HyAR: Addressing Discrete-Continuous Action Reinforcement Learning via Hybrid Action Representation

Anonymous authors
Paper under double-blind review

Abstract

Discrete-continuous hybrid action space is a natural setting in many practical problems, such as robot control and game AI. However, most previous Reinforcement Learning (RL) works only demonstrate the success in controlling with either discrete or continuous action space, while seldom take into account the hybrid action space. One naive way to address hybrid action RL is to convert the hybrid action space into a unified homogeneous action space by discretization or continualization, so that conventional RL algorithms can be applied. However, this ignores the underlying structure of hybrid action space and also induces the scalability issue and additional approximation difficulties, thus leading to degenerated results. In this paper, we propose **Hy**brid **A**ction **R**epresentation (**HyAR**) to learn a compact and decodable latent representation space for the original hybrid action space. HyAR constructs the latent space and embeds the dependence between discrete action and continuous parameter via an embedding table and conditional Variational Auto-Encoder (VAE). To further improve the effectiveness, the action representation is trained to be semantically smooth through unsupervised environmental dynamics prediction. Finally, the agent then learns its policy with conventional DRL algorithms in the learned representation space and interacts with the environment by decoding the hybrid action embeddings to the original action space. We evaluate HyAR in a variety of environments with discrete-continuous action space. The results demonstrate the superiority of HyAR when compared with previous baselines, especially for high-dimensional action spaces.

1 Introduction

Deep Reinforcement learning (DRL) has recently shown a great success in a variety of decision-making problems that involve controls with either discrete actions, such as Go (Silver et al., 2016) and Atari (Mnih et al., 2015), or continuous actions, such as robot control (Schulman et al., 2015; Lillicrap et al., 2015). However, in contrast to these two kinds of homogeneous action space, many real-world scenarios requires more complex controls with discrete-continuous hybrid action space, e.g., Robot soccer (Masson et al., 2016) and Real Time Strategic games (Xiong et al., 2018). For example, in robot soccer, the agent not only needs to choose whether to shoot or pass the ball (i.e., discrete actions) but also the associated angle and force (i.e., continuous parameters). Such a hybrid action is also called parameterized action in some previous works (Hausknecht & Stone, 2016; Fu et al., 2019). Unfortunately, most conventional RL algorithms cannot deal with such a heterogeneous action space directly, thus preventing the application of RL in these kinds of practical problems.

To deal with hybrid action space, the most straightforward approach is to convert the heterogeneous space into a homogeneous one through discretization or continualization. However, it is apparent that discretizing continuous parameter space suffers from the scalability issue due to the exponentially exploring number of discretized actions; while casting all discrete actions into a continuous dimension produces a piecewise-function action subspace, resulting in additional difficulties in approximation and generalization. To overcome these problems, a few recent works propose specific policy structures to learn DRL policies over the original hybrid action space directly. Parameterized Action DDPG (PADDPG) (Hausknecht & Stone, 2016) makes use of a DDPG (Lillicrap et al., 2015) structure where the actor is modified to output a unified continuous vector as the concatenation of values for all discrete actions and all corresponding continuous parameters. By contrast, Hybrid PPO

(HPPO) (Fan et al., 2019) uses multiple policy heads consisting of one for discrete actions and the others for corresponding continuous parameter of each discrete action separately. These methods are convenient to implement and are demonstrated to effective in simple environments with low-dimensional hybrid action space. However, PADDPG and HPPO neglect the dependence between discrete and continuous components of hybrid actions, thus can be problematic since the dependence is vital to identifying the optimal hybrid actions in general. Besides, the modeling of all continuous parameter dimensions all the time introduces redundancy in computation and policy learning, and may also have the scalability issue when the hybrid action space becomes high-dimensional.

To model the dependence, Parameterized DQN (PDQN) (Xiong et al., 2018) proposes a hybrid structure of DQN (Mnih et al., 2015) and DDPG. The discrete policy is represented by a DQN which additionally takes as input all the continuous parameters output by the DDPG actor; while the DQN also serves as the critic of DDPG. Due to the DDPG actor's modeling of all parameters, PDQN also have the redundancy and potential scalablity issue. In an upside-down way, Hierarchical Hybrid Q-Network (HHQN) (Fu et al., 2019) models the dependent hybrid-action policy with a two-level hierarchical structure. The high level is for the discrete policy and the selected discrete action serves as the condition (in analogy to subgoal) which the low-level continuous policy conditions on. This can be viewed as a special two-agent cooperative game where the high level and low level learn to coordinate at the optimal hybrid actions. Although the hierarchical structure seems to be natural, it suffers from the high-level non-stationarity caused by off-policy learning dynamics (Wang et al., 2020), i.e., a discrete action can no longer induce the same transition in historical experiences due to the change of the low-level policy. In contrast, PADDPG, HPPO and PDQN are stationary in this view since they all learn an overall value function and policy thanks to their special structures, which is analogous to learning a joint policy in the two-agent game. All the above works focus on policy learning over original hybrid action space. As summarized in Table 1, none of them is able to offer three desired properties, i.e., scalability, stationarity and action dependence, at the same time.

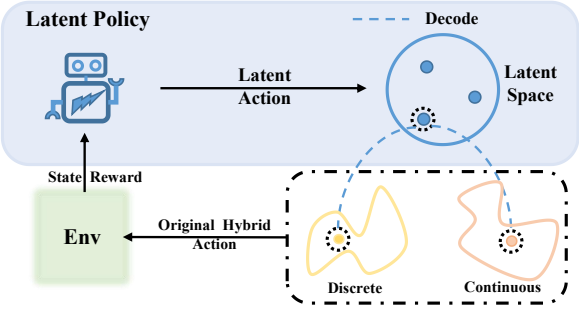


Figure 1: An overview of HyAR. Agent learns a latent policy in the latent representation space of discrete-continuous actions. The selected latent action is then decoded into the original space so as to interact with the environment.

In this paper, we propose a novel framework for hybrid action RL, called **Hy**brid **A**ction **R**epresentation (**HyAR**), to achieve all three properties in Table 1. A conceptual overview of HyAR is shown in Fig. 1. The main idea is to construct a unified and decodable representation space for original discrete-continuous hybrid actions, among which the agent learns a latent policy. Then, the selected latent action is decoded back to the original hybrid action space so as to interact with the environment. HyAR is inspired by recent advances in Representation Learning in DRL. Action representation learning has shown the potentials in boosting learning performance (Whitney et al., 2020), reducing large discrete action space (Chandak et al., 2019), improving generalization in offline RL (Zhou et al., 2020) and so on. Different from these works, to the best knowledge, we are the first to propose representation learning for discrete-continuous hybrid actions, which consist of heterogeneous and dependent action components.

In HyAR, we maintain a continuous vector for each discrete action in a learnable embedding table; then a conditional Variational Auto-encoder (VAE) (Kingma & Welling, 2014) that conditions on the state and the embedding of discrete action is used to construct the latent representation space for the associated continuous parameters. Different from HHQN, the conditional VAE models and embeds the dependence in an implicit fashion. The

Algorithm	Scalability	Stationarity	Dependence	Latent
PADDPG	✗	✓	✗	✗
HPPO	✗	✓	✗	✗
PDQN	✗	✓	✓	✗
HHQN	✓	✗	✓	✗
HyAR (Ours)	✓	✓	✓	✓

Table 1: A comparison on algorithmic properties of existing methods for discrete-continous hybrid action RL.

learned representation space are compact and thus scalable, while also provides convenient decoding by nearest-neighbor lookup of the embedding table and the VAE decoder. Moreover, we utilize the unsupervised environmental dynamics to learn dynamics predictive hybrid action representation. Such a representation space can be semantically smooth, i.e., hybrid action representations that are close in the space have similar influence on environmental dynamics, thus benefits hybrid action RL in the representation space. With the constructed action representation space, we use TD3 algorithm (Fujimoto et al., 2018)for the latent policy learning. To ensure the effectiveness, we further propose two mechanisms: *latent space constraint* and *representation shift correction* to deal with unreliable latent representations and outdated off-policy action representation experiences respectively. In our experiments, we evaluate HyAR in a few representative environments with hybrid action space, as well as several new and more challenging benchmarks.

Our main contributions are summarized below:

- We propose a novel and generic framework for discrete-continuous hybrid action RL by leveraging representation learning of hybrid action space for the first time.
- We propose an unsupervised method of learning a compact and decodable representation space for discrete-continuous hybrid actions, along with two mechanisms to improve the effectiveness of latent policy learning.
- Our algorithm consistently outperforms prior algorithms in representative hybrid-action benchmarks, especially demonstrating significant superiority when the hybrid action space becomes larger.[1]

2 Preliminaries

2.1 Markov Decision Process

Consider a standard Markov Decision Process (MDP) $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, T \rangle$, defined with a state set $\mathcal{S}$, an action set $\mathcal{A}$, transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, discounted factor $\gamma \in [0, 1)$ and horizon T. The agent interacts with the MDP by performing its policy $\pi : \mathcal{S} \to \mathcal{A}$. The objective of an RL agent is to optimize its policy to maximize the expected discounted cumulative reward $J(\pi) = \mathbb{E}_\pi[\sum_{t=0}^{T} \gamma^t r_t]$, where $s_0 \sim \rho_0(s_0)$ the initial state distribution, $a_t \sim \pi(s_t)$, $s_{t+1} \sim \mathcal{P}(s_{t+1} \mid s_t, a_t)$ and $r_t = \mathcal{R}(s_t, a_t)$. The state-action value function Q^π is defined as $Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t r_t \mid s_0 = s, a_0 = a\right]$.

