Sample Efficient Experience Replay in Non-stationary Environments

Anonymous submission

Abstract

In non-stationary environments, off-policy reinforcement learning (RL) agents heavily rely on effective sampling for adaptive planning. However, existing prioritized replay mechanisms assume a stationary environment, neglecting the negative impact of outdated samples. In this paper, we introduce Discrepancy of Environment Dynamics (DoE), which quantifies the value of transitions in non-stationary environments by measuring changes in environmental dynamics. Based on DoE, we propose **D**iscrepancy of **E**nvironment Prioritized **E**xperience **R**eplay (DEER), sample-efficient experience replay method for non-stationary environments. It employs binary classifier-based density estimation to monitor environmental dynamics in real time and adaptively adjust the sampling strategy. DEER prioritizes transitions beneficial to the latest policy, swiftly correcting policy biases caused by environmental changes. Experimental results across four non-stationary Mujoco tasks with two off-policy algorithms demonstrate that DEER significantly improves sample efficiency compared to the state-of-the-art (SOTA) methods.

Introduction

Reinforcement Learning (RL) is a dynamic sequential decision-making planning method widely applied in real-world scenarios, many of which exhibit non-stationarity. Non-stationary environments refer to systems where environmental dynamics (such as state transition probability functions and rewards) evolve over time. This necessitates adaptive planning to cope with the complexity and unpredictability of real-world environments.

Off-policy RL, which allows agents to learn from past experiences, has been widely employed to address the issue of sparse and expensive sampling in high-dimensional continuous action-state spaces (Yarats et al. 2021b). This is achieved through Experience Replay (ER), which stores and reuses past experiences (known as transitions) for training. Many studies have explored non-uniform sampling to improve sample efficiency, primarily focusing on using temporal difference error (TD-error) as a prioritized metric, which has proven effective in stationary environments(Sun, Zhou, and Li 2020; Li, Qian, and Song 2024). TD-error measures the disparity between the estimated return and the actual return. By prioritizing transitions with higher TD-error, informative experiences are frequently reused, thus accelerating the training.

In non-stationary environments, past experiences are no longer representative of the current environment, posing challenges to effective sampling (Rahimi-Kalahroudi et al. 2023), and existing methods overlook the negative impact of outdated samples in these settings. The environmental dynamics and policy improvement determine TD-error. When the value function has been well-trained to adapt to the new environment, the transition collected before environmental changes has a higher TD-error than those from the new environment. Prioritizing these outdated samples emphasizes irrelevant experiences to the current environment and thus deteriorates the effectiveness of model training. Similar issues occur with other prioritization metrics, such as those based on rewards (Cao et al. 2019) or access frequency (Sun, Zhou, and Li 2020). The core problem is that prioritization metrics focused solely on policy improvement fail to account for changing environmental dynamics, thereby not assessing the importance of transitions in the current environment.

To address this challenge, we introduce Discrepancy of Environment Dynamics (DoE) to quantify the impact of environmental dynamics on a transition. For the same state-action pair, the difference in the action value functions is an intuitive representation of the dynamics disparity in the environment. DoE is calculated as the discrepancy between the value of the current action value function and its value before the environmental changes, excluding the difference caused by policy improvements.

Based on DoE, we propose a sample-efficient experience replay method, named Discrepancy of Environment Prioritized Experience Replay (DEER), which adaptively prioritizes experiences that facilitate both policy improvement and environmental adaptation in non-stationary environments. Due to the complexity of modeling dynamic patterns, we use a binary classifier to evaluate the probability density of reward sequences within adjacent time windows, indirectly observing changes in environmental dynamics. We employ Jensen-Shannon divergence as a metric to quantify the extent of change in the current environment. Additionally, the intrinsic characteristics of non-stationary environments necessitate using different priority metrics for transitions before and after the change. For pre-change transitions, a lower DoE indicates their effectiveness in the new environment, enabling the exclusion of outdated samples. For post-change transitions, we apply a hybrid metric priority sam-

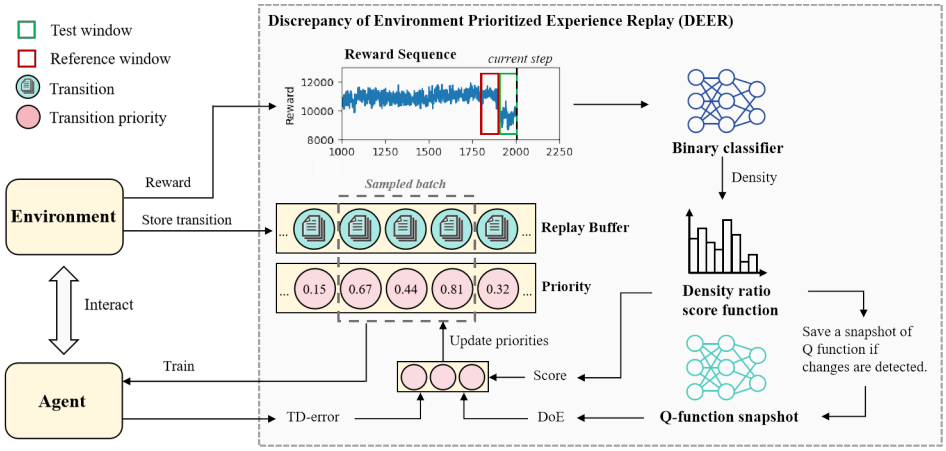


Figure 1: The workflow of DEER.

pling based on real-time density differences, incorporating both TD-error and DoE. This design enables adaptive prioritization between policy improvement and environmental adaptation, while maintaining balance and diversity in sampling. The source code of DEER is available at XXX.[1]

Our contributions are summarized as follows:

(1) We present DoE, as a priority metric that quantifies the importance of samples in non-stationary environments, by calculating the impact of environmental dynamics changes in the environment on the value of the transition.

(2) We propose DEER, a novel hybrid metric prioritized experience replay method capable of adaptively adjusting sampling weights in response to changes in environmental dynamics.

(3) We deploy two popular off-policy RL algorithms to validate the effectiveness of the DEER across different non-stationary continuous control tasks. The results demonstrate that the proposed method outperforms all baselines.

The rest of the paper is organized as follows. Section 2 elaborates on the design of DEER, followed by performance evaluation in Section 3. Related works are discussed in Section 4. Finally, conclusions are presented in Section 5.

Methodology

In this section, we will elaborate on the proposed Discrepancy of Environment Prioritized Experience Replay (DEER) method. Figure 1 shows the workflow of DEER. The agent interacts with the environment, generating rewards and storing transitions in a replay buffer. We use a binary classifier to compute density scores for reference-test time window pairs in real-time. These scores are then processed through a density-ratio score function. Concurrently, the system detects change points based on the scores and saves Q-function snapshots to compute the DoE of transitions. The sample priority is then updated based on a hybrid metric that combines TD error and DoE.

[1]The repository URL will be made available upon acceptance.

Preliminary

We formulate the problem of RL in non-stationary environment as a family of Markov decision processes (MDPs) (Padakandla 2021), which is defined as $\{\langle \mathcal{S}, \mathcal{A}, P_i, R_i, T_i, \gamma \rangle\}_{i=0}^{\infty}$, where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ represents the action space and R_i is the reward function. At each time step t, the agent following the policy $\pi(a_t \mid s_t)$ chooses an action $a_t \in \mathcal{A}$ and interacts with the environment. The environment transits to the next state $s_{t+1} \in \mathcal{S}$ according to state-transition probability function $P(s_{t+1} \mid s_t, a_t)$ and provides a reward $r_t = R_i(s_t, a_t)$ to the agent. T_i denotes the time steps at which environmental dynamics change, known as change points. Environmental dynamics are determined by the state-transition probability function and the reward function, which denotes as $\langle P_i, R_i \rangle$. The state-transition probability function of non-stationary environments is defined as:

$$P(s_{t+1} \mid s_t, a_t) = \begin{cases} P_0(s_{t+1} \mid s_t, a_t), 0 \le t \le T_0 \\ P_1(s_{t+1} \mid s_t, a_t), T_0 \le t \le T_1 \\ \dots \\ P_i(s_{t+1} \mid s_t, a_t), T_{i-1} < t \le T_i \\ \dots \end{cases} \quad (1)$$

The reward function of non-stationary environments is defined as:

$$R(s_t, a_t) = \begin{cases} R_0(s_t, a_t), 0 \le t \le T_0 \\ R_1(s_t, a_t), T_0 \le t \le T_1 \\ \dots \\ R_i(s_t, a_t), T_{i-1} < t \le T_i \\ \dots \end{cases} \quad (2)$$

The objective of an agent is to learn a policy π to maximize the expected discounted return $\mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k}\right]$, where γ denotes the discount factor. Q-functions (namely, action value functions) is proposed to estimate expected discounted return (Mnih et al. 2013). Given the state-action pair (s_t, a_t), the Q-function is defined as the expected discounted return following the policy and environmental dynamics:

$$Q(s_t, a_t) = \mathbb{E}_{s_j \sim P, a_j \sim \pi(a|s_j)}\left[\sum_{j=t}^{\infty} \gamma^{j-t} R(s_j, a_j)\right] \quad (3)$$

The policy in Q-learning derives from maximizing the expected discounted return, i.e., $\pi = \arg\max_a Q(s,a)$. The Q-function is updated by temporal difference learning:

$$\begin{aligned} Q(s_t,a_t) \leftarrow & Q(s_t,a_t) + \\ & \eta\left[r_t + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)\right] \end{aligned} \tag{4}$$

where η is the learning rate and $r_t + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)$ is called the temporal difference error (TD-error).

Motivation

Off-policy RL employs temporal difference learning to update the policy and the Q-function, by comparing the estimation error (i.e., TD-error) between the current and the next time step's action-state pair. For a given transition (s_k, a_k, r_k, s_{k+1}), the TD-error is defined as follows:

$$TD(s_k,a_k,r_k,s_{k+1}) = r_k + \gamma Q(s_{k+1},a_{k+1}) - Q(s_k,a_k) \tag{5}$$

In stationary environments, the absolute value of TD-error can serve as a priority metric. A higher absolute value of TD-error indicates a larger value estimation error for the action-state pair in the transition, indicating greater opportunities for policy improvement. However, after a change in environmental dynamics, transitions collected before the change may have higher priority than those obtained recently. Motivated by this, we first derive the priority of transition (s_k, a_k, r_k, s_{k+1}) as follows:

$$\begin{aligned} & |TD(s_k,a_k,r_k,s_{k+1})| \\ & = |r_k + \gamma Q(s_{k+1},a_{k+1}) - Q(s_k,a_k)| \\ & = \Big| r_k + \gamma \mathbb{E}_{s_j\sim P_i, a_j\sim\pi(a|s_j)} \left[\textstyle\sum_{j=k+1}^{\infty} \gamma^{j-t-1} R_i(s_j,a_j)\right] \\ & \quad - \mathbb{E}_{s_j\sim P_i, a_j\sim\pi(a|s_j)} \left[\textstyle\sum_{j=k}^{\infty} \gamma^{j-t} R_i(s_j,a_j)\right] \Big| \\ & = \left| R(s_k,a_k) - \mathbb{E}_{s_k\sim P_i, a_k\sim\pi(a|s_k)} \left[R_i(s_k,a_k)\right] \right| \end{aligned} \tag{6}$$

Note that the Q-function estimates expected returns based on current environmental dynamics $\langle P_i, R_i \rangle$, with the reward function $R(s_k,a_k)$ is given by Eq. 2. Eq. 6 indicates that priority is determined by the reward function and estimated rewards following the current state-transition probability function. With the Q-function fine-tuned for the new environmental dynamics, transitions collected in the previous environmental dynamics are given higher priority than those in the current dynamics. This occurs due to the significant disparity between the previous and current reward functions. Consequently, higher-priority transitions reflect outdated experiences, hindering the agent's adaptation to new environmental dynamics.

Discrepancy of Environment (DoE)

An experience replay method for non-stationary environments should assign higher priority to transitions that facilitate agent adaptation to current environmental dynamics and exhibit significant differences from previous environmental dynamics. Consequently, we first monitoring changes in environmental dynamics to identify change points.

Since the environment dynamics are implicitly modeled in the state-transition probability function, we monitor these environmental dynamics through trajectories

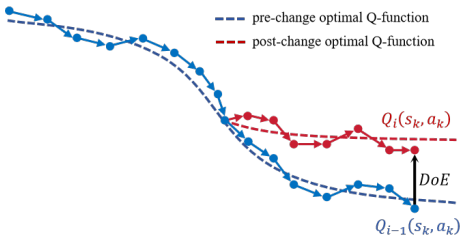


Figure 2: Illustration of fine-tuning the Q-function with the high DoE transition.

formed by environment-agent interactions. Let $\tau = [s_0, a_0, s_1, a_1, \ldots, s_i, a_i, \ldots]$ represent the trajectory generated by the agent following the policy $\pi(a \mid s)$. The probability density of trajectory is determined by the policy and the environment dynamics:

$$P(\tau) = P(s_0) \prod_{t=0}^{\infty} \pi(a_t \mid s_t) P(s_{t+1} \mid s_t, a_t) \tag{7}$$

where $P(s_0)$ denotes the initial state distribution, i.e., the probability density that trajectory τ starts from state s_0. When the initial state density and policy remain constant, any change in environmental dynamics leads to varying trajectory distribution. We define $\mathbf{r}_\tau = [r_0, r_1, \ldots, r_i, \ldots]$ as the reward sequence obtained by the agent following the trajectory τ, where $r_i = R(s_i,a_i)$. The probability density of the reward sequence $P(\mathbf{r}_\tau)$ is also determined by Eq. 7, as the reward is generated by the reward function and is independent of the state-transition probability function. Therefore, we approximate the changes in environmental dynamics by analyzing changes in the reward sequence.

We use density-ratio estimation (DRE) based on binary classifier (Hillman and Hocking 2021) on reward sequence to detect change points. DRE enables online computation, making it suitable for off-policy reinforcement learning with online transition collection. We use two adjacent time windows to generate reference and test sets from the reward sequence and set a binary label $l \in \{0,1\}$ for them. Specifically, let $\widetilde{\mathbf{r}}_{rf} = \{(\mathbf{r}_{i:j}, l) \mid t-2m+1 \le i < j \le t-m, l=0\}$ and $\widetilde{\mathbf{r}}_{te} = \{(\mathbf{r}_{i:j}, l) \mid t-m < i < j \le t, l=1\}$ be the reference and test sets at time step t, respectively, where $\mathbf{r}_{i:j} = [r_i, r_{i+1}, \ldots, r_j]$ and m denotes the window size. We use a binary classifier to estimate the probability density of the reference and test sets:

$$P(\mathbf{r} \mid f(\mathbf{r}) = l) = \begin{cases} P_{rf}(\mathbf{r}) & \text{if } l = 0 \\ P_{te}(\mathbf{r}) & \text{if } l = 1 \end{cases} \tag{8}$$

where $P_{rf}(\cdot)$ and $P_{te}(\cdot)$ denote the density functions of the reference and test sets, respectively, and $f(\cdot)$ represents the binary classifier. We use a neural network as a binary classifier and calculate the loss using the following cross-entropy function:

$$\mathcal{L}(f) = -\frac{1}{n_{rf}} \sum_{\mathbf{r}\in\widetilde{\mathbf{r}}_{rf}} \log(1 - f(\mathbf{r})) - \frac{1}{n_{te}} \sum_{\mathbf{r}\in\widetilde{\mathbf{r}}_{te}} \log f(\mathbf{r}) \tag{9}$$

where n_{rf} and n_{te} represent the number of samples in the reference and test sets, respectively. In practice, we partition the reward sequence within the window into equidistant segments based on the number of samples to generate these sets.

The change point is detected by the density ratio score function. If the density ratio score function $S(\widetilde{\mathbf{r}}_{te}, \widetilde{\mathbf{r}}_{rf}) \geq \mu$, is greater than or equal to the detection threshold μ is the detection threshold, a change point is considered to occur at time step $t-m$. We use the Jensen-Shannon divergence (Tu, Li, and Cai 2023) as the density ratio score function:

$$\begin{aligned} S(\widetilde{\mathbf{r}}_{te}, \widetilde{\mathbf{r}}_{rf}) = & \log 2 + \frac{1}{2n_{te}} \sum_{\mathbf{r} \in \widetilde{\mathbf{r}}_{te}} \log P(\mathbf{r} \mid f(\mathbf{r}) = 1) \\ & + \frac{1}{2n_{rf}} \sum_{\mathbf{r} \in \widetilde{\mathbf{r}}_{rf}} \log P(\mathbf{r} \mid f(\mathbf{r}) = 0) \end{aligned} \tag{10}$$

The derivation of Eq. 10 is detailed in Appendix A. After identifying change points, we propose a metric called "Discrepancy of Environment (DoE)", which quantifies the impact of environmental dynamics on the value estimation of action-state pair in the transition. We define the DoE for transition (s_k, a_k, r_k, s_{k+1}) as the difference in the Q-function between the pre- and post-change environmental dynamics for the state-action pair (s_k, a_k):

$$\begin{aligned} & DoE(s_k, a_k) \\ & = Q_i(s_k, a_k) - Q_{i-1}(s_k, a_k) \\ & = \mathbb{E}_{s_j \sim P_i, a_j \sim \pi(a|s_j)} \left[\sum_{j=k}^{\infty} \gamma^{k-t} R_i(s_j, a_j) \right] \\ & - \mathbb{E}_{s_j \sim P_{i-1}, a_j \sim \pi'(a|s_j)} \left[\sum_{j=k}^{\infty} \gamma^{j-t} R_{i-1}(s_j, a_j) \right] \end{aligned} \tag{11}$$

where $Q_{i-1}(\cdot)$ represents the Q-function at T_{i-1}, i.e., the latest Q-function learned in pre-change dynamics $\langle P_{i-1}, R_{i-1} \rangle$, and $Q_i(\cdot)$ represents the current Q-function in post-change dynamics $\langle P_i, R_i \rangle$. As shown in Figure 2, DoE measures the bias of the Q-function caused by changes in environmental dynamics. High DoE transitions in the new environmental dynamics can guide the agent to fine-tune the Q-function, thereby improving the policy to adapt to the changes. Conversely, low DoE transitions from the old environment can help the agent retain relevant information that remains applicable.

Discrepancy of Environment Prioritized Experience Replay (DEER)

The inherent characteristics of non-stationary environments necessitate distinct approaches for calculating the importance of a sample before and after a change. When no change point is detected, DEER uses the absolute value of the TD error as a priority. In the following, we will specifically analyze how DEER calculates the priority for pre- and post-change transitions once a change point is detected at time step T_{i-1}, respectively.

In prioritizing pre-change transitions, we focus on those with a small DoE. This is because a low DoE indicates that the transition is less affected by changes in environmental dynamics, implying its continued relevance in the new environment. The priority of a pre-change transition is calculated as:

$$p_k = 2sig(-|DoE(s_k, a_k)|) \tag{12}$$

where p_k denotes the priority of the transition (s_k, a_k, r_k, s_{k+1}) collected at time step $k\,(k \leq T_{i-1})$ and $sig(\cdot)$ represents the sigmoid function to normalize the priority.

For post-change transitions, DEER employs a hybrid metric priority sampling based on real-time density scores that integrates both TD-error and DoE. The density ratio score, as defined in Eq.10, measures the magnitude of fluctuations in the reward sequence. A high score indicates that the agent has not yet adapted to the new environment. In such cases, we prioritize post-change transitions with high DoE to facilitate rapid adaptation to the new environment; conversely, when the score is low, we prioritize post-change transitions with high TD errors to expedite policy improvement. We use dynamic weighting prioritized sampling, defined as follows:

$$\begin{aligned} p_k = & S(\widetilde{\mathbf{r}}_{te}, \widetilde{\mathbf{r}}_{rf}) \left(2sig(|DoE(s_k, a_k)|) - 1\right) + \\ & \left(1 - S(\widetilde{\mathbf{r}}_{te}, \widetilde{\mathbf{r}}_{rf})\right) \left(2sig(|TD(s_k, a_k, r_k, s_{k+1})|) - 1\right) \end{aligned} \tag{13}$$

where p_k denotes the priority of the transition (s_k, a_k, r_k, s_{k+1}) collected at time step $k\,(k > T_{i-1})$ and $S(\widetilde{\mathbf{r}}_{te}, \widetilde{\mathbf{r}}_{rf})$ denotes the density ratio score at the current time step t. To avoid overfitting caused by frequent replays of high-priority transitions, the probabilities of sampling pre- and post-transitions are normalized as follows:

$$P(k) = \frac{p_k^{\alpha}}{\sum_i p_i^{\alpha}} \tag{14}$$

Algorithm 1: Main Loop for DEER with DQN

Input: Maximum time step T, batch size N, buffer capacity L, window size m, samples per window n, maximum iteration M, detection threshold μ, exponents α and β.

Initialize Q-function Q with network parameters θ, replay buffer $\mathcal{D}$ of capacity L and binary classifier f.
Initialize state s_0 and *IsChange* = *FALSE*.
for $t = 1$ to T **do**
  Select and execute a_{t-1} according to the policy π, observe reward r_{t-1} and next state s_t.
  Store transition $(s_{t-1}, a_{t-1}, r_{t-1}, s_t)$ in $\mathcal{D}$ with maximal priority $p_t = \max_{i \in \mathcal{D}} p_i$
  if $t \geq 2m$ **then**
    Create reference set $\widetilde{\mathbf{r}}_{rf}$ and test set $\widetilde{\mathbf{r}}_{te}$.
    # Environmental dynamics change monitoring
    $d = \texttt{ScoreComputation}(f, \widetilde{\mathbf{r}}_{rf}, \widetilde{\mathbf{r}}_{te})$
    if $d \geq \mu$ **then**
      IsChange = *TRUE*
      Save the network parameters θ' at time step $t - m$ as a snapshot of the Q function Q'.
    end if
    # Priority update and experience replay
    $\Delta = \texttt{PriorityUpdate}(Q, Q', \mathcal{D}, d, \textit{IsChange})$
    # Q-function update
    $\theta \leftarrow \theta + \eta \cdot \Delta$
  end if
end for

where $\alpha \in (0, 1]$ is the degree of priority, and a smaller α indicates that DEER is closer to random uniform sampling. Since prioritized sampling alters the distribution relied upon by the Q-function for estimating expected returns, we employ importance sampling to correct bias. The weight for updating the Q-function using the transition is as follows:

$$w_k = \frac{1}{(N \cdot P(k))^{\beta} \cdot \max_i w_i} \tag{15}$$

where N denotes replay buffer size and $\beta \in (0, 1]$ denotes the bias correction degree. A larger beta indicates more full correction for prioritized sampling, requiring modifications to the Q-function updates in RL. For example, the corresponding Q-function update of the transition in Q-Learning is as follows:

$$\begin{aligned} Q(s_k, a_k) \leftarrow & Q(s_k, a_k) + \\ & w_k \cdot \eta \left[r_k + \gamma Q(s_{k+1}, a_{k+1}) - Q(s_k, a_k) \right] \end{aligned} \tag{16}$$

Algorithm 1 describes the main loop of DEER with DQN (Mnih et al. 2013). As there are no constraints on Q-function or policy updates, DEER can be applied to almost any off-policy RL algorithm (e.g., SAC (Haarnoja et al. 2018), TD3 (Fujimoto, Hoof, and Meger 2018), etc.). As shown in Algorithm 1, DEER primarily consists of three components: environment dynamics change detection, priority update and experience replay, and Q-function update. If necessary, the third component can include both Q-function and policy updates. The pseudocode for the subroutines `ScoreComputation` and `PriorityUpdate` is provided in Appendix B.

Evaluation

Non-stationary Environment Setups

We evaluate the proposed integration with two popular off-policy RL algorithms, SAC (Haarnoja et al. 2018) and TD3 (Fujimoto, Hoof, and Meger 2018), in four extensively studied robotic continuous control tasks provided by the MuJoCo gymnasium (Todorov, Erez, and Tassa 2012): Ant-v4, HalfCheetah-v4, Hopper-v4 and InvertedDoublePendulum-v4, abbreviated as ID-Pendulum in the rest of the paper.

To introduce non-stationarity, we incorporated a series of offsets for friction coefficients and joint damping coefficients. This emulates realistic scenarios where objects interact with varying environmental conditions, such as different surfaces and mediums. The offsets are determined in preliminary experiments and set to the largest feasible values while ensuring convergence of SAC with uniform sampling. For further details, refer to Appendix C.

In our experiments, the total training step is set to 5×10^5 (for ID-Pendulum task) or 1×10^6 (for other tasks). Non-stationary factors are introduced when training reaches half of the total steps. All experiments are conducted multiple times, with standard deviation represented as shaded regions. All RL-related networks have two hidden layers of 256 units each. The learning rates for both the actor and critic networks are set to 1×10^{-3}, with a discount factor of 0.99. The replay buffer has a capacity of 1×10^6. The training batch size is 256. For DEER, we employ a multi-layer perceptron-based binary classifier with two hidden layers of 100 units and a maximum iteration of 50. The time window size for environment dynamics detection is set to 500, with 10 samples within the window, each consisting of 50 data points. A complete list of hyperparameters can be found in Appendix D.

Baselines

We compared DEER with the following baselines:

(1) Prioritized Experience Replay (PER) (Schaul et al. 2015): As the seminal and most widely-used ER, PER prioritizes transitions with high TD-error.

(2) Recency-biased Prioritized Experience Replay (RB-PER) (Li, Jacobsen, and White 2021): RB-PER is designed for non-stationary environments. It constructs a bias towards less sampled transitions to ensure sufficient diversity in the sampled batches.

(3) Combined Experience Replay (CER) (Zhang and Sutton 2017): CER introduces a quasi-on-policy element by consistently including the most recent transitions in each training batch.

(4) Loss Adjusted Approximate Actor Prioritized Experience Replay (LA3P) (Saglam et al. 2023): LA3P prioritizes transitions with reliable knowledge from critics (with low TD-error) to train the actor network, thus mitigating the detrimental impact of high TD-error samples on the policy network.

Comparison Metrics

