

SELECT BEFORE ACT: SPATIALLY DECOUPLED ACTION REPETITION FOR CONTINUOUS CONTROL

Buqing Nie¹, Yangqing Fu¹, Yue Gao^{1,2*}

¹MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University

²Shanghai Innovation Institute, Shanghai, P.R. China

{niebuqing, frank79110, yuegao}@sjtu.edu.cn

ABSTRACT

Reinforcement Learning (RL) has achieved remarkable success in various continuous control tasks, such as robot manipulation and locomotion. Different to mainstream RL which makes decisions at individual steps, recent studies have incorporated action repetition into RL, achieving enhanced action persistence with improved sample efficiency and superior performance. However, existing methods treat all action dimensions as a whole during repetition, ignoring variations among them. This constraint leads to inflexibility in decisions, which reduces policy agility with inferior effectiveness. In this work, we propose a novel repetition framework called **SDAR**, which implements **Spatially Decoupled Action Repetition** through performing closed-loop *act-or-repeat* selection for each action dimension individually. SDAR achieves more flexible repetition strategies, leading to an improved balance between action *persistence* and *diversity*. Compared to existing repetition frameworks, SDAR is more sample-efficient with higher policy performance and reduced action fluctuation. Experiments are conducted on various continuous control scenarios, demonstrating the effectiveness of spatially decoupled repetition design proposed in this work.

1 INTRODUCTION

Recently, Deep Reinforcement Learning (DRL) (Sutton & Barto, 2018) has achieved remarkable success in continuous control domains, such as robot manipulation (Gu et al., 2017), locomotion (Lee et al., 2020b; Zhang et al., 2024), and autonomous driving (Kiran et al., 2021). Although conventional Reinforcement Learning (RL) algorithms have demonstrated significant potential across various applications, they typically make decisions at individual time steps, neglecting higher-level decision-making mechanisms and the temporal consistency of action sequences (Silver et al., 2014; Schulman et al., 2017; Haarnoja et al., 2018). This leads to inefficient exploration during training, challenging credit assignment tasks over long horizons, and poor sample efficiency (Dabney et al., 2021; Yu et al., 2021; Biedenkapp et al., 2021; Zhang et al., 2022; Lee et al., 2024).

One mainstream solution is Hierarchical RL (HRL) based on temporal abstraction (Sutton et al., 1999; Precup, 2000), where the high-level policy decomposes the task into simpler subgoals, and low-level policies are designed to solve isolated subtasks (Vezhnevets et al., 2017; Pateria et al., 2021). However, most HRL methods are task specific, which requires expert knowledge and hand-craft design given different applications, such as pre-defining options in the option framework (Bacon et al., 2017; Nachum et al., 2018; Zhang et al., 2021). A simple yet effective strategy for low-level policies involves executing an action repeatedly for a number of steps, which has been actively explored and proven to be effective in various applications (Lakshminarayanan et al., 2017; Sharma et al., 2017; Dabney et al., 2021; Lee et al., 2024). Action repetition methods improve *action persistence* through *act-or-repeat* selections, making exploration trajectories more temporally consistent, leading to higher training efficiency and reduced fluctuations (Chen et al., 2021; Yu et al., 2021; Biedenkapp et al., 2021).

*Corresponding author.

However, as illustrated in Fig. 1, existing action-repetition approaches treat all action dimensions as a whole during *act-or-repeat* selection, disregarding differences among them. This constrains all action dimensions to repeat previous actions or make new decisions simultaneously at each step. This design reduces the effectiveness of action repetition, because different dimensions may be required to select repetition at different time steps, which is quite common in continuous control tasks (Kalyanakrishnan et al., 2021). For example, robotic systems typically consist of multiple controllers operating at different frequencies due to their system specification, corresponding to different repetition schema for each actuator (Raman et al., 2017; Lee et al., 2020a).

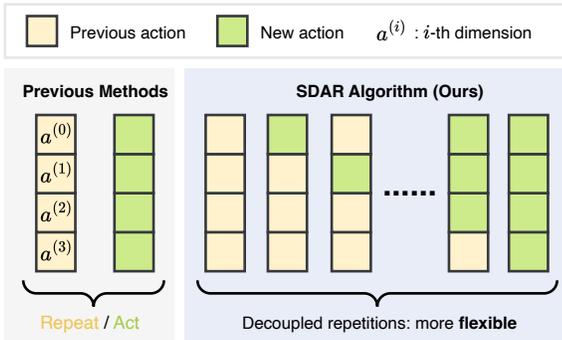


Figure 1: Difference between repetition strategies of previous methods (left) and SDAR (right). Our method achieves a more flexible strategy through the spatially decoupled repetition design.

To address this issue, we propose **Spatially Decoupled Action Repetition (SDAR)**, a new flexible action repetition framework for continuous control tasks. SDAR conducts closed-loop repetition for each action dimension individually, which is composed of two stages: *selection* and *action*. During *selection*, the agent selects whether the previous actions should be repeated for each dimension. Afterwards, the agent generates new decisions for dimensions where *act* is chosen in the previous stage. As shown in Fig. 1, different to previous methods, *act-or-repeat* selections in SDAR are performed in a decoupled manner, leading to more flexible repetition strategies. Compared to existing repetition frameworks, SDAR achieves an improved balance between action *persistence* and *diversity*, resulting in improved efficiency and superior performance with reduced action fluctuation.

The main contributions of this work are summarized as follows:

- We propose a novel action repetition framework called **SDAR** for continuous control tasks. SDAR implements **Spatially Decoupled Action Repetition** through performing closed-loop *act-or-repeat* selection for each action dimension individually.
- Compared to previous repetition methods, SDAR offers more flexible repetition strategies, leading to improved balance between action *persistence* and *diversity*. This results in a higher sample efficiency with superior performance and reduced action fluctuation simultaneously.
- Experiments are conducted on various continuous control tasks, demonstrating the superior performance of our method on training efficiency and final performance. This demonstrates the effectiveness of the spatially decoupled framework for action repetition.

2 RELATED WORK

2.1 TEMPORAL ABSTRACTION

Temporal Abstraction is proposed in the semi-MDP formulation (Sutton et al., 1999; Precup, 2000) and commonly implemented based on the *options* framework (Stolle & Precup, 2002; Bacon et al., 2017; Harutyunyan et al., 2018). Each option describes a low-level policy and is defined as $\langle \mathcal{I}, \pi, \beta \rangle$, where \mathcal{I} denotes the admissible states for the option initialization, π is the policy that the option follows, and β determines when the option is terminated. High-level policies are trained to solve tasks utilizing temporally extended actions provided in the options, rather than one-step actions without action persistence. Plenty of Hierarchical RL methods are proposed based on temporal abstraction, achieving faster exploration and higher sample efficiency in various sequential decision tasks (Lin et al., 2021; Yang et al., 2021). Some works are proposed to learn to design options through various techniques, including discovering state connectedness (Chaganty et al., 2012), replay buffer analysis (Eysenbach et al., 2019), and learning termination criteria (Vezhnevets et al., 2016; Harutyunyan et al., 2019). However, designing options is still a challenging task, which requires prior knowledge and handcraft tuning (Pateria et al., 2021; Yu et al., 2021; Lee et al., 2024).