2.2 Parameterized Action MDP

In this paper, we focus on a Parameterized Action Markov Decision Process (PAMDP) $\langle \mathcal{S}, \mathcal{H}, \mathcal{P}, \mathcal{R}, \gamma, T \rangle$ (Masson et al., 2016). PAMDP is an extension of stardard MDP with a discrete-continuous hybrid action space $\mathcal{H}$ defined as:

$$\mathcal{H} = \{(k, x_k) \mid x_k \in \mathcal{X}_k \text{ for all } k \in \mathcal{K}\}, \tag{1}$$

where $\mathcal{K} = \{1, \cdots, K\}$ is the discrete action set, $\mathcal{X}_k$ is the corresponding continuous parameter set for each $k \in \mathcal{K}$. We call any pair of k, x_k as a hybrid action and also call $\mathcal{H}$ as hybrid action space for short in this paper. In turn, we have state transition function $\mathcal{P} : \mathcal{S} \times \mathcal{H} \times \mathcal{S} \to \mathbb{R}$, reward function $\mathcal{R} : \mathcal{S} \times \mathcal{H} \to \mathbb{R}$, agent's policy $\pi : \mathcal{S} \to \mathcal{H}$ and hybrid-action value function $Q^\pi(s, k, x_k)$.

Conventional RL algorithms are not compatible with hybrid action space $\mathcal{H}$. Typical policy representations such as Multinomial distribution or Gaussian distribution can not model the heterogeneous components among the hybrid action. Implicit policies derived by action value functions, often adopted in value-based algorithms, also fail due to intractable maximization over infinite hybrid actions. In addition, there exists the dependence between discrete actions and continuous parameters, as a discrete action k determines the valid parameter space $\mathcal{X}_k$ associated with it. In other words, the same parameter paired with different discrete actions can be significantly different in semantics. This indicate that in principle an optimal hybrid-action policy can not determine the continuous parameters beforehand the discrete action is selected.

[1]Codes are provided in the supplementary material.

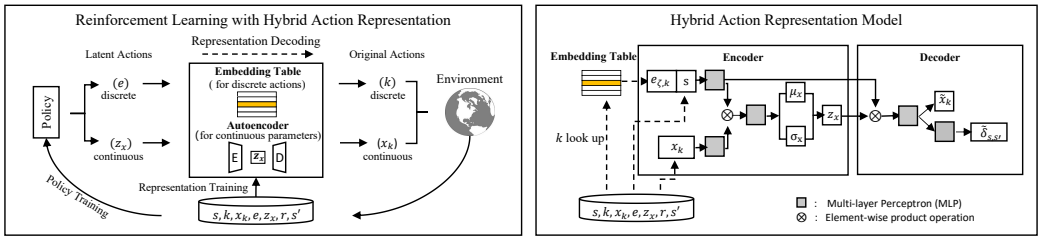


Figure 2: Illustrations of: (*left*) the framework DRL with HyAR; and (*right*) overall workflow of hybrid action representation model, consisting of the discrete action embedding table and condition VAE.

3 Hybrid Action Representation (HyAR)

As mentioned in previous sections, it is non-trivial for an RL agent to learn with discrete-continuous hybrid action space efficiently due to the heterogeneity and action dependence. Naive solutions by converting the hybrid action space into either a discrete or a continuous action space can result in degenerated performance due to the scalability issue and additional approximation difficulty. Previous efforts concentrate on proposing specific policy structures (Hausknecht & Stone, 2016; Fu et al., 2019) that are feasible to learn hybrid-action policies directly over original hybrid action space. However, these methods fail in providing the three desired properties: scalability, stationarity and action dependence simultaneously (See Tab. 1).

Inspired by recent advances in Representation Learning for RL (Whitney et al., 2020; Chandak et al., 2019), we propose Hybrid Action Representation (HyAR), a novel framework that converts the original hybrid-action policy learning into a continuous policy learning problem among the latent action representation space. The intuition behind HyAR is that discrete action and continuous parameter are heterogeneous in their original representations but they jointly influence the environment. Therefore, we can assume that hybrid actions lie on a homogeneous manifold that is closely related to environmental dynamics semantics. In the following of this section, we introduce an unsupervised approach of constructing a compact and decodable latent representation space to approximate such a manifold.

3.1 Dependence-aware Encoding and Decoding

A desired latent representation space for hybrid actions should take the dependence between the two heterogeneous components into account. Moreover, we need the representation space to be decodable, i.e., the latent actions selected by a latent policy can be mapped back to the original hybrid actions so as to interact with the environment. To this end, we propose dependence-aware encoding and decoding of hybrid action. The overall workflow is depicted in the right of Fig. 2. We establish an embedding table $E_\zeta \in \mathbb{R}^{K \times d_1}$ with learnable parameter ζ to represent the K discrete actions, where each row $e_{\zeta,k} = E_\zeta(k)$ (with k being the row index) is a d_1-dimensional continuous vector for the discrete action k. Then, we use a conditional Variational Auto-Encoder (VAE) (Kingma & Welling, 2014) to construct the latent representation space for the continuous parameters. In specific, for a hybrid action k, x_k and a state s, the encoder $q_\phi(z \mid x_k, s, e_{\zeta,k})$ parameterized by ϕ takes s and the embedding $e_{\zeta,k}$ as condition, and maps x_k into the latent variable $z \in \mathbb{R}^{d_2}$. With the same condition, the decoder $p_\psi(\tilde{x}_k \mid z, s, e_{\zeta,k})$ parameterized by ψ then reconstructs the continuous parameter $\tilde{x}_k$ from z. In principle, the conditional VAE can be trained by maximizing the variational lower bound (Kingma & Welling, 2014).

More concretely, we adopt a Gaussian latent distribution $\mathcal{N}(\mu_x, \sigma_x)$ for $q_\phi(z \mid x_k, s, e_{\zeta,k})$ where μ_x, σ_x are the mean and standard deviation outputted by the encoder. For any latent variable sample $z \sim \mathcal{N}(\mu_x, \sigma_x)$, the decoder decodes it deterministicly, i.e., $\tilde{x}_k = p_\psi(z, s, e_{\zeta,k})$. With a batch of states and original hybrid actions from buffer $\mathcal{D}$, we train the embedding table E_ζ and the conditional VAE q_ϕ, p_ψ together by minimizing the loss function L_{VAE} below:

$$L_{\text{VAE}}(\phi, \psi, \zeta) = \mathbb{E}_{s,k,x_k \sim \mathcal{D}, z \sim q_\phi} \Big[\|x_k - \tilde{x}_k\|_2^2 + D_{\text{KL}}\big(q_\phi(\cdot \mid x_k, s, e_{\zeta,k}) \| \mathcal{N}(0, I)\big) \Big], \tag{2}$$

where the first term is the L_2-norm square reconstruction error and the second term is the Kullback-Leibler divergence D_{KL} between the variational posterior of latent representation z and the standard Gaussian prior. Note $\tilde{x}_k$ is differentiable with respect to ψ, ζ and ϕ through reparameterization trick (Kingma & Welling, 2014).

The embedding table and conditional VAE jointly construct a compact and decodable hybrid action representation space ($\in \mathbb{R}^{d_1+d_2}$) for hybrid actions. We highlight that this is often much smaller than the joint action space $\mathbb{R}^{K+\sum_k |\mathcal{X}_k|}$ considered in previous works (e.g., PADDPG, PDQN and HPPO), especially when K or $\sum_k |\mathcal{X}_k|$ is large. In this sense, HyAR is expected to be more scalable when compared in Tab. 1. Moreover, the conditional VAE embeds the dependence of continuous parameter on corresponding discrete action in the latent space; and allows to avoid the redundancy of outputting all continuous parameters at any time (i.e., $\mathbb{R}^{\sum_k |\mathcal{X}_k|}$). This resembles the conditional structure adopted by HHQN (Fu et al., 2019) while HyAR is free of the non-stationary issue thanks to learning a single policy in the hybrid representation space.

For any latent variables $e \in \mathbb{R}^{d_1}$ and $z_x \in \mathbb{R}^{d_2}$, they can be decoded into hybrid action k, x_k conveniently by nearest-neighbor lookup of the embedding table along with the VAE decoder. Formally, we summarize the encoding and decoding process below:

$$\begin{aligned} &\textbf{Encoding:}\ e_{\zeta,k} = E_\zeta(k), \quad z_x \sim q_\phi(\cdot \mid x_k, s, e_{\zeta,k}) && \text{for } s, k, x_k \\ &\textbf{Decoding:}\ k = g_E(e) = \arg\min_{k' \in \mathcal{K}} \|e_{\zeta,k'} - e\|_2, \quad x_k = p_\psi(z_x, s, e_{\zeta,k}) && \text{for } s, e, z_x \end{aligned} \tag{3}$$

3.2 Dynamics Predictive Representation

In the above, we introduce how to construct a compact and decodable latent representation space for original hybrid actions. However, the representation space learned by pure reconstruction of VAE may be pathological in the sense that it is not discriminative to how hybrid actions have different influence on the environment, similarly studied in (Grosnit et al., 2021). Therefore, such a representation space may be ineffective when involved in the learning of a RL policy and value functions, as these functions highly depends on the knowledge of environmental dynamics. To this end, we make full use of environmental dynamics and propose an unsupervised learning loss based on state dynamics prediction to further refine the hybrid action representation.

Intuitively, the dynamics predictive representation learned is semantically smooth. In other words, hybrid action representations that are closer in the space reflects similar influence on environmental dynamics of their corresponding original hybrid actions. Therefore, in principle such a representation space can be superior in the approximation and generalization of RL policy and value functions, than that learned purely from VAE reconstruction. The benefits of dynamics predictive representation are also demonstrated in (Whitney et al., 2020) (Schwarzer et al., 2020).

As shown in the right of Fig. 2, HyAR adopts a subnetwork that is cascaded after the main body of the conditional VAE decoder (called *cascaded structure* or *cascaded head* below) to produce the prediction of the state residual of transition dynamics. For any transition sample (s, k, x_k, s'), the state residual is denoted by $\delta_{s,s'} = s' - s$. With some abuse[2], the prediction $\tilde{\delta}_{s,s'}$ is produced as follows, which completes Eq. 3:

$$\textbf{Prediction:}\ \tilde{\delta}_{s,s'} = p_\psi(z_x, s, e_{\zeta,k}) \qquad \text{for } s, e, z_x \tag{4}$$

Then we minimize the L_2-norm square prediction error:

$$L_{\text{Dyn}}(\phi, \psi, \zeta) = \mathbb{E}_{s,k,x_k,s'} \left[\|\tilde{\delta}_{s,s'} - \delta_{s,s'}\|_2^2 \right]. \tag{5}$$

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

So far, we derive the ultimate training loss for hybrid action representation as follows:

$$L_{\text{HyAR}}(\phi, \psi, \zeta) = L_{\text{VAE}}(\phi, \psi, \zeta) + \beta L_{\text{Dyn}}(\phi, \psi, \zeta), \tag{6}$$

[2] We use ψ to denote the parameters of both the decoder network and prediction network which share the first layer for the convenience. Strictly speaking, it is not proper to also call the prediction network as decoder.

Algorithm 1: HyAR-TD3

1 Initialize actor π_ω and critic networks $Q_{\theta_1}, Q_{\theta_2}$ with random parameters $\omega, \theta_1, \theta_2$
2 Initialize discrete action embedding table E_ζ and conditional VAE q_ϕ, p_ψ with random parameters ζ, ϕ, ψ
3 Prepare replay buffer $\mathcal{D}$
4 **repeat** Stage ❶
5 | Update ζ and ϕ, ψ using samples in $\mathcal{D}$ ▷ see Eq. 6
6 **until** *reaching maximum warm-up training times*;
7 **repeat** Stage ❷
8 | **for** $t \leftarrow 1$ *to* T **do**
9 | | // select latent actions in representation space
10 | | $e, z_x = \pi_\omega(s) + \epsilon_{\text{e}}$, with $\epsilon_{\text{e}} \sim \mathcal{N}(0, \sigma)$
11 | | // decode into original hybrid actions
12 | | $k = g_E(e), x_k = p_\psi(z_x, s, e_{\zeta,k})$ ▷ see Eq. 3
13 | | Execute (k, x_k), observe r_t and new state s'
14 | | Store $\{s, k, x_k, e, z_x, r, s'\}$ in $\mathcal{D}$
15 | | Sample a mini-batch of N experience from $\mathcal{D}$
16 | | Update $Q_{\theta_1}, Q_{\theta_2}$ ▷ see Eq. 7
17 | | Update π_ω with policy gradient ▷ see Eq. 8
18 | **repeat**
19 | | Update ζ and ϕ, ψ using samples in $\mathcal{D}$ ▷ see Eq. 6
20 | **until** *reaching maximum representation training times*;
21 **until** *reaching maximum total environment steps*;

where β is a hyper-parameter that weights the dynamics predictive representation loss. Note that the ultimate loss depends on reward-agnostic data of environmental dynamics, which is dense and usually more convenient to obtain (Stooke et al., 2021; Yarats et al., 2021; Erraqabi et al., 2021).

4 DRL WITH HYBRID ACTION REPRESENTATION

In previous section, we introduce the construction of a compact, decodable and semantically smooth hybrid action representation space. As the conceptual overview in Fig. 1, the next thing is to learn a latent RL policy in the representation space. In principle, our framework is algorithm-agnostic and any RL algorithms for continuous control can be used for implementation. In this paper, we adopt model-free DRL algorithm TD3 (Fujimoto et al., 2018) for demonstration. Though there remains the chance to build a world model based on hybrid action representation, we leave the study on model-based RL with HyAR for future work.

TD3 is popular deterministic-policy Actor-Critic algorithm which is widely demonstrated to be effective in continuous control. As illustrated in the left of Fig. 2, with the learned hybrid action representation space, the actor network parameterizes a latent policy π_ω with parameter ω that outputs the latent action vector, i.e., $e, z_x = \pi_\omega(s)$ where $e \in \mathbb{R}^{d_1}, z_x \in \mathbb{R}^{d_2}$. The latent action can be decoded according to Eq. 3 and obtain the corresponding hybrid action k, x_k. The double critic networks $Q_{\theta_1}, Q_{\theta_2}$ take as input the latent action to approximate hybrid-action value function Q^{π_ω}, i.e., $Q_{\theta_{i=1,2}}(s, e, z_x) \approx Q^{\pi_\omega}(s, k, x_k)$. With a buffer of collected transition sample (s, e, z_x, r, s'), the critics are trained by Clipped Double Q-Learning, with the loss function below for $i = 1, 2$:

$$L_{\text{CDQ}}(\theta_i) = \mathbb{E}_{s,e,z_x,r,s'}\left[\left(y - Q_{\theta_i}(s, e, z_x)\right)^2\right], \quad \text{where} \;\; y = r + \gamma \min_{j=1,2} Q_{\bar{\theta}_j}\left(s', \pi_{\bar{\omega}}(s')\right), \tag{7}$$

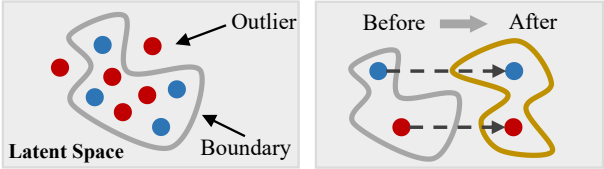
where $\bar{\theta}_{j=1,2}, \bar{\omega}$ are the target network parameters. The actor (latent policy) is updated with Deterministic Policy Gradient (Silver et al., 2014) as follows:

$$\nabla_\omega J(\omega) = \mathbb{E}_s\left[\nabla_{\pi_\omega(s)} Q_{\theta_1}(s, \pi_\omega(s)) \nabla_\omega \pi_\omega(s)\right]. \tag{8}$$

Algorithm 1 describes the pseudo-code of HyAR-TD3, containing two major stages: ❶ ***warm-up stage*** and ❷ ***training stage***. In the warm-up stage, the hybrid action representation models are pre-trained using a prepared replay buffer $\mathcal{D}$ (line 4-6). The parameters the embedding table and conditional VAE is updated by minimizing the VAE and dynamics prediction loss. Note that the proposed algorithm has no requirement on how the buffer $\mathcal{D}$ is prepared and here we simply use a random policy for the environment interaction and data generation by default. In the learning

stage, given a environment state, the latent policy outputs a latent action perturbed by a Gaussian exploration noise, with some abuse of notions e, z_x (line 10). The latent action is decoded into original hybrid action so as to interact with the environment, after which the collected transition sample is stored in the replay buffer (line 12-14). Then, the latent policy learning is preformed using the data sampled from $\mathcal{D}$ (line 15-17). It is worth noting that the action representation model is updated concurrently in the training stage to make continual adjustment to the change of data distribution (line 19-21).

One significant distinction of DRL with HyAR described above compared with conventional DRL is that, the hybrid action representation space is learned from finite samples that are drawn from a moving data distribution. The induced *unreliability* and *shift* of learned representations can severely cripple the performance of learned latent policy if they are not carefully handled. Hence, we propose two mechanisms to resolve the above two considerations as detailed below.



(a) Representation unreliability (b) Representation shift

Figure 3: Illustrations of representation unreliability and representation shift. Dots denote the hybrid action representations selected by policy (red) and known by encoder (blue). The gray line forms a area, among which representations can be well decoded and estimated.

Latent Space Constraint (LSC) As the latent representation space is constructed by finite hybrid action samples, some areas in the latent space can be highly unreliable in decoding as well as Q-value estimation. Similar evidences are also founded in (Zhou et al., 2020; Notin et al., 2021). In Fig. 3(a), the latent action representations inside the boundary can be well decoded and estimated the values, while the outliers cannot. Once the latent policy outputs outliers, which can be common in the early learning stage, the unreliability can quickly deteriorate the policy and lead to bad results. Therefore, we propose to constrain the action representation space of the latent policy inside a reasonable area adaptively. In specific, we re-scale the output of latent policy (i.e., $[-1, 1]^{d_1+d_2}$ by tanh activation) to a bounded range $[b_{\text{lower}}, b_{\text{upper}}]$. For a number of s, k, x_k collected previously, the bounds $b_{\text{lower}}, b_{\text{upper}}$ are obtained by calculating the c-percentage central range where $c \in [0, 100]$. We empirically demonstrate the importance of LSC. See more details in Appendix A & C.3.

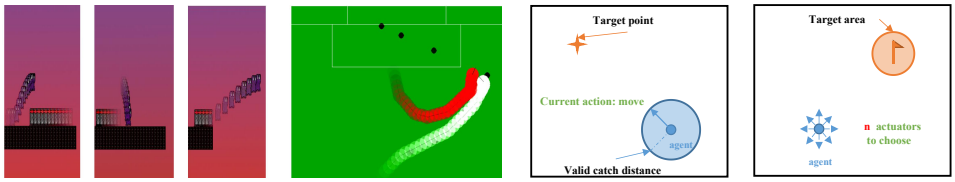
Representation Shift Correction (RSC) As in Algorithm 1, the hybrid action representation space is continuously optimized along with RL process. Thus, the representation distribution of original hybrid actions in the latent space can shift after a certain learning interval (Igl et al., 2020). Fig. 3(b) illustrates the shift (denoted by different shapes). This negatively influences the value function learning since the outdated latent action representation no longer reflects the same transition at present. To handle this, we propose a representation relabeling mechanism. In specific, for each mini-batch training in Eq.7, we check the semantic validity of hybrid action representations in current representation space and relabel the invalid ones with the latest representations. In this way, the policy learning is always performed on latest representations, so that the issue of representation shift can be alleviate. Empirically evaluations demonstrate the superiority of relabeling techniques in achieving a better performance with a lower variance. See more details in Appendix A & C.3.

5 Experiments

We evaluate HyAR in various hybrid action environments against representative prior algorithms. Then, a detailed ablation study is conducted to verify the contribution of each component in HyAR. Moreover, we provide visual analysis for better understandings of HyAR.

5.1 Experiment Setups

Benchmarks Fig. 4 visualizes the evaluation benchmarks, including the *Platform* and *Goal* from (Masson et al., 2016), *Catch Point* from (Fan et al., 2019), and a newly designed *Hard Move* specific to the evaluation in larger hybrid action space. We also build a complex version of Goal, called *Hard Goal*. All benchmarks have hybrid actions and require the agent to select reasonable actions to complete the task. See complete description of benchmarks in Appendix B.1.



(a) Platform (b) Goal (c) Catch Point (d) Hard Move

Figure 4: Benchmarks with discrete-continuous actions: (a) the agent selects a discrete action (run, hop, leap) and the corresponding continuous parameter (horizontal displacement) to reach the goal; (b) The agent selects a discrete strategy (move, shoot) and the continuous 2-D coordinate to score; (c) The agent selects a discrete action (move, catch) and the continuous parameter (direction) to grab the target point; (d) The agent has n equally spaced actuators. It can choose whether each actuator should be on or off (thus 2^n combination in total) and determine the corresponding continuous parameter for each actuator (moving distance) to reach the target area.

ENV	HPPO	PADDPG	PDQN	HHQN	HyAR-DDPG	PATD3	PDQN-TD3	HHQN-TD3	HyAR-TD3
	PPO-based	DDPG-based				TD3-based			
Goal	0.0 ± 0.0	0.05 ± 0.10	0.70 ± 0.07	0.0 ± 0.0	0.53 ± 0.02	0.0 ± 0.0	0.71 ± 0.10	0.0 ± 0.0	**0.78 ± 0.03**
Hard Goal	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.30 ± 0.08	0.44 ± 0.05	0.06 ± 0.07	0.01 ± 0.01	**0.60 ± 0.07**
Platform	0.80 ± 0.02	0.36 ± 0.06	0.93 ± 0.05	0.46 ± 0.25	0.87 ± 0.06	0.94 ± 0.10	0.93 ± 0.03	0.62 ± 0.23	**0.98 ± 0.01**
Catch Point	0.69 ± 0.09	0.82 ± 0.06	0.77 ± 0.07	0.31 ± 0.06	0.89 ± 0.01	0.82 ± 0.10	0.89 ± 0.07	0.27 ± 0.05	**0.90 ± 0.03**
Hard Move (n = 4)	0.09 ± 0.02	0.03 ± 0.01	0.69 ± 0.07	0.39 ± 0.14	0.91 ± 0.03	0.66 ± 0.13	0.85 ± 0.10	0.52 ± 0.17	**0.93 ± 0.02**
Hard Move (n = 6)	0.05 ± 0.01	0.04 ± 0.01	0.41 ± 0.05	0.32 ± 0.17	0.91 ± 0.04	0.04 ± 0.02	0.74 ± 0.08	0.29 ± 0.13	**0.92 ± 0.04**
Hard Move (n = 8)	0.04 ± 0.01	0.06 ± 0.03	0.04 ± 0.01	0.05 ± 0.02	0.85 ± 0.06	0.06 ± 0.02	0.05 ± 0.01	0.05 ± 0.02	**0.89 ± 0.03**
Hard Move (n = 10)	0.05 ± 0.01	0.04 ± 0.01	0.06 ± 0.02	0.04 ± 0.01	**0.82 ± 0.06**	0.07 ± 0.02	0.05 ± 0.02	0.05 ± 0.02	0.75 ± 0.05

Table 2: Comparisons of the baselines regarding the average performance at the end of training process with the corresponding standard deviation over 5 runs. Values in bold indicate the best results in each environment.

Baselines Four state-of-the-art approaches are selected as baselines: HPPO (Fan et al., 2019), PDQN (Xiong et al., 2018), PADDPG (Hausknecht & Stone, 2016), HHQN (Fu et al., 2019). In addition, for a comprehensive study, we extend the baselines which consists of DDPG to their TD3 variants, denoted by PDQN-TD3, PATD3, HHQN-TD3. Last, we use HyAR-DDPG and HyAR-TD3 to denote our implementations of DRL with HyAR based on DDPG and TD3. For a fair comparison, the network architecture (i.e., DDPG and TD3) used in associated baselines are the same. For all experiments, we give each baseline the same training budget. For our algorithms, we use a random strategy to interact with the environment for 5000 episodes during the warm-up stage. For each experiment, we run 5 trials and report the average results. Complete details of setups are provided in Appendix B.

5.2 Performance Evaluation

To conduct a comprehensive comparison, all baselines implemented based on either DDPG or TD3 are reported. To counteract implementation bias, codes of PADDPG, PDQN, and HHQN are directly adopted from prior works. Comparisons in terms of the averaged results are summarized in Tab. 2, where bold numbers indicate the best result. Overall, we have the following findings. HyAR-TD3 and HyAR-DDPG show the better results and lower variance than the others. Moreover, the advantage of HyAR is more obvious in environments in larger hybrid action space (e.g., Hard Goal & Hard Move). Taking Hard Move for example, as the action space grows exponentially, the performance of HyAR is steady and barely degrades, while the others deteriorate rapidly. Similar results can be found in Goal and Hard Goal environments. This is due to the superiority of HyAR of utilizing the hybrid action representation space, among which the latent policy can be learned based on compact semantics. These results not only reveal the effectiveness of HyAR in achieving better performance, but also the scalability and generalization.

In almost all environments, HyAR outperforms other baselines for both the DDPG-based and TD3-based cases. The exceptions are in Goal and Platform environments, where PDQN performs slightly better than HyAR-DDPG. We hypothesize that this is because the hybrid action space of these two environments is relatively small. For such environments, the learned latent action space could be sparse and noisy, which in turn degrades the performance. One evidence is that the conservative (underestimation) nature in TD3 could compensate and alleviates this issue, achieving significant

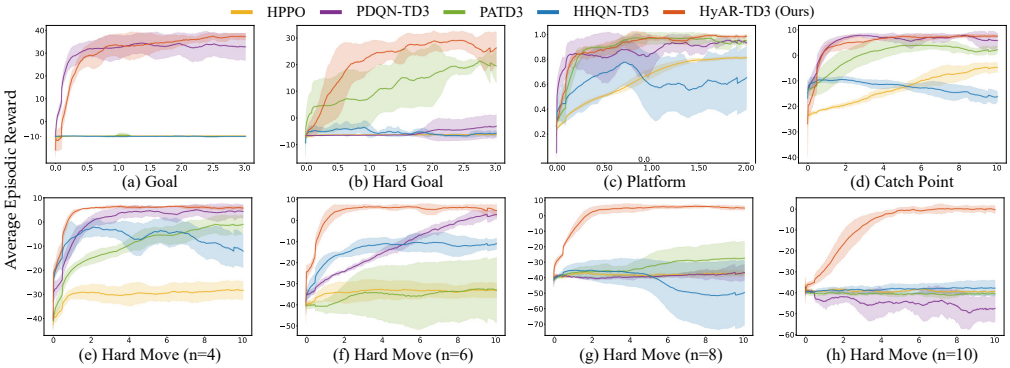


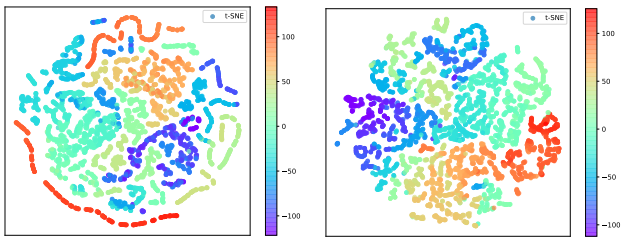
Figure 5: Comparisons of algorithms in different environments. The x- and y-axis denote the environment steps ($\times 10^5$) and average episode reward over recent 100 episodes. The curve and shade denote the mean and a standard deviation over 5 runs.

improvements (HyAR-TD3 v.s. HyAR-DDPG). Fig. 5 renders the learning curves, where HyAR-TD3 outperforms other baselines in both the final performance and learning speed across all environments. Similar results are observed in DDPG-based comparisons and can be found in Appendix C.1. In addition, HyAR-TD3 shows good generalization across environments, while the others more or less fail in some environments (e.g., HPPO, PATD3, and HHQN-TD3 fail in Fig. 5(a) and PDQN-TD3 fails in Fig. 5(b)). Moreover, when environments become complex (Fig. 5(e-h)), HyAR-TD3 still achieves steady and better performance, particularly demonstrating the effectiveness of HyAR in high-dimensional hybrid action space.

5.3 Ablation Study and Visual Analysis

We further evaluate the contribution of the major components in HyAR: the two mechanisms for latent policy learning, i.e., latent space constraint (LSC) and representation shift correction (RSC), and the dynamics predictive representation loss. We briefly conclude our results as follows. For LSC, properly constraining the output space of the latent policy is critical to performance; otherwise, both loose and conservative constraints dramatically lead to performance degradation. RSC and dynamics predictive representation loss show similar efficacy: they improve both learning speed and convergence results, additionally with a lower variance. Such superiority is more significant in the environment when hybrid actions are more semantically different (e.g., Goal). We also conduct ablation studies on other factors along with hyperparameter analysis. See complete details and ablation results in Appendix C.2 & C.3.

Finally, we adopt t-SNE (Maaten & Hinton, 2008) to visualize the learned hybrid action representations, i.e., (e, z_x), in a 2D plane. We color each action based on its impact on the environment i.e., $\tilde{\delta}_{s,s'}$. As shown in Fig. 6, we observe that actions with a similar impact on the environment are relatively closer in the latent space. This demonstrates the dynamics predictive representation loss is helpful for deriving dynamics-aware representation for further improving the learning performance, efficacy, and stability (see results in Appendix C.2 & C.4)



(a) Goal (b) Hard Move (n = 8)

Figure 6: 2D t-SNE visualizations of learned representation for original hybrid actions, colored by 1D t-SNE of the corresponding environmental impact.

6 Conclusion

In this paper, we propose Hybrid Action Representation (HyAR) for DRL agents to efficiently learn with discrete-continuous hybrid action space. HyAR use an unsupervised method to derive a com-

pact and decodable representation space for discrete-continuous hybrid actions. HyAR can be easily extended with modern DRL methods to leverage additional advantages. Our experiments demonstrate the superiority of HyAR regarding performance, learning speed and robustness in most hybrid action environment, especially in high-dimensional action spaces.

REFERENCES

M. Azabou, M. G. Azar, R. Liu, C. H. Lin, E. C. Johnson, K. B. Nair, M. D., K. B. Hengen, W. G. Roncal, M. V., and E. Dyer. Mine your own view: Self-supervised learning through across-sample prediction. *CoRR*, abs/2102.10106, 2021.

Y. Chandak, G. Theocharous, J. Kostas, S. M. Jordan, and P. S. Thomas. Learning action representations for reinforcement learning. In *ICML*, volume 97, pp. 941–950, 2019.

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

Z. Fan, R. Su, W. Zhang, and Y. Yu. Hybrid actor-critic reinforcement learning in parameterized action space. *IJCAI*, pages2279-2285, 2019.

H. Fu, H. Tang, J. Hao, Z. Lei, Y. Chen, and C. Fan. Deep multi-agent reinforcement learning with discrete-continuous hybrid action spaces. *IJCAI*, pages2329-2335, 2019.

S. Fujimoto, H. v. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *ICML*, volume 80, pp. 1582–1591, 2018.

A. Grosnit, R. Tutunov, A. Maraval, R. Griffiths, A. Cowen-Rivers, L. Yang, L. Zhu, W. Lyu, Z. Chen, J. Wang, J. Peters, and H. Bou-Ammar. High-dimensional bayesian optimisation with variational autoencoders and deep metric learning. *CoRR*, abs/2106.03609, 2021.

M. Hausknecht and P. Stone. Deep reinforcement learning in parameterized action space. *ICLR*, 2016.

M. Igl, G. Farquhar, J. Luketina, W. Boehmer, and S. Whiteson. The impact of non-stationarity on generalisation in deep reinforcement learning. *CoRR*, abs/2006.05826, 2020.

D. P. Kingma and M. Welling. Auto-encoding variational Bayes. In *ICLR*, 2014.

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

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

L. V. D. Maaten and G. E. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

W. Masson, P. Ranchod, and G. D. Konidaris. Reinforcement learning with parameterized actions. In *AAAI*, pp. 1934–1940, 2016.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

P. Notin, J. M. Hernández-Lobato, and Y. Gal. Improving black-box optimization in VAE latent space using decoder uncertainty. *CoRR*, abs/2107.00096, 2021.

J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz. Trust region policy optimization. In *ICML*, pp. 1889–1897, 2015.

J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. In *ICLR*, 2016.

M. Schwarzer, A. Anand, R. Goel, R. D. Hjelm, A. C. Courville, and P. Bachman. Data-efficient reinforcement learning with momentum predictive representations. *CoRR*, abs/2007.05929, 2020.

D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. A. Riedmiller. Deterministic policy gradient algorithms. In *ICML*, pp. 387–395, 2014.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

A. Stooke, K. Lee, P. Abbeel, and M. Laskin. Decoupling representation learning from reinforcement learning. In *ICML*, volume 139, pp. 9870–9879, 2021.

H. Tang, Z. Meng, G. Chen, P. Chen, C. Chen, Y. Yang, L. Zhang, W. Liu, and J. Hao. Foresee then evaluate: Decomposing value estimation with latent future prediction. In *AAAI*, pp. 9834–9842, 2021.

R. Wang, R. Yu, B. An, and Z. Rabinovich. I^2hrl: Interactive influence-based hierarchical reinforcement learning. In *IJCAI*, pp. 3131–3138, 2020.

W. F. Whitney, R. Agarwal, K. Cho, and A. Gupta. Dynamics-aware embeddings. In *ICLR*, 2020.

J. Xiong, Q. Wang, Z. Yang, P. Sun, L. Han, Y. Zheng, H. Fu, T. Zhang, J. Liu, and H. Liu. Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space. *CoRR*, abs/1810.06394, 2018.

D. Yarats, R. Fergus, A. Lazaric, and L. Pinto. Reinforcement learning with prototypical representations. In *ICML*, volume 139, pp. 11920–11931, 2021.

W. Zhou, S. Bajracharya, and D. Held. PLAS: latent action space for offline reinforcement learning. *CoRR*, abs/2011.07213, 2020.

A DETAILED FOR LATENT SPACE CONSTRAINT (LSC) AND REPRESENTATION SHIFT CORRECTION(RSC)

Latent Space Constraint (LSC) As we can see in Fig. 3(a), the latent action representations inside the boundary can be well decoded and estimated the values, while the outliers cannot. Therefore, the most critical problem for latent space constraint (LSC) is to find a reasonable latent space boundary. Simply re-scale policy's outputs in a fixed bounded area $[-b, b]$ could lose some important information and make the latent space unstable (Zhou et al., 2020; Notin et al., 2021) . We propose a mechanism to constrain the action representation space of the latent policy inside a reasonable area adaptively. In specific, we re-scale the output of latent policy (i.e., $[-1, 1]^{d_1+d_2}$ by tanh activation) to a bounded range $[b_{\text{lower}}, b_{\text{upper}}]$. At intervals (actually concurrent with the updates of the hybrid action representation models), we first sample M transitions s, k, x_k from buffer, then we obtain the corresponding latent action representations with current representation models. In this way, we will get M different latent variable values in each dimension. We sort the latent variable of each dimension, calculate the c-percentage central range and let the lower bound and upper bound of the range to be $[b_{\text{lower}}]$ and $[b_{\text{upper}}]$ of the current latent variable. Note that n control the c-percentage central range where $c \in [0, 100]$, we called latent select range. With the decrease of c , the constained latent action representation space becomes smaller. The experiment on the value of n and latent select range is in Appendix C.3.

Representation Shift Correction (RSC) Since the hybrid action representation space is continuously optimized along with the RL learning, the representation distribution of original hybrid actions in the latent space can shift after a certain learning interval (Igl et al., 2020). Fig. 3(b) visualizes the shifting (denoted by different shapes). This negatively influences the value function learning since the outdated latent action representation no longer reflects the same transition at present. To handle this, we propose a representation relabeling mechanism. In specific, we feed the batch of stored original hybrid actions to our representation models to obtain the latest latent representations, for each mini-batch training in Eq.7. For latent discrete action e, if it can not be mapped to the corresponding original action k in the latest embedding table, we will relabel e through looking up the table with stored original discrete action k, i.e., $e \leftarrow e_{\zeta,k} + \mathcal{N}(0, 0.1)$. The purpose of adding noise $\mathcal{N}(0, 0.1)$ is to ensure the diversity of the relabeled action representations. For latent continuous action z_x, we first obtain $\tilde{\delta}_{s,s'}$ through the latest decoder $p_\psi(z_x, s, e_{\zeta,k})$. Then we verify if $\|\tilde{\delta}_{s,s'} - \delta_{s,s'}\|_2^2 > \delta_0$ (threshold value δ_0 is set to be $4 * \hat{L}_{\text{Dyn}}$, where $\hat{L}_{\text{Dyn}}$ is the moving empirical loss), i.e., the case that indicates that the historical representations has no longer semantically consistent (with respect to environmental dynamics) under current representation models. Then z_x will be relabeled by the latest latent representations $z \sim q_\phi(\cdot \mid x_k, s, e_{\zeta,k})$. In this way, the policy learning is always performed on latest representations, so that the issue of representation distribution shift can be effectively alleviated. The experiment on relabeling techniques is in Appendix C.3.

Limitation and Potential Improvements of LSC and RSC The ideas of LSC and RSC are general. In our work, both LSC and RSC are easy to implement and empirically demonstrated to be effective. We also consider that there remains some room to further improve these two mechanisms in some more challenging cases. For LSC, one potential drawback may be that the central range constraint on per dimension adopted in our work, is suboptimal to the case if the distribution of learned action representation of original hybrid actions is multi-modal. In such cases, the central range constraint may be less effective and can be further improved. For RSC, it requires additional memory and computation cost. However, in our experiments, such memory and computation cost is relatively negligible. The dimensionality of each original hybrid action sample is relatively small compared to the state sample stored; the additional wall-clock time cost of HyAR (with RSC) to HyAR without RSC is about 6% (tested in *Goal*).

B EXPERIMENTAL DETAILS

B.1 SETUPS

Our codes are implemented with Python 3.7.9 and Torch 1.7.1. All experiments were run on a single NVIDIA GeForce GTX 2080Ti GPU. Each single training trial ranges from 4 hours to 10 hours, depending on the algorithms and environments. For more details of our code can refer to the `HyAR.zip` in the supplementary results.

Layer	Actor Network ($\pi(s)$)	Critic Network ($Q(s,a)$ or $V(s)$)
Fully Connected	(state dim, 256)	(state dim + $\mathbb{R}^{K+\sum_k \lvert\mathcal{X}_k\rvert}$, 256) or (state dim + $\mathbb{R}^{\sum_k \lvert\mathcal{X}_k\rvert}$, 256) or (state dim, 256)
Activation	ReLU	ReLU
Fully Connected	(256, 256)	(256, 256)
Activation	ReLU	ReLU
Fully Connected	(256, $\mathbb{R}^K$) and (256, $\mathbb{R}^{\sum_k \lvert\mathcal{X}_k\rvert}$) or (256, $\mathbb{R}^{\sum_k \lvert\mathcal{X}_k\rvert}$)	(256, 1) or (256, $\mathbb{R}^K$)
Activation	Tanh	None

Table 3: Network structures for the actor network and the critic network (Q-network or V-network).

Benchmark Environments We conduct our experiments on several hybrid action environments and detailed experiment description is below.

- ***Platform*** (Masson et al., 2016): The agent need to reach the final goal while avoiding the enemy or falling into the gap. The agent need to select the discrete action (run, hop, leap) and determine the corresponding continuous action (horizontal displacement) simultaneously to complete the task. The horizon of an episode is 20.
- ***Goal*** (Masson et al., 2016): The agent shoots the ball into the gate to win. Three types of hybrid actions are available to the agent including *kick-to(x,y), shoot-goal-left(h), shoot-goal-right(h)*. The continuous action parameters position (x, y) and position (h) along the goal line are quit different. Furthermore, We built a complex version of the goal environment, called ***Hard Goal***. We redefined the shot-goal action and split it into ten parameterized actions by dividing the goal line equidistantly. The continuous action parameters of each shot action will be mapped to a region in the goal line. The horizon of an episode is 50.
- ***Catch Point*** (Fan et al., 2019): The agent should catch the target point (orange) in limited opportunity (10 chances). There are two hybrid actions *move* and *catch*. Move is parameterized by a continuous action value which is a directional variable and catch is to try to catch the target point. The horizon of an episode is 20.
- ***Hard Move (designed by us)***: The agent needs to control n equally spaced actuators to reach target area (orange). Agent can choose whether each actuator should be on or off. Thus, the size of the action set is exponential in the number of actuators that is $\mathbf{2^n}$. Each actuator controls the moving distance in its own direction. n controls the scale of the action space. As n increases, the dimension of the action will increase. The horizon of an episode is 25.

B.2 Network Structure

Our PATD3 is implemented with reference to `github.com/sfujim/TD3` (TD3 source-code). PADDPG and PDQN are implemented with reference to `https://github.com/cycraig/MP-DQN`. For a fair comparison, all the baseline methods have the same network structure (except for the specific components to each algorithm) as our HyAR-TD3 implementation. For PDQN, PADDPG, we introduce a Passthrough Layer (Masson et al., 2016) to the actor networks to initialise their action-parameter policies to the same linear combination of state variables. HPPO paper does not provide open source-code and thus we implemented it by ourselves according to the guidance provided in their paper. For HPPO, the discrete actor and continuous actor do not share parameters (better than share parameters in our experiments).

As shown in Tab. 3, we use a two-layer feed-forward neural network of 256 and 256 hidden units with ReLU activation (except for the output layer) for the actor network for all algorithms. For PADDPG, PDQN and HHQN, the critic denotes the Q-network. For HPPO, the critic denotes the V-network. Some algorithms (PATD3, PADDPG, HHQN) output two heads at the last layer of the actor network, one for discrete action and another for continuous action parameters.

The structure of HyAR is shown in Tab. 4. We introduced element-wise product operation (Tang et al., 2021) and cascaded head structure (Azabou et al., 2021) to our HyAR model. More details about their effects are in Appendix C.3.

Model Component	Layer (Name)	Structure
Discrete Action Embedding Table E_ζ	Parameterized Table	$(\mathbb{R}^{d_1}, \mathbb{R}^K)$
Conditional Encoder Network	Fully Connected (encoding)	$(\mathbb{R}^{\mathcal{X}_k}, 256)$
$q_\phi(z \mid x_k, s, e_{\zeta,k})$	Fully Connected (condition)	(state dim + $\mathbb{R}^{d_1}$, 256)
	Element-wise Product	ReLU (encoding) · ReLU(condition)
	Fully Connected	(256, 256)
	Activation	ReLU
	Fully Connected (mean)	(256, $\mathbb{R}^{d_2}$)
	Activation	None
	Fully Connected (log_std)	(256, $\mathbb{R}^{d_2}$)
	Activation	None
Conditional Decoder Network	Fully Connected (latent)	$(\mathbb{R}^{d_2}, 256)$
$p_\psi(x_k \mid z_k, s, e_{\zeta,k})$	Fully Connected (condition)	(state dim + $\mathbb{R}^{d_1}$, 256)
	Element-wise Product	ReLU(decoding) · ReLU(condition)
	Fully Connected	(256, 256)
	Activation	ReLU
	Fully Connected (reconstruction)	(256, $\mathbb{R}^{\mathcal{X}_k}$)
	Activation	None
	Fully Connected	(256, 256)
	Activation	ReLU
	Fully Connected (prediction)	(256, state dim)
	Activation	None

Table 4: Network structures for the hybrid action representation (HyAR) including, the discrete action embedding table and the conditional VAE.

B.3 Hyperparameter

For all our experiments, we use the raw state and reward from the environment and no normalization or scaling are used. No regularization is used for the actor and the critic in all algorithms. An exploration noise sampled from $N(0, 0.1)$ (Fujimoto et al., 2018) is added to all baseline methods when select action. The discounted factor is 0.99 and we use Adam Optimizer (Kingma & Ba, 2015) for all algorithms. Tab. 5 shows the common hyperparamters of algorithms used in all our experiments.

Hyperparameter	HPPO	PADDPG	PDQN	HHQN	PATD3	PDQN-TD3	HHQN-TD3	HyAR-DDPG	HyAR-TD3
Actor Learning Rate	$1{\cdot}10^{-4}$	$1{\cdot}10^{-4}$	$1{\cdot}10^{-4}$	$1{\cdot}10^{-4}$	$3{\cdot}10^{-4}$	$3{\cdot}10^{-4}$	$3{\cdot}10^{-4}$	$1{\cdot}10^{-4}$	$3{\cdot}10^{-4}$
Critic Learning Rate	$1{\cdot}10^{-3}$	$1{\cdot}10^{-3}$	$1{\cdot}10^{-3}$	$1{\cdot}10^{-3}$	$3{\cdot}10^{-4}$	$3{\cdot}10^{-4}$	$3{\cdot}10^{-4}$	$1{\cdot}10^{-3}$	$3{\cdot}10^{-4}$
Representation Model Learning Rate	-	-	-	-	-	-	-	$1{\cdot}10^{-4}$	$1{\cdot}10^{-4}$
Discount Factor	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
Optimizer	Adam	Adam	Adam	Adam	Adam	Adam	Adam	Adam	Adam
Target Actor Update Rate	-	$1{\cdot}10^{-3}$	$1{\cdot}10^{-3}$	$1{\cdot}10^{-3}$	$5{\cdot}10^{-3}$	$5{\cdot}10^{-3}$	$5{\cdot}10^{-3}$	$1{\cdot}10^{-3}$	$5{\cdot}10^{-3}$
Target Critic Update Rate	-	$1{\cdot}10^{-2}$	$1{\cdot}10^{-2}$	$1{\cdot}10^{-2}$	$5{\cdot}10^{-3}$	$5{\cdot}10^{-3}$	$5{\cdot}10^{-3}$	$5{\cdot}10^{-3}$	$5{\cdot}10^{-3}$
Exploration Policy	$\mathcal{N}(0, 0.1)$	$\mathcal{N}(0, 0.1)$	$\mathcal{N}(0, 0.1)$	$\mathcal{N}(0, 0.1)$	$\mathcal{N}(0, 0.1)$	$\mathcal{N}(0, 0.1)$	$\mathcal{N}(0, 0.1)$	$\mathcal{N}(0, 0.1)$	$\mathcal{N}(0, 0.1)$
Batch Size	128	128	128	128	128	128	128	128	128
Buffer Size	10^5	10^5	10^5	10^5	10^5	10^5	10^5	10^5	10^5
Actor Epoch	2	-	-	-	-	-	-	-	-
Critic Epoch	10	-	-	-	-	-	-	-	-

Table 5: A comparison of common hyperparameter choices of algorithms.We use '-' to denote the 'not applicable' situation.

B.4 Additional Implementation Details

Training setup: For HPPO, the actor network and the critic network are updated every 2 and 10 episodes respectively for all environment. The clip range of HPPO algorithm is set to 0.2 and we use GAE (Schulman et al., 2016) for stable policy gradient. For DDPG-based, the actor network and the critic network is updated every 1 environment step. For TD3-based, the critic network is updated every 1 environment step and the actor network is updated every 2 environment step.

Algorithm 2: HyAR-DDPG

Initialize actor π_ω and critic networks Q_θ with random parameters ω, θ, and the corresponding target network parameters $\bar{\omega}, \bar{\theta}$
Initialize discrete action embedding table E_ζ and conditional VAE q_ϕ, p_ψ with random parameters ζ, ϕ, ψ
Prepare replay buffer $\mathcal{D}$
repeat Stage ❶
  Update ζ and ϕ, ψ using samples in $\mathcal{D}$ ▷ see Eq. 6
until *reaching maximum warm-up training times*;
repeat Stage ❷
  for $t \leftarrow$ *1 to* T **do**
    // select latent actions in representation space
    $e, z_x = \pi_\omega(s) + \epsilon_e$, with $\epsilon_e \sim \mathcal{N}(0, \sigma)$
    // decode into original hybrid actions
    $k = g_E(e), x_k = p_\psi(z_x, s, e_{\zeta,k})$ ▷ see Eq. 3
    Execute (k, x_k), observe r_t and new state s'
    Store $\{s, k, x_k, e, z_x, r, s'\}$ in $\mathcal{D}$
    Sample a mini-batch B of N experience from $\mathcal{D}$
    Update critic by minimizing empirical loss $\hat{L}_Q(\theta) = N^{-1} \sum_B (y - Q_\theta(s, e, z_x))^2$, where $y = r + \gamma Q_{\bar{\theta}}(s', \pi_{\bar{\omega}}(s'))$
    Update actor by the deterministic policy gradient $\nabla_\omega J(\omega) = N^{-1} \sum_{s \in B} \left[\nabla_{\pi_\omega(s)} Q_\theta(s, \pi_\omega(s)) \nabla_\omega \pi_\omega(s) \right]$.
  repeat
    Update ζ and ϕ, ψ using samples in $\mathcal{D}$ ▷ see Eq. 6
  until *reaching maximum representation training times*;
until *reaching maximum total environment steps*;

For the warm-up stage, we run 5000 episodes (please refer to Tab. 7 for the corresponding environment steps in different environments) for experience collection. We then pre-train the hybrid action representation model (discrete action embedding table and conditional VAE) for 5000 batches with batch size 64, after which we start the training of the latent policy. The discrete action embedding table is initialized randomly by drawing each dimension from the uniform distribution $U(-1, 1)$ before representation pre-training. The representation models (the embedding table and conditional VAE) are trained every 10 episodes for 5 batches with batch size 64 for the rest of RL training. The latent action dim (discrete or continuous latent action) default value is 6. We set the KL weight in representation loss L_{VAE} as 0.5 and dynamics predictive representation loss weight β as 10 (default). More details about dynamics predictive representation loss weight are in C.2.

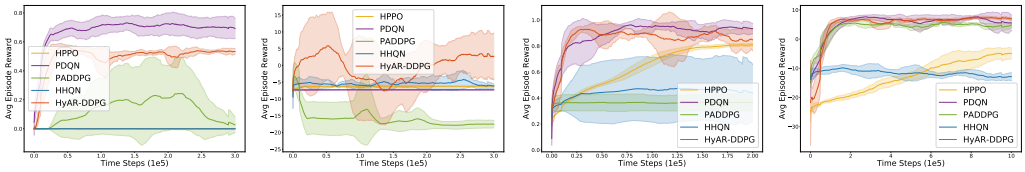
B.5 DDPG-based HyAR algorithm

Additionally, we implemented HyAR with DDPG (Lillicrap et al., 2015), called HyAR-DDPG. The pseudo-code of complete algorithm is shown in Algorithm 2. Results of DDPG-based experimental comparisons can be found in Appendix C.1.

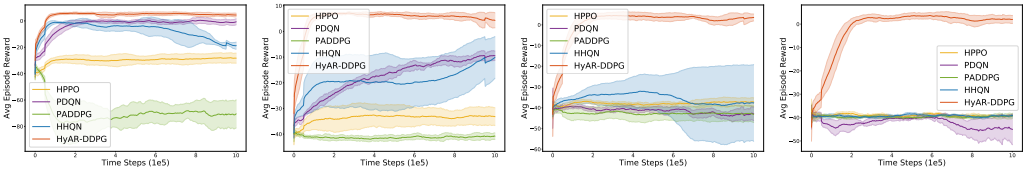
C Complete Learning Curves and Additional Experiments

C.1 Learning Curves for DDPG-based Comparisons

Fig. 7 visualizes the learning curves of DDPG-based comparisons, where HyAR-DDPG outperforms other baselines in both the final performance and learning speed in most environments. Besides the learning speed, HyAR-DDPG also achieves the best generalization as HyAR-TD3 across different environments. When the environments become complex (shown in Fig. 7(e-h)), HyAR-DDPG still achieves steady and better performance than the others, particularly demonstrating the effectiveness and generalization of HyAR in high-dimensional hybrid action spaces.

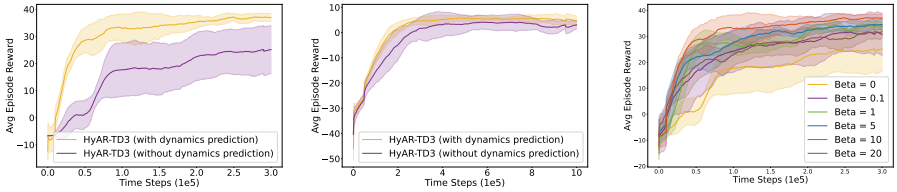


(a) Goal (b) Hard Goal (c) Platform (d) Catch Point



(e) Hard Move (n = 4) (f) Hard Move (n = 6) (g) Hard Move (n = 8) (h) Hard Move (n = 10)

Figure 7: DDPG-based comparisons of related baselines in different environments. The x- and y-axis denote the environment steps ($\times 10^5$) and average episode reward over recent 100 episodes. The results are averaged using 5 runs, while the solid line and shaded represent the mean value and a standard deviation respectively.



(a) Goal (b) Hard Move (n = 8) (c) Goal

Figure 8: Learning curves for variants of dynamics predictive representation in HyAR. (a) and (b) show the comparison between HyAR-TD3 with and without dynamics prediction auxiliary loss; (c) shows the effects of the balancing weight β in Eq.6. The results are averaged using 5 runs, while the solid line and shaded represent the mean value and a standard deviation respectively.

Operation		Structure						Result	
Elem.-Wise Prod.	Concat.	Cascaded	Parallel	Latent Select Range c	Latent Action Dim	Relabeling	Dynamics Predictive	Results (Goal)	Results (Hard Move)
✓		✓		96% (n = 2)	6	✓	✓	**0.78 ± 0.03**	0.89 ± 0.03
	✓	✓		96% (n = 2)	6	✓	✓	0.66± 0.10	0.83± 0.04
✓			✓	96% (n = 2)	6	✓	✓	0.71 ± 0.04	0.80± 0.13
✓		✓		96% (n = 2)	6		✓	0.66 ± 0.07	0.83± 0.08
✓		✓		100% (n = 0)	6	✓	✓	0.62 ± 0.11	0.78± 0.13
✓		✓		90% (n = 5)	6	✓	✓	0.61 ± 0.04	0.78± 0.08
✓		✓		80% (n = 10)	6	✓	✓	0.08 ± 0.17	0.56± 0.12
✓		✓		96% (n = 2)	3	✓	✓	0.59 ± 0.09	0.58± 0.16
✓		✓		96% (n = 2)	12	✓	✓	0.65 ± 0.09	**0.90 ± 0.04**
✓		✓		96% (n = 2)	6	✓		0.55 ± 0.15	0.84 ± 0.05

Table 6: Ablation of our algorithm across each contribution in Goal and Hard Move (n = 8). Results are average success rates at the end of training process over 5 runs. ± corresponds to a standard deviation. The corresponding episode reward learning curves are shown in Fig. 9.

C.2 Learning Curves for the Dynamics Predictive Representation

Fig. 8 shows the learning curves of HyAR-TD3 with dynamics predictive representation loss (Fig. 8(a-b)) and the influence of dynamics predictive representation loss weight β on algorithm performance (Fig. 8(c)). We can easily find that the representation learned by dynamics predictive representation loss is better than without dynamics predictive representation loss. For the weight β of dynamics predictive representation loss, the performance of the algorithm will gradually improve with the increase of weight β, until a certain threshold is reached. We can conclude that the dynamics predictive representation loss is helpful for deriving an environment-awareness representation for further improving the learning performance, efficacy, and stability. More experiments on representation visualization are in Appendix C.4.

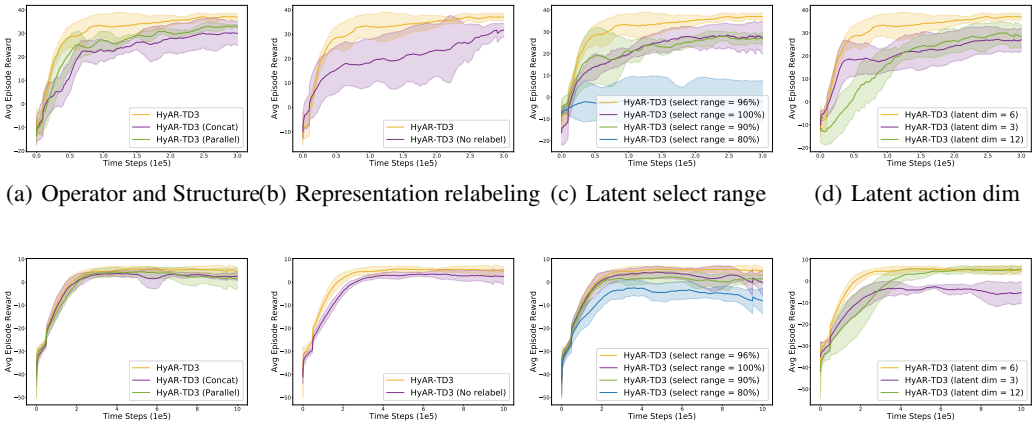
C.3 Learning Curves and Table for the Results in Ablation Study

As briefly discussed in Sec. 5.3, we conduct detailed ablation experiments on the key components of the algorithm, including:

- element-wise product (Tang et al., 2021) (v.s. concat) operation;
- cascaded head (Azabou et al., 2021) (v.s. parallel head) structure;
- latent select range(from 80% to 100%), corresponding to latent space constraint (LSC);
- action representation relabeling, corresponding to representation shift correction (RSC);
- latent action dim (from 3 to 12);

Fig. 9 shows the learning curves of HyAR-TD3 and its variants for ablation studies, corresponding to the results in Tab. 6.

First, we can observe that element-wise product achieves better performance than concatenation (Fig. 9(a,e)). As similarly discovered in (Tang et al., 2021), we hypothesize that the explicit relation between the condition and representation imposed by element wise product forces the conditional VAE to learn more effective hidden features. Second, the significance of cascaded head is demonstrated by its superior performance over parallel head (Fig. 9(a,e)) which means cascaded head can better output two different features. Third, representation relabeling shows an apparent improvement (Fig. 9(b,f)) which show that representation shift leads to data invalidation in the experience buffer which will affect RL training. Fourth, a reasonable latent select range plays an important role in algorithm learning (Fig. 9(c,g)). Only constrain the action representation space of the latent policy inside a reasonable area (both large and small will fail), can the algorithm learn effectively and reliably. These experimental results supports our analysis above. We also analyse the influence of latent action dim for RL (Fig. 9(d,h)). In the low-dimensional hybrid action environment, we should choose a moderate value (e.g., 6). While for high-dimensional environment, larger value may be better (e.g., 12).



(a) Operator and Structure (b) Representation relabeling (c) Latent select range (d) Latent action dim

(e) Operator and Structure (f) Representation relabeling (g) Latent select range (h) Latent action dim

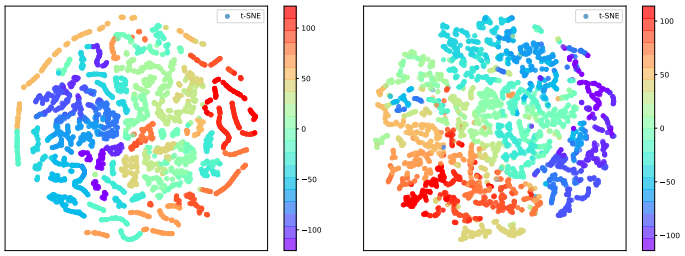
Figure 9: Learning curves of ablation studies for HyAR (i.e., element-wise + cascaded head + representation relabeling + latent select range = 96% + latent action dim = 6) corresponding to Tab. 6. From top to bottom is Goal and Hard Move (n = 8) environment. The shaded region denotes standard deviation of average evaluation over 5 runs.

C.4 Representation Visual Analysis

In order to further analyze the hybrid action representation, we visualize the learned hybrid action representations. Fig. 10 and Fig. 11 shows the t-SNE visualization for HyAR in Goal and Hard Move (n = 8) environment.

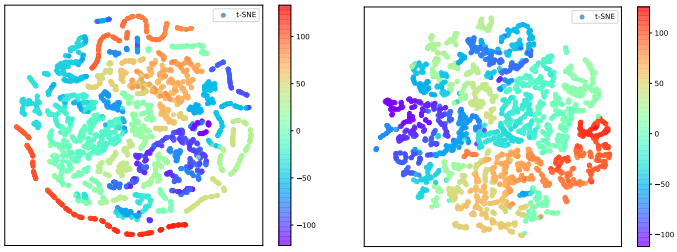
As we can see from Fig. 10, we adopt t-SNE to cluster the latent continuous actions, i.e., (z_x), outputted by the latent policy, and color each action based on latent discrete actions i.e., (e). We can conclude that latent continuous actions can be clustered by latent discrete actions, but there are multiple modes in the global range. Our dependence-aware representation model makes good use of this relationship that the choice of continuous action parameters is depend on discrete actions.

For the dynamics predictive representation loss, we adopt t-SNE to cluster the latent actions, i.e., (e, z_x), outputted by the latent policy, and color each action based on its impact on the environment (i.e., $\delta_{s,s'}$). As shown in Fig. 11, we observe that actions with a similar impact on the environment are relatively closer in the latent space. This demonstrates the dynamics predictive representation loss is helpful for deriving an environment-awareness representation for further improving the learning performance, efficacy, and stability.



(a) Goal (b) Hard Move (n = 8)

Figure 10: t-SNE visualization diagram of continuous action embedding z_x, color coded by discrete action embedding e. The continuous actions related to the same discrete actions are mapped to the similar regions of the representation space.



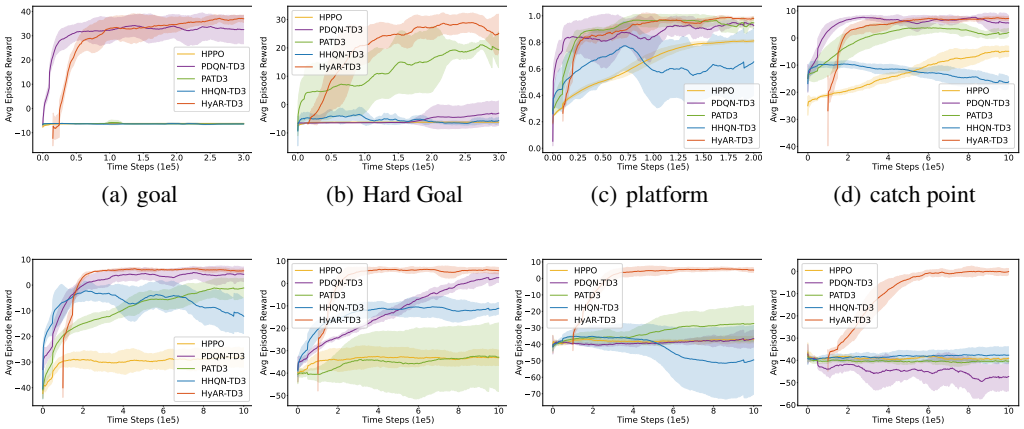
(a) Goal (b) Hard Move (n = 8)

Figure 11: t-SNE visualization diagram of hybrid action embedding pair (e, z_x), color coded by $\tilde{\delta}_{s,s'}$. The hybrid actions with a similar impact on the environment are relatively closer in the latent space.

D Additional Experiments

We provide some additional experimental results in our revision:

- We provide the exact number of samples used in the warm-up stage (i.e., stage 1 in Algorithm 1 in each environment in Tab. 7. The numbers of warm-up environment steps are about $5\%-10\%$ of the total environment steps in our original experiments.
- Moreover, we also conduct some experiments to further reduce the number of samples used in the warm-up stage (at most 80% off). See the colored results in Tab. 7. HyAR can achieve comparable performance with $< 3\%$ samples of the total environment steps.
- We also provide Fig. 12 for another view of the learning curves where the number of samples used in the warm-up stage is also counted for HyAR.
- We conduct additional experiments to compare HyAR-TD3 and HyAR-TD3 with fixed hybrid action representation space trained in the warm-up stage in the environments *Platform, Goal, Hard Goal, Catch Point and Hard Move* ($n = 8$). The results are provided in Fig. 13, demonstrating the necessity of subsequent training of the representation trained in the warm-up stage.



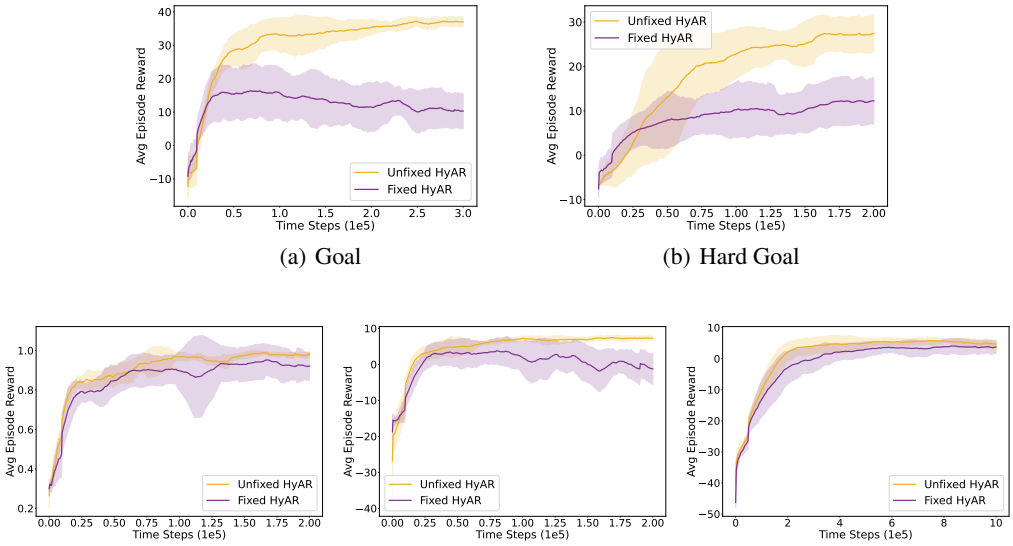
(a) goal (b) Hard Goal (c) platform (d) catch point

(e) Hard Move (n=4) (f) Hard Move (n=6) (g) Hard Move (n=8) (h) Hard Move (n=10)

Figure 12: Another view of the learning curves shown in Fig. 5 where the number of samples used in warm-up training (i.e., stage 1 in Algorithm 1) is also counted for HyAR. Comparisons of algorithms in different environments. The x- and y-axis denote the environment steps and average episode reward over recent 100 episodes. The results are averaged using 5 runs, while the solid line and shaded represent the mean value and a standard deviation respectively.

Environment	Number of Warm-up Env. Steps *original*	*new*	Number of Total Env. Steps
Goal	20000 (0.067|0.78)	5000 (0.017|0.75)	300000
Hard Goal	20000 (0.067|0.60)	5000 (0.017|0.55)	300000
Platform	10000 (0.05|0.98)	5000 (0.025|0.96)	200000
Catch Point	100000 (0.1|0.90)	20000 (0.02|0.82)	1000000
Hard Move (n=4)	100000 (0.1|0.93)	20000 (0.02|0.91)	1000000
Hard Move (n=6)	100000 (0.1|0.92)	20000 (0.02|0.92)	1000000
Hard Move (n=8)	100000 (0.1|0.89)	20000 (0.02|0.83)	1000000
Hard Move (n=10)	100000 (0.1|0.75)	20000 (0.02|0.70)	1000000

Table 7: The exact number of samples used in warm-up stage training in different environments. The column of 'original' denotes what is done in our experiments; the column of 'new' denotes additional experiments we conduct with fewer warm-up samples (and proportionally fewer warm-up training). For each entry $x(\mathbf{y}|\mathbf{z})$, x is the number of samples (environment steps), $\mathbf{y}$ denotes the percentage $\frac{\text{number of warm-up environment steps}}{\text{number of total environment steps during the training process}}$, and $\mathbf{z}$ denotes the corresponding performance of HyAR-TD3 as evaluated in Tab. 2. **Conclusion:** The numbers of warm-up environment steps are about $\mathbf{5\%-10\%}$ of the total environment steps in our original experiments. The number of warm-up environment steps can be further reduced by at most 80% off (thus leading to $<$ **3%** of the total environment steps) while comparable performance of our algorithm remains.



(a) Goal (b) Hard Goal

(c) Platform (d) Catch Point (e) Hard Move(n=8)

Figure 13: Comparison between HyAR-TD3 (denoted by **Unfixed HyAR**) and HyAR-TD3 with fixed hybrid action representation space trained in warm-up stage (denoted by **Fixed HyAR**) in different environments. The x- and y-axis denote the environment steps and average episode reward over recent 100 episodes. The results are averaged using 5 runs, while the solid line and shaded represent the mean value and a standard deviation respectively. **Conclusion:** Unfixed HyAR outperforms Fixed HyAR across all the environments; while Fixed HyAR performs very poorly in *Goal* and *Hard Goal*. We conjecture that in these environments, random policy is quite limited in collecting effective and meaning hybrid actions in these environments thus the learned fixed representation space is not able to support the emergence of effective latent policy.