a state count threshold $t = 2$ in the policy distance, $\overline{s_2}$ and $\overline{s_3}$ are then not included in the intersection state $\overline{S'_\phi}$, in which it leads to the playstyle distance computed as follows:

$$\begin{aligned}
d_\phi(A, B) &= \frac{d_\phi(A|B)}{2} + \frac{d_\phi(B|A)}{2} \\
&\approx \frac{1.225 \times 1.0}{2} + \frac{1.225 \times 1.0}{2} \\
&= 1.225
\end{aligned} \tag{22}$$

C A STUDY OF SELECTING THE SIZE OF STATE SPACE

How to select a proper state space in our metric is an important topic. We present a wide search of using different sizes of the state space in TORCS, where the results are shown in Table 8 based on the playstyle accuracy. We can observe that: If the state space is too small, it is hard to obtain good accuracy; However, if we use a large state space like 2^{64}, the inherent problem of high-dimensional observations would occurs, in which the size of the intersection state set is too small to give a precise playstyle computation. In results, $2^{16} \sim 2^{32}$ is a good range to explore, where we suggest to use 2^{20} as a default space size in our main paper. Also, a vanilla VQ-VAE has a too large state space that can not find intersection states.

D EXPERIMENTAL SETTINGS

In our experiments, we use the same network architecture in all video games, as described in the following. The encoder is based on a simple version of IMPALA network, which is proposed by Espeholt et al. [2018]. We show the full architecture in Figure 4, where the unit count X in the hierarchical decoder depends on the dimension of the CNN encoder output. For example, in Atari games, $X = 11 \times 11 \times 32$. The unit count around 500 means that we use the closest divisible number to the number of cell size B. For example, in 2^{16} state space, the number is 512, which is divisible to 16, and in 2^{20} state space, the number is 500, which is divisible to 20. A truncated normal initializer initializes the embedding tensors with a standard deviation of 0.02. When training these HSD models, we add noises sampled from a normal distribution with zero mean and standard deviation 4 to the raw pixel values (0-255); after that, the pixel values are divided by 255 as a simple method for pre-processing the observations.

D.1 TRAINING HSD MODELS IN DIFFERENT EXPERIMENTS

The training data for the HSD models in TORCS experiments is sampled from human players with a joystick. There are 14502 observation-action pairs in the training data; For training the HSD models in RGSK experiments, we sample

	2^0	2^4	2^8	2^{16}	2^{32}	2^{64}	Pixel	VQ-VAE
$t = 2$	79.40	70.93	68.93	**89.73**	84.53	66.40	20.00	20.00
$t = 4$	79.40	68.53	72.80	**90.13**	88.80	43.87	20.00	20.00

(a) The playstyle accuracy(%) over 5 different target speeds.

	2^0	2^4	2^8	2^{16}	2^{32}	2^{64}	Pixel	VQ-VAE
$t = 2$	38.28	52.80	63.87	**78.27**	74.53	55.73	20.00	20.00
$t = 4$	38.28	53.07	61.73	**84.53**	81.33	41.33	20.00	20.00

(b) The playstyle accuracy(%) over 5 different noise levels.

Table 8: The experiments of using different sizes of the state space in TORCS.