2.2 ACTION REPETITION

One simple option strategy is repeating a primitive action for a number of steps, which is similar to the frame-skipping utilized in RL for video games (Bellemare et al., 2013; Braylan et al., 2015). Recently, action repetition has been actively researched and widely adopted in RL, which can achieve deeper exploration (Dabney et al., 2021), improve sample efficiency by reducing control granularity (Biedenkapp et al., 2021), and reduce action oscillations (Chen et al., 2021). Existing repetition works are classified as two categories: *open-loop* and *closed-loop* manners. Open-loop methods force the agent to repeat actions for a predicted number of steps without opportunity of early terminations, such as DAR (Lakshminarayanan et al., 2017), FiGAR (Sharma et al., 2017), TempoRL (Biedenkapp et al., 2021), and UTE (Lee et al., 2024). In contrast, closed-loop methods conduct *act-or-repeat* binary decision to decide if the previous action should be repeated, such as PIC (Chen et al., 2021) and TAAC (Yu et al., 2021). Compared to open-loop methods, closed-loop methods examine whether to repeat based on the current state, which is more flexible and improves performance in emergency situations. In this work, we propose a new closed-loop method to conduct act-or-repeat selections for each actuator individually given current state, which is more flexible and achieves higher action persistence with sufficient action diversity.

3 PRELIMINARIES

3.1 PROBLEM FORMULATION

In this work, we focus on model-free RL with continuous action space. The interaction process in RL is formulated as a Markov Decision Process (MDP), denoted as a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, where \mathcal{S} is the state space, \mathcal{A} is the action space, $P(s'|s, a)$ is the transition probability of the environment, $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ denotes the reward function, and $\gamma \in [0, 1)$ denotes the discount factor. The agent takes actions according to its policy, i.e. $a \sim \pi(\cdot|s)$. Our objective is to find a policy π^* that maximizes the expected discounted return, i.e. $\pi^* = \arg \max_{\pi} \mathbb{E}_{a_t \sim \pi, s_{t+1} \sim P} [\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$.

3.2 MODEL-FREE RL FOR CONTINUOUS CONTROL

In order to conduct policy evaluation, we define state-action value function $Q(s_t, a_t) = R(s_t, a_t) + \gamma \mathbb{E}_{\pi, P} [\sum_{t'=t}^{\infty} \gamma^{t'} R(s_{t'}, a_{t'})]$ as the discounted return starting from s_t , given that a_t is taken and then π is followed. The value function $V(s_t) = \mathbb{E}_{a_t \sim \pi} [Q(s_t, a_t)]$ denotes the discounted return starting from s_t following π . Typically, both Q and V functions are modeled as neural networks, which are optimized using the Mean Square Error (MSE) loss, with target values obtained based on the Bellman equation. Take Soft Actor Critic (SAC) (Haarnoja et al., 2018) as an example, the policy is trained through the entropy augmented objective:

$$\pi' \leftarrow \arg \max_{\pi} \mathbb{E}_{s \sim \mathcal{D}, a \sim \pi} [Q(s, a) - \alpha \log \pi(a|s)], \quad (1)$$

where \mathcal{D} denotes the replay buffer, α is the temperature parameter, and Q represents accumulated discounted reward augmented by the entropy.

4 METHODOLOGY

4.1 TWO-STAGE POLICY: SELECTION AND ACTION

As described in Fig. 2, the decision process of SDAR is composed of two stages. **(1) Selection:** choose which action dimensions to change previous decisions utilizing the selection policy β . **(2) Action:** generate new actions for action dimensions that choose *act* in the previous stage utilizing action policy π . Both two stages are described as follows.

4.1.1 STAGE 1: SELECTION POLICY

As illustrated in the gray region of Fig. 2, given current state s and the action a^- at the previous step, we conduct *act-or-repeat* selection for each action dimension individually. Formally, $\beta(b|s, a^-) \in$

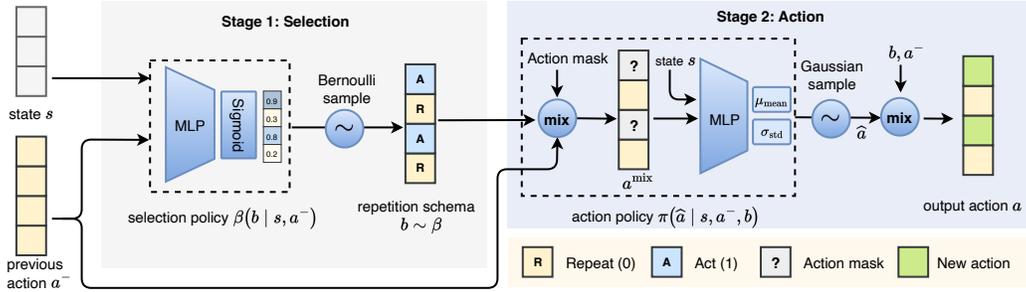


Figure 2: The two-stage decision process of SDAR algorithm. In the first stage (gray region), the selection policy β makes *act-or-repeat* decision for each action dimension, determining whether the previous action a^- (yellow blocks) should be repeated. In the second stage (blue region), the action policy π generates new actions (green blocks) for dimensions that choose *act* in the first stage.

$[0, 1]^{|A|}$, where β_i denotes the probability of choosing *act* in the i -th action dimension. Afterwards, the repetition schema $b \in \{0, 1\}^{|A|}$ at the current state is obtained by sampling on the Bernoulli distribution, i.e. $b_i \sim \beta_i(\cdot | s, a^-)$, where $b_i = 0$ and $b_i = 1$ indicate *repetition* and *action* of the i -th action dimension correspondingly.

4.1.2 STAGE 2: ACTION POLICY

As shown in the blue region of Fig. 2, the agent generates new actions for dimensions that choose *act* based on the current state s , the previous action a^- , and the repetition schema b . Firstly, we reduce redundant input dimensions through introducing the following Mix operation:

$$a^{\text{mix}} = \text{Mix}(b, a^-, \xi) = (1 - b) \odot a^- + b \odot \xi, \quad (2)$$

where \odot denotes element-wise hadamard product and ξ denotes the action mask, which is a constant mask vector filled with meaningless values. In this work, $\xi_i = -2$ for $1 \leq i \leq |A|$ is utilized with $A = [-1, 1]^{|A|}$. This operation eliminates the information of previous actions a^- at dimensions where *act* was chosen in the previous stage. This information is redundant and may disrupt the training process of π . Afterwards, we compute \hat{a} utilizing action policy π :

$$\mu_{\text{mean}}, \sigma_{\text{std}} = \text{MLP}(s, a^{\text{mix}}), \quad \hat{a} = \mu_{\text{mean}} + \sigma_{\text{std}} \cdot n, \quad n \sim \mathcal{N}(0, \mathcal{I}), \quad (3)$$

where μ_{mean} and σ_{std} are the mean and standard derivation of the action distribution predicted by action policy π . The reparameterization trick is utilized to ensure that this process remains differentiable (Hafner et al., 2018). Finally, we obtain the output action a through the Mix operation formulated as follows:

$$a = \text{Mix}(b, a^-, \hat{a}) = (1 - b) \odot a^- + b \odot \hat{a}. \quad (4)$$

This procedure guarantees that the final decision a replicates the same actions as the previous action a^- in dimensions where *repetition* is selected, i.e. $(a - a^-) \odot (1 - b) = 0$. The detailed proof is straightforward and can be found in Appendix A.

Above all, the decision of the whole two-stage policy π^{all} can be described as follows:

$$\pi^{\text{all}}(a | s, a^-) = \sum_b \beta(b | s, a^-) \int_{\hat{a}} \pi(\hat{a} | s, a^-, b) \cdot \delta(\text{Mix}(b, a^-, \hat{a}) - a) d\hat{a}. \quad (5)$$

4.2 POLICY EVALUATION

In this work, we employ two identical Q functions to evaluate the performance of π^{all} . Each Q is trained utilizing Mean Square Error (MSE) loss with targets obtained based on the Bellman operator \mathcal{T} formulated as follows:

$$\min \mathbb{E}_{(s,a) \sim \mathcal{D}} [Q(s, a) - \mathcal{T}Q(s, a)]^2, \quad \text{with } \mathcal{T}Q(s, a) = R(s, a) + \gamma \mathbb{E}_{P, \beta, \pi} [Q(s', a')], \quad (6)$$

Algorithm 1 Spatially Decoupled Action Repetition (SDAR) Algorithm

```

1: Initialize: Selection policy  $\beta$ , action policy  $\pi$ , buffer  $\mathcal{D}$ , critics  $Q_{\{1,2\}}, Q_{\text{targ},\{1,2\}}$ .
2: for each environment step  $t$  do
3:   if a new episode starts then
4:      $b = [1, 1, \dots, 1]_{|A|}$ 
5:   else
6:      $b \sim \beta(\cdot|s_t, a_{t-1})$ 
7:   end if
8:   Obtain  $a_t$  through action policy, i.e. Eq. (2)-(4)
9:   Execute  $a_t$  and observe  $s_{t+1}$  with reward  $r_t$ .
10:  Store transition  $(s_t, a_{t-1}, a_t, r_t, s_{t+1}, d_t)$  into buffer  $\mathcal{D}$ 
11:  Sample a batch of data  $\{(s, a^-, a, r, s', d)\}$  from  $\mathcal{D}$ 
12:  Compute  $a'$  for state  $s'$  using  $\beta$  and  $\pi$ .
13:  Update  $Q_{\{1,2\}}$  through Eq. (6)
14:  if  $t \bmod \text{policy\_delay}$  then
15:    Update  $\theta^\pi$  through Eq. (7)
16:    Update  $\theta^\beta$  through Eq. (8) or Eq. (9)
17:    Update  $\alpha_\beta$  and  $\alpha_\pi$  through Eq. (10)
18:    Update target networks  $Q_{\text{targ},\{1,2\}}$  based on soft updates.
19:  end if
20: end for

```

where \mathcal{D} is the replay buffer. \mathcal{T} in this work is similar to that in standard off-policy RL algorithms (Silver et al., 2014; Lillicrap, 2015), which is demonstrated to converge to the optimal Q after sufficient iterations. In addition, we formulate two Q functions based on neural networks, and train using clipped double-Q learning to alleviate the overestimation problem (Fujimoto et al., 2018). More details are given in Algorithm 1.

4.3 POLICY IMPROVEMENT

We propose to optimize the policy β and π through maximizing the objective $J(\theta^\beta, \theta^\pi)$ formulated as follows, where θ^β and θ^π are learnable parameters of β and π correspondingly.

$$J(\theta^\beta, \theta^\pi) = \underbrace{\mathbb{E}_{(s, a^-) \sim \mathcal{D}} \mathbb{E}_{b \sim \beta, \hat{a} \sim \pi}}_{\text{make decisions on samples}} \left[\underbrace{Q(s, a)}_{\text{max } Q \text{ values}} \underbrace{- \alpha_\beta \log \beta(b|s, a^-) - \alpha_\pi \log \pi(\hat{a}|s, a^-, b)}_{\text{entropy-based exploration}} \right], \quad (7)$$

where entropy terms $\mathbb{E}[-\log \beta(b|s, a^-)]$ and $\mathbb{E}[-\log \pi(\hat{a}|s, a^-, b)]$ are utilized to encourage exploration during training, which is widely utilized in prior works (Haarnoja et al., 2018; Yu et al., 2021). α_β and α_π are temperature parameters to adjust exploration strategy, both of which are tuned automatically during training. As illustrated in Sec. 4.1 and Sec. 4.2, the computation process of π and Q functions are both differentiable. Therefore, the action policy π can be directly optimized using gradient descent with $\nabla_{\theta^\pi} J$.

In order to optimize selection policy β , we can transform objective J into the following formulations:

$$\max_{\theta^\beta} \mathbb{E}_{(s, a^-) \sim \mathcal{D}} \sum_{b \in \mathcal{B}} \beta(b|s, a^-) \mathbb{E}_{\hat{a} \sim \pi} [Q(s, a) - \alpha_\beta \log \beta(b|s, a^-) - \alpha_\pi \log \pi(\hat{a}|s, a^-, b)], \quad (8)$$

where $\mathcal{B} = \{0, 1\}^{|A|}$ is the action space for selection policy β . In contrast to Eq. (7), the new objective in Eq. (8) substitutes the expectation operator $\mathbb{E}_{b \sim \beta}$ with the summation operator $\sum_{b \in \mathcal{B}}$ by listing all possible situations $b \in \mathcal{B}$. The new objective is differentiable for θ^β , thus can be used to optimize β through gradient descent directly.

However, Eq. (8) is quite computationally expensive, because we are required to compute the score for all $b \in \mathcal{B}$, which needs to calculate Q and π for $|\mathcal{B}| = 2^{|A|}$ times. Thus, this method is only practical for tasks with small action spaces, such as LunarLander with $|A| = 2$. For tasks with large action spaces such as Humanoid ($|A| = 17$), the selection policy β can be optimized by sampling

several $b \in \mathcal{B}$, which is formulated as follows:

$$\max_{\theta^\beta} \mathbb{E}_{\mathcal{D}} \mathbb{E}_{b \sim \beta_{\text{old}}, \hat{a} \sim \pi} \left[Q(s, a) - \alpha_\beta \log \beta_{\text{old}}(b|s, a^-) - \alpha_\pi \log \pi(\hat{a}|s, a^-, b) \right] \cdot \frac{\beta(b|s, a^-)}{\beta_{\text{old}}(b|s, a^-)}, \quad (9)$$

where β_{old} denotes the old selection policy before optimization. This objective finds the expectation value $\mathbb{E}_{b \sim \beta}[\cdot]$ in Eq. (7) utilizing importance sampling. Compared to uniform sampling, sampling with β_{old} increases the probability of sampling b with high Q values during training, leading to higher sampling efficiency and training stability. As illustrated in Eq. (9), we are only required to compute Q and π given several b sampled from β , thus is more computationally efficient and practical than Eq. (8) in applications.

In addition, temperature parameters α_β and α_π are tuned automatically following the objective:

$$\min_{\log(\alpha_\beta), \log \alpha_\pi} \mathbb{E}_{\beta, \pi} \left[-\log(\alpha_\beta) (\log \beta(b|s, a^-) + \mathcal{H}_\beta) - \log \alpha_\pi (\log \pi(\hat{a}|s, a^-, b) + \mathcal{H}_\pi) \right], \quad (10)$$

where \mathcal{H}_β and \mathcal{H}_π are target entropies. During training, $\log \alpha_\beta$ and $\log \alpha_\pi$ are updated automatically through solving Eq. (10) with gradient descent, which is similar to prior works (Haarnoja et al., 2018). The detailed settings of the target entropies are described in Appendix B.1.

The whole training process of SDAR is illustrated in Algorithm 1, which iteratively collects data, trains critics, and updates policies. As illustrated in line 10, different to typical DRL algorithms, the previous action a_{t-1} is stored in the buffer \mathcal{D} , which is utilized to improve θ^π and θ^β described in line 15-16. Besides, the agent is forced to choose act at the initial step of an episode, i.e. $b = [1, \dots, 1]_{|A|}$, because a^- is absent at this state.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Tasks: In this work, we conduct experiments on multiple continuous control tasks, which are categorized into the following three types of scenarios. More details are given in Appendix B.2.

- (a) **Classic Control:** Several control tasks with small observation and action spaces, including *MountainCarContinuous*, *LunarLanderContinuous*, and *BipedalWalker*.
- (b) **Locomotion:** Locomotion tasks based on the MuJoCo (Todorov et al., 2012) simulation environment: *Walker2d*, *Hopper*, *HalfCheetah*, *Humanoid*, and *Ant*.
- (c) **Manipulation** tasks including *Pusher*, *Reacher*, and *FetchReach*.

Baseline Methods. In this experiment, we compare the performance of SDAR with the following baseline methods: (a) **SAC:** vanilla Soft Actor Critic (Haarnoja et al., 2018) algorithm; (b) **N-Rep:** repeat policy actions for n times, where n is a *fixed* number (Mnih et al., 2015); (c) **TempoRL:** repeat policy actions dynamically in an *open-loop* manner, using a *skip-policy* to predict repetition steps (Biedenkapp et al., 2021); (d) **UTE:** Uncertainty-aware Temporal Extension (Lee et al., 2024) enhances training efficiency through incorporating uncertainty estimation of repeated action values into the *open-loop* repetition framework. (e) **TAAC:** Temporally Abstract Actor-Critic (Yu et al., 2021), which repeats actions dynamically in a *closed-loop* manner. TAAC determines whether to repeat the previous action at each step utilizing a switch policy. More details of the baseline methods are illustrated in Appendix B.3.

This study trains each method on various tasks using multiple random seeds over a range of 100K to 3M steps, depending on the complexity of the task. More settings including hyperparameters settings are described in Appendix B.1.

5.2 EXPERIMENT RESULTS ON SAMPLE EFFICIENCY

The training curves are illustrated in Fig. 3, and the AUC scores are shown in Table 1. The AUC scores are normalized into $[0, 1]$ to evaluate the sample efficiency across different tasks, where 0.0

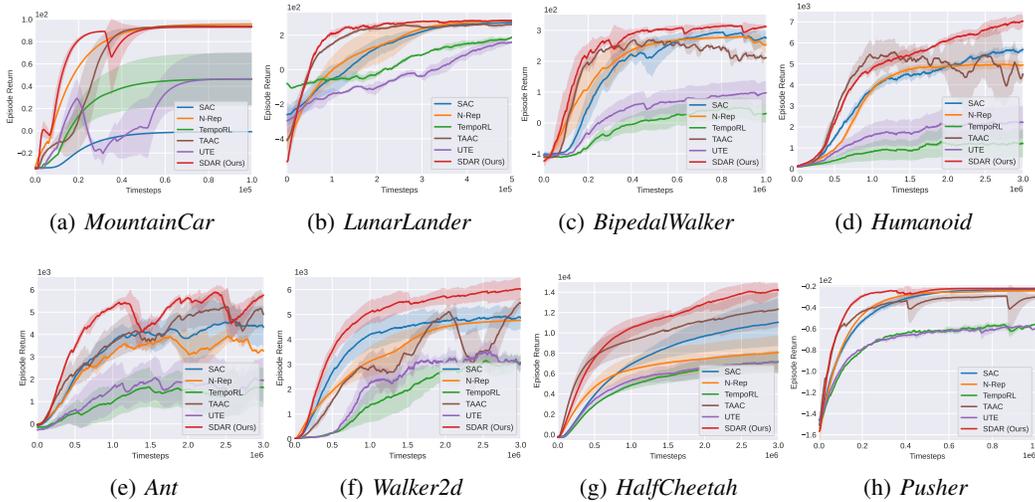


Figure 3: Learning curves of SDAR (red) in various tasks against baseline methods. Each method is trained with at least 10 random seeds. The lines denote the mean episode return, while shaded regions denote the standard error during training. As shown in the figures, our method generally achieves higher sample efficiency in various tasks compared to previous methods. More learning curves are given in Appendix. D.

Table 1: Experiment results on AUC scores (area under the learning curves) in different types of environments, where higher AUC scores indicate higher sample efficiency of corresponding DRL algorithms. The average scores along with the standard errors are calculated across various tasks within the respective category. The results are normalized to evaluate performance across different tasks, where 1.0 denotes the best performance. The best results are boldfaced.

| Env. Category | Normalized AUC Score | | | | | |
|-----------------|----------------------|-----------|-----------|-----------|-----------|----------------|
| | SAC | N-Rep | TempoRL | UTE | TAAC | SDAR |
| Classic Control | 0.60±0.13 | 0.89±0.01 | 0.66±0.02 | 0.55±0.03 | 0.92±3E-3 | 1.0±0.0 |
| Locomotion | 0.78±2E-3 | 0.71±7E-3 | 0.35±0.02 | 0.43±0.01 | 0.80±0.02 | 1.0±0.0 |
| Manipulation | 0.91±4E-5 | 0.90±8E-4 | 0.77±0.02 | 0.79±0.02 | 0.95±6E-3 | 1.0±0.0 |
| Average | 0.76±0.02 | 0.83±0.01 | 0.59±0.05 | 0.59±0.03 | 0.90±6E-3 | 1.0±0.0 |

denotes the performance of random policies without effective training, while 1.0 denotes the performance of the best method. See Appendix C for the detailed computation process of the normalization. More experiment results are given in the Appendix D.

As illustrated in Fig. 3, our method **SDAR** (red lines) achieves higher sample efficiency than other baseline methods in various continuous control environments, corresponding to higher AUC scores shown in Table 1. This demonstrate effectiveness of performing *act-or-repeat* selection for each action dimension individually. Such spatially decoupled repetition framework enhances action persistence while maintaining policy flexibility, leading to efficient exploration and training. In the following, we analyze the performance of each method individually:

- Naive action repetition **N-Rep** achieves unstable improvement compared to vanilla **SAC**, but underperforms closed-loop repetition approaches generally. As described in Table 1, **N-Rep** achieves higher AUC scores than **SAC** in classic control tasks, such as *MountainCar* shown in Fig. 3(a). Naive repetition improves the persistence of actions during exploration, accelerating the training progress effectively in simple tasks. Nevertheless, this approach lacks flexibility, which

could negatively impact the performance in tasks requiring agile movements, such as *HalfCheetah* (Fig. 3(g)) in locomotion domains.

- Open-loop methods (**TempoRL** and **UTE**) show minor improvements over **SAC** in classic control tasks, while performing worse performance in other domains. These approaches force the agent to repeat actions for a predicted number of steps, eliminating the possibility of early termination. This inflexibility makes such methods difficult to be utilized in tasks requiring agile movements, such as *Humanoid* shown in Fig. 3(d).
- The closed-loop method **TAAC** outperforms **SAC** and open-loop approaches generally in various classes of tasks. **TAAC** introduces a switch policy to check whether to repeat previous actions at each step, which solves the lack of a termination mechanism in open-loop repetition methods. This leads to a more flexible repetition, which is suitable for both simple tasks and locomotion tasks requiring agile motions, such as *Ant* shown in Fig. 3(e).

However, **TAAC** treats all action dimensions as a whole during *act-or-repeat* selection, which downgrades the effectiveness of action repetition. Take *Humanoid* and *HalfCheetah* shown in Fig. 3(d) and Fig. 3(g) respectively as examples, **TAAC** (brown lines) demonstrates superior sample efficiency during the initial training phase, specifically between 0 and 1M steps. However, in later stages, the agent’s learning speed diminishes compared to **SDAR**, which even results in performance decline in the *Humanoid* task depicted in Fig. 3(d). This is because the *Humanoid* is composed of multiple actuators, where some actuators are required to select *act* for action diversity, while others choosing *repeat* for action persistence at the same time. However, **TAAC** is constrained to repeat actions of all actuators simultaneously, leading to either inadequate repetition for inefficient exploration, or excessive repetition to damage the policy performance.

- Our method **SDAR** outperforms other methods in various tasks, including **TAAC** in *locomotion* tasks. **SDAR** decouples action dimensions during closed-loop *act-or-repeat* selection, which is more flexible and suitable for various types of tasks. Taking the *Humanoid* task illustrated in Fig. 3(d) as an instance, **SDAR** compromises marginal efficiency in the initial stages to achieve greater overall efficiency and stability throughout the entire training process. Above all, **SDAR** achieves a better balance between action persistence for efficient exploration, and action diversity for high policy performance.

5.3 POLICY PERFORMANCE AND FLUCTUATION

In this work, we assess the policies trained through each method on the following metrics: **(1) Episode Return**: the average cumulative reward acquired by the policy in an episode, assessing the policy’s capability to solve the task successfully. **(2) Action Persistence Rate (APR)** and **(3) Action Fluctuation Rate (AFR)** formulated as follows:

$$\text{APR} = \frac{1}{1-p}, p = \mathbb{E}_\pi \left[\frac{1}{T \cdot |A|} \sum_{t=1}^T \sum_{i=1}^{|A|} \delta \left(a_t^{(i)} - a_{t-1}^{(i)} \right) \right]; \text{AFR} = \mathbb{E}_\pi \left[\frac{1}{T} \sum_{t=1}^T \|a_t - a_{t-1}\| \right],$$

where T denotes the trajectory length, $a_t^{(i)}$ denotes the action at the t -th step in the i -th dimension, and p denotes the average repetition probability of $a_t^{(i)}$ at each step. In this work, APR represents the average interval that two new decisions are made. APR evaluates *action persistenc* of the policy, where a larger APR denotes more repetition times during interactions. AFR evaluates the mean amplitude of fluctuations between successive steps, where a smaller AFR indicates smoother actions with fewer fluctuations.

The results of episode return, APR, and AFR are shown in Table 2. As illustrated in the tables, although the baseline method such as **N-Rep** achieves excellent results in both APR and AFR, it suffers a considerable reduction on policy effectiveness, corresponding to lower episode return compared to **SAC**. This is unacceptable and makes such methods impractical for real-world applications. In contrast, our method **SDAR** outperforms baseline methods on episode return, while achieving a higher APR and a lower AFR than the vanilla DRL, demonstrating the effectiveness of our method. In addition, we can also observe that:

- Naive repetition method **N-Rep** achieves high APR and low AFR, at the expense of performance reduction on episode returns. This is because the actions in **N-Rep** are repeated for frozen times,

Table 2: Experiment results on **Episode Return**, **APR (higher is better)**, and **AFR (lower is better)** in different tasks. The best results on episode returns are boldfaced. Our method SDAR outperforms baseline methods on episode returns generally, achieving excellent action persistence with fewer fluctuation than vanilla methods.

| Tasks | Episode Return (Mean \pm Standard Error) | | | | | |
|----------------|---|----------------------------------|------------------|----------------------------------|------------------|----------------------------------|
| | Action Persistence Rate (APR) / Action Fluctuation Rate (AFR) | | | | | |
| | SAC | N-Rep | TempoRL | UTE | TAAC | SDAR (Ours) |
| LunarLander | 275.9 \pm 6.83 | 280.8 \pm 1.21 | 281.5 \pm 7.22 | 282.8\pm8.74 | 261.4 \pm 18.9 | 282.2 \pm 5.84 |
| | 1.00 / 0.09 | 4.00 / 0.08 | 1.43 / 0.18 | 1.38 / 0.36 | 3.05 / 0.11 | 11.18 / 0.10 |
| Walker2d | 5305 \pm 367 | 4724 \pm 163 | 2866 \pm 897 | 2986 \pm 836 | 5660 \pm 394 | 6028\pm406 |
| | 1.00 / 0.15 | 4.00 / 0.09 | 5.74 / 0.26 | 7.81 / 0.24 | 1.30 / 0.22 | 2.96 / 0.12 |
| HalfChee. | 13122 \pm 2877 | 8378 \pm 1753 | 8065 \pm 1799 | 7917 \pm 293 | 11148 \pm 3921 | 15131\pm1279 |
| | 1.00 / 0.68 | 2.00 / 0.47 | 2.10 / 0.66 | 2.69 / 0.58 | 1.02 / 0.61 | 1.22 / 0.62 |
| Humanoid | 6184 \pm 717 | 5074 \pm 310 | 1022 \pm 397 | 2595 \pm 334 | 7308 \pm 244 | 7483\pm288 |
| | 1.00 / 0.28 | 4.00 / 0.09 | 5.27 / 0.15 | 5.69 / 0.18 | 1.21 / 0.25 | 1.67 / 0.19 |
| Pusher | -22.5 \pm 1.40 | -21.2\pm1.08 | -48.2 \pm 2.05 | -41.3 \pm 5.56 | -30.5 \pm 1.85 | -21.3 \pm 1.26 |
| | 1.00 / 0.022 | 4.00 / 0.019 | 1.15 / 0.032 | 1.01 / 0.018 | 1.75 / 0.031 | 1.69 / 0.015 |
| Average | 0.89 \pm 0.07 | 0.81 \pm 0.19 | 0.59 \pm 0.33 | 0.64 \pm 0.27 | 0.91 \pm 0.10 | 1.00\pm0.001 |
| | 1.00 / 0.245 | 3.60 / 0.150 | 3.12 / 0.257 | 3.71 / 0.276 | 1.66 / 0.244 | 3.75 / 0.208 |

where repeated actions may be unreasonable in some states. This inflexible repetition strategy cannot be agilely adjusted during interaction, leading to performance reduction, especially in *locomotion* tasks such as *HalfCheetah*.

- Open-loop methods achieve high APR, indicating a lot of action repetitions during interaction, at the expense of more fluctuations and superior episode return performance. For instance, **UTE** achieves higher APR (3.71) than **SAC** (1.0), with large fluctuation and low episode return. This suggests that *the increment of repetition times in an inflexible manner may be harmful to policy effectiveness and smoothness*. In contrast, **TAAC** solves this problem, achieving a moderate AFR (1.66), with a comparable AFR (0.244) and high episode return (0.91).
- Different to **UTE** and **SDAR**, although **SDAR** also selects a lot of *repeat* during interaction with a high APR (3.69), **SDAR** achieves excellent performance on fluctuation and episode returns simultaneously. This is because **SDAR** performs repetitions in a more flexible manner. The decoupling design is consistent with the requirement of continuous control tasks, thus can conduct repetition without harms to episode returns and fluctuation performance.

5.4 VISUALIZATION OF ACT-OR-REPEAT SELECTION

In order to analyze the difference between repetition behaviors of **SDAR** and previous methods, we perform visualization of *act-or-repeat* selections in *LunarLander* and *Walker2d*. The results are illustrated in Fig. 4(a) and Fig. 4(b), where dark blue and light blue blocks denote *act* and *repeat* respectively.

As shown in the figures, different dimensions require different decision frequencies, corresponding to actions being repeated at distinct steps for each dimension. Take *LunarLander* in Fig. 4(a) for example, the lateral boosters (a_1) change actions frequently to adjust the rocket pose, while the main engine (a_0) adjusts decisions occasionally.

Table 3: Experiment results of each action dimensions in *Walker2d*.

| APR Val. | TAAC | SDAR |
|-----------------|------|-------------|
| Thigh (a_0) | 1.30 | 3.70 |
| Leg (a_1) | 1.30 | 2.17 |
| Foot (a_2) | 1.30 | 3.44 |
| Episode Ret. | 5660 | 6028 |
| AFR | 0.22 | 0.12 |

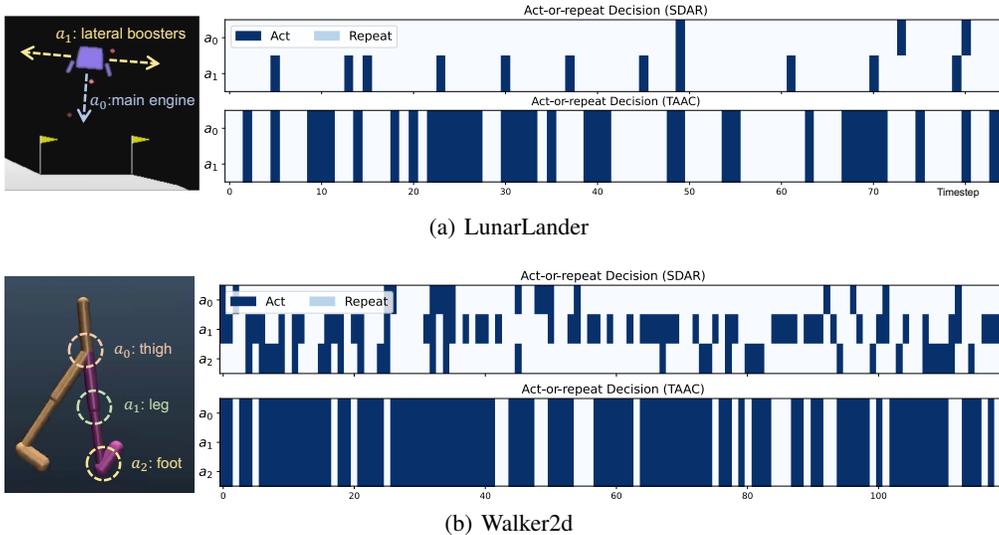


Figure 4: Visualization of *act-or-repeat* selections of **SDAR** and **TAAC** algorithms in *LunarLander* and *Walker2d* tasks. The x -axis denotes timesteps, and the y -axis denotes different action dimensions. The light blue blocks indicate *repetition*, while dark blue blocks represent *act*, i.e. change actions in the corresponding action dimensions.

SDAR can adjust action persistence for each dimension automatically through spatially decoupled framework, offering more flexible repetition strategies compared to **TAAC**. As described in Fig. 4(b) and Table 3, **SDAR** controls the walker with higher persistence for the leg joints, while lower persistence for thighs and feet. In contrast, all joints in **TAAC** are required to repeat simultaneously, resulting in same persistence for each joint with lower repetitions behaviors.

SDAR achieves high action persistence, while maintaining agility required by the task, leading to higher policy performance with fewer action fluctuations. As shown in Table 3, **SDAR** obtains higher APR than **TAAC** in *Walker2d*, while achieving higher episode return with lower AFR, demonstrating the effectiveness of our framework in continuous control tasks.

6 CONCLUSION

In this work, we propose a novel action repetition framework for continuous control tasks called SDAR, which implements closed-loop *act-or-repeat* selection for each action dimension individually. Such a spatially decoupled design improves the flexibility of repetition strategies, leading to improved balance between action *persistence* and *diversity*, while maintaining action agility for continuous control tasks. Experiments are conducted in various task scenarios, where SDAR achieves higher sample efficiency, superior policy performance, and reduced action fluctuation, demonstrating the effectiveness of our method.

This work provides insights into spatially decoupled framework for action repetition and temporal abstraction. In this study, the selection policy β is designed to decide *act-or-repeat* for each dimension independently, without accounting for inter-dimensional correlations. For instance, actuators in humanoid robots can be divided into multiple categories, with each category typically requiring same decisions during *act-or-repeat* selections. This issue will be further researched in future works.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (Grant No. 62373242 and No.92248303), Shanghai Municipal Science and Technology Major Project (Grant No. 2021SHZDZX0102).

REFERENCES

- Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.
- 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*, 47: 253–279, 2013.
- André Biedenkapp, Raghu Rajan, Frank Hutter, and Marius Lindauer. Temporal: Learning when to act. In *International Conference on Machine Learning*, pp. 914–924. PMLR, 2021.
- Alex Braylan, Mark Hollenbeck, Elliot Meyerson, and Risto Miikkulainen. Frame skip is a powerful parameter for learning to play atari. In *Workshops at the twenty-ninth AAAI conference on artificial intelligence*, 2015.
- Arun Tejasvi Chaganty, Prateek Gaur, and Balaraman Ravindran. Learning in a small world. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pp. 391–397, 2012.
- Chen Chen, Hongyao Tang, Jianye Hao, Wulong Liu, and Zhaopeng Meng. Addressing action oscillations through learning policy inertia. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 7020–7027, 2021.
- Will Dabney, Georg Ostrovski, and Andre Barreto. Temporally-extended ϵ -greedy exploration. In *International Conference on Learning Representations*, 2021. URL <https://openreview.net/forum?id=ONBPHFZ7zG4>.
- Ben Eysenbach, Russ R Salakhutdinov, and Sergey Levine. Search on the replay buffer: Bridging planning and reinforcement learning. *Advances in neural information processing systems*, 32, 2019.
- Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. PMLR, 2018.
- Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 3389–3396. IEEE, 2017.
- Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.
- Anna Harutyunyan, Peter Vrancx, Pierre-Luc Bacon, Doina Precup, and Ann Nowe. Learning with options that terminate off-policy. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.
- Anna Harutyunyan, Will Dabney, Diana Borsa, Nicolas Heess, Remi Munos, and Doina Precup. The termination critic. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 2231–2240. PMLR, 2019.
- Matteo Hessel, Joseph Modayil, H. V. 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*, 2017.
- Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and JoÃ£o GM AraÃ§o. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.
- Shivaram Kalyan Krishnan, Siddharth Aravindan, Vishwajeet Bagdawat, Varun Bhatt, Harshith Goka, Archit Gupta, Kalpesh Krishna, and Vihari Piratla. An analysis of frame-skipping in reinforcement learning. *arXiv preprint arXiv:2102.03718*, 2021.

- B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926, 2021.
- Aravind Lakshminarayanan, Sahil Sharma, and Balaraman Ravindran. Dynamic action repetition for deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.
- Jongmin Lee, Byung-Jun Lee, and Kee-Eung Kim. Reinforcement learning for control with multiple frequencies. *Advances in Neural Information Processing Systems*, 33:3254–3264, 2020a.
- Joongkyu Lee, Seung Joon Park, Yunhao Tang, and Min-hwan Oh. Learning uncertainty-aware temporally-extended actions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 13391–13399, 2024.
- Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47):eabc5986, 2020b.
- TP Lillicrap. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.
- Zichuan Lin, Junyou Li, Jianing Shi, Deheng Ye, Qiang Fu, and Wei Yang. Juewu-mc: Playing minecraft with sample-efficient hierarchical reinforcement learning. *arXiv preprint arXiv:2112.04907*, 2021.
- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Belle-mare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.
- Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, 31, 2018.
- Shubham Pateria, Budhitama Subagdja, Ah-hwee Tan, and Chai Quek. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 54(5):1–35, 2021.
- Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.
- Doina Precup. *Temporal abstraction in reinforcement learning*. University of Massachusetts Amherst, 2000.
- Ritu Raman, Caroline Cvetkovic, and Rashid Bashir. A modular approach to the design, fabrication, and characterization of muscle-powered biological machines. *Nature protocols*, 12(3):519–533, 2017.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- Sahil Sharma, Aravind S. Lakshminarayanan, and Balaraman Ravindran. Learning to repeat: Fine grained action repetition for deep reinforcement learning. In *International Conference on Learning Representations*, 2017. URL <https://openreview.net/forum?id=BlGOWV5eg>.
- David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pp. 387–395. Pmlr, 2014.
- Martin Stolle and Doina Precup. Learning options in reinforcement learning. In *Abstraction, Reformulation, and Approximation: 5th International Symposium, SARA 2002 Kananaskis, Alberta, Canada August 2–4, 2002 Proceedings 5*, pp. 212–223. Springer, 2002.
- Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

- Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.
- Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.
- Alexander Vezhnevets, Volodymyr Mnih, Simon Osindero, Alex Graves, Oriol Vinyals, John Agapiou, et al. Strategic attentive writer for learning macro-actions. *Advances in neural information processing systems*, 29, 2016.
- Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *International conference on machine learning*, pp. 3540–3549. PMLR, 2017.
- Xintong Yang, Ze Ji, Jing Wu, Yu-Kun Lai, Changyun Wei, Guoliang Liu, and Rossitza Setchi. Hierarchical reinforcement learning with universal policies for multistep robotic manipulation. *IEEE Transactions on Neural Networks and Learning Systems*, 33(9):4727–4741, 2021.
- Haonan Yu, Wei Xu, and Haichao Zhang. Taac: Temporally abstract actor-critic for continuous control. *Advances in Neural Information Processing Systems*, 34:29021–29033, 2021.
- Haichao Zhang, Wei Xu, and Haonan Yu. Generative planning for temporally coordinated exploration in reinforcement learning. In *International Conference on Learning Representations, 2022*. URL <https://openreview.net/forum?id=YZHES8wIdE>.
- Jesse Zhang, Haonan Yu, and Wei Xu. Hierarchical reinforcement learning by discovering intrinsic options. In *International Conference on Learning Representations, 2021*. URL <https://openreview.net/forum?id=r-gPPHEjpmw>.
- Yang Zhang, Buqing Nie, and Yue Gao. Robust locomotion policy with adaptive lipschitz constraint for legged robots. *IEEE Robotics and Automation Letters*, 2024.

A PROOF OF THE STATEMENT IN SECTION 4.1

Statement. Given $\forall s \in \mathcal{S}, a^- \in \mathcal{A}, b \in \{0, 1\}^{|A|}$, based on the two-stage policy described in Sec. 4.1, the output action a replicates the same actions as a^- in *repetition* dimensions $\{i | b_i = 0, 1 \leq i \leq |A|\}$.

Proof. As described in Sec. 4.1, $a = (1 - b) \odot a^- + b \odot \hat{a}$. Thus, for $\forall b, a^-$, we can obtain that:

$$\begin{aligned} (a - a^-) \odot (1 - b) &= ((1 - b) \odot a^- + b \odot \hat{a} - a^-) \odot (1 - b) \\ &= b \odot (\hat{a} - a^-) \odot (1 - b) \\ &= b \odot (1 - b) \odot (\hat{a} - a^-) \\ &= \mathbf{0} \end{aligned} \tag{11}$$

For $\forall i \in \{i | b_i = 0\}$, we have $[(a - a^-) \odot (1 - b)]_i = a^{(i)} - a^{-, (i)} = 0$, where $a^{(i)}$ denotes the action in the i -th dimension. Thus, a replicates the same actions as a^- in dimensions choosing *repetition*. \square

B EXPERIMENT DETAILS

B.1 ALGORITHM SETTINGS

Table 4: Hyper-parameter settings for SDAR algorithm.

| Parameter | Setting | Parameter | Setting |
|-------------------------|--------------------|----------------------------|--------------------|
| Learning rate (π) | 3×10^{-4} | Learning rate (β) | 3×10^{-4} |
| Learning rate (Q) | 1×10^{-3} | Learning rate (α) | 1×10^{-3} |
| Optimizer | Adam | Discount factor γ | 0.99 |
| Batch size | 256 | Policy delay | 2 |
| Soft update τ | 0.005 | Sample number (b) | 10 |

The hyper-parameter settings are shown in Table 4. In addition, we need to tune the target entropies \mathcal{H}_β and \mathcal{H}_π to improve the efficiency of the entropy-based exploration described in Eq. (10). In this work, we utilize $\mathcal{H}_\pi = -|A|$, where $|A|$ denotes the size of the action space, corresponding to the recommendation given in (Haarnoja et al., 2018). Besides, $\mathcal{H}_\beta = \lambda \cdot |A| \log 2$, where λ is a hyper-parameter tuned according to the tasks. In this work, we set $\lambda \in [0.4, 0.6]$, which achieves excellent performance on sample efficiency and policy effectiveness.

As illustrated in Sec. 4.1, SDAR is composed of two policy networks β and π based on MLP. In this work, we design two MLPs with the same structures: $(|S| + |A| \rightarrow 256 \rightarrow 256 \rightarrow |A|)$ with ReLU activation functions.

SDAR policies for tasks with $|A| \leq 3$ are trained through Eq. (8), such as *Reacher* and *LunarLanderContinuous*. Policies in other environments such as *HalfCheetah* and *Humanoid* are optimized through Eq. (9) for lower computation costs.

B.2 EXPERIMENT TASKS

The experiment tasks are listed in Table 5, where the action spaces of all tasks are normalized as $[-1, 1]$ for convenience. All tasks are constructed based on Gymnasium (Plappert et al., 2018). The *FetchPickandPlace* and *FetchReach* tasks are implemented by Gymnasium-Robotics (Plappert et al., 2018), where *observation* and *desired_goal* are concatenated as the observation in this experiment.

Table 5: Descriptions of experiment tasks in this work.

| Caterogy | Task Name | Obervation Space | Action Space |
|-----------------|-----------------------|--------------------|----------------|
| Classic Control | MountainCarContinuous | \mathbb{R}^2 | $[-1, 1]^1$ |
| | LunarLanderContinuous | \mathbb{R}^8 | $[-1, 1]^2$ |
| | BipedalWalker | \mathbb{R}^{24} | $[-1, 1]^6$ |
| Locomotion | HalfCheetah | \mathbb{R}^{17} | $[-1, 1]^6$ |
| | Humanoid | \mathbb{R}^{376} | $[-1, 1]^{17}$ |
| | Walker2d | \mathbb{R}^{17} | $[-1, 1]^6$ |
| | Hopper | \mathbb{R}^{11} | $[-1, 1]^3$ |
| | Ant | \mathbb{R}^{27} | $[-1, 1]^8$ |
| Manipulation | FetchReach | \mathbb{R}^{13} | $[-1, 1]^4$ |
| | Pusher | \mathbb{R}^{23} | $[-1, 1]^7$ |
| | Reacher | \mathbb{R}^{11} | $[-1, 1]^2$ |

B.3 BASELINES

- (1) **SAC** (Haarnoja et al., 2018) is a famous model-free RL in continuous control domains, which trains policies efficiently with entropy-based exploration strategies. In this work, we utilize SAC implementation and hyper-parameter settings proposed in CleanRL (Huang et al., 2022)¹.
- (2) **N-Rep** forces the agent to repeat the actions output by the policy for n times, where n is a hyper-parameter. In this work, this algorithm is implemented based on SAC, where n is tuned in $[2, 5]$ to achieve the balance between persistence of action and diversity.
- (3) **TempoRL** (Biedenkapp et al., 2021) repeats policy actions dynamically in an *open-loop* manner, using a *skip-policy* to predict when to make the next decision. In this work, this method is implemented based on the official repository².
- (4) **UTE** (Lee et al., 2024) is improved based on **TempoRL**, which performs repetition while incorporating the estimated uncertainty of the repeated action values. In this work, this method is implemented based on the official repository³.
- (4) **TAAC** (Yu et al., 2021) is a closed-loop repetition framework, which determines whether to repeat the previous action at each step utilizing a switch policy. In this work, this experiment is conducted utilizing the official implemenatation⁴.

B.4 COMPUTATION RESOURCES

In this work, we conduct all experiment utilizing NVIDIA RTX 3090 GPU and Pytorch 2.1 with CUDA 12.2. The training time of our method compared to vanilla SAC is shown in Table 6.

As shown in Table 6, our method requires less training times than vanilla **SAC** to achieve the same policy performance, because of the higher sample efficiency of **SDAR**, requiring less training steps in this experiment. In addition, there exist two policy networks β and π in the SDAR, both of which need to be optimized through gradient descent. However, β and π can be optimized in parallel during training, achieving an improved time efficiency, which will be researched in future works.

C NORMALIZATION OF PERFORMANCE

Different tasks can have vastly different reward scales, thus cannot be directly averaged. In order to depict the average episode return and AUC scores across various tasks, we compute the normalized normalized score (n-score) of each method, which is widely utilized in prior works Hessel et al.

¹<https://github.com/vwxyzjn/cleanrl>

²<https://github.com/automl/TempoRL>

³<https://github.com/oh-lab/UTE-Uncertainty-aware-Temporal-Extension/>

⁴<https://github.com/HorizonRobotics/alf>

Table 6: The computational cost for the training of SAC and SDAR algorithms. Both two methods are trained to achieve the same performance as SAC with sufficient steps.

| Tasks | Steps Cost | | Time Cost | |
|---------------|------------|------|-----------|--------|
| | SAC | SDAR | SAC | SDAR |
| LunarLander | 300K | 150K | 60min | 36min |
| BipedalWalker | 600K | 300K | 108min | 63min |
| Walker2d | 1.5M | 700K | 282min | 167min |
| Humanoid | 3.0M | 1.5M | 595min | 365min |

(2017); Yu et al. (2021). Take normalized episode return as an example, given an episode return Z , its n-score is calculated as $Z_{norm} = \frac{Z - Z_0}{Z_1 - Z_0}$, where Z_0 and Z_1 denote the episode return of random policies and vanilla DRL policies respectively. The n-score $Z_{norm} \in [0, 1]$, thus can be averaged across different tasks.

D ADDITIONAL TRAINING CURVES

Additional training curves of each method in additional tasks are illustrated in Fig. 5.

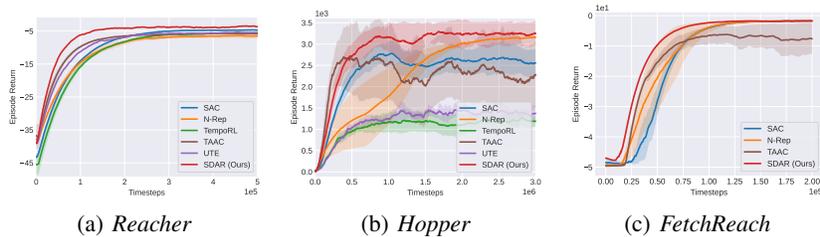


Figure 5: Learning curves of SDAR (red) in additional tasks against baseline methods.