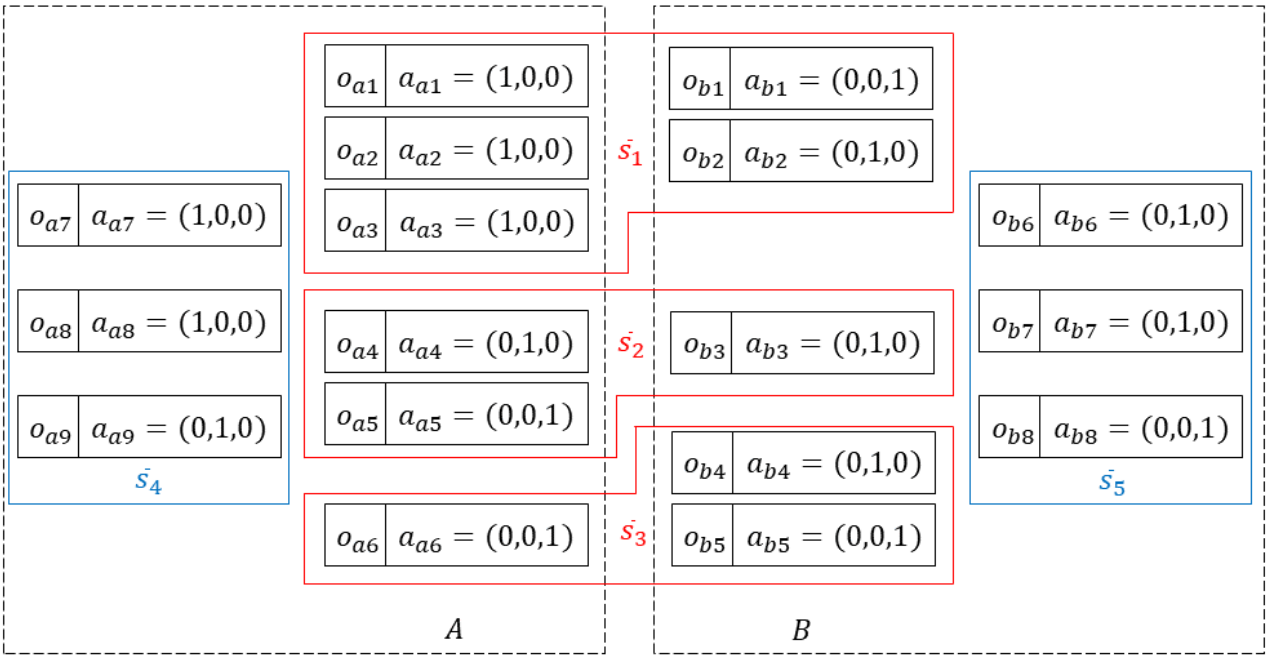

Figure 3: An example scenario for showcasing the computation of playstyle distance.

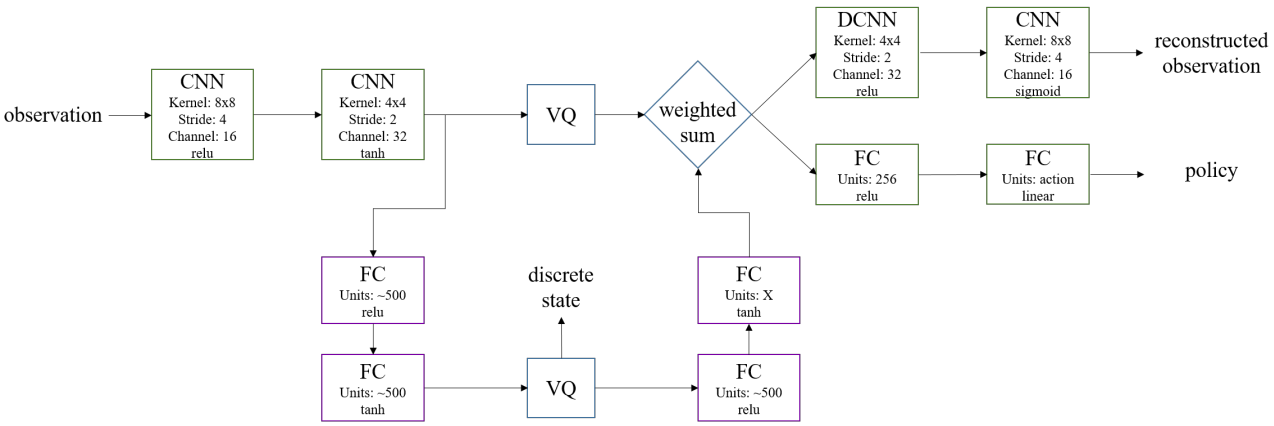

Figure 4: The architecture of the HSD model used in all video game experiments.

some demonstrations from human players. There are 26 episodes in the training data, with each consisting of two complete circuits; For training the HSD models in Atari experiments, we use the first Rainbow [Hessel et al., 2018] model in Dopamine [Castro et al., 2018] with ϵ-greedy (ϵ = 0.01) to sample 100 episodes as the training data for HSD models in each game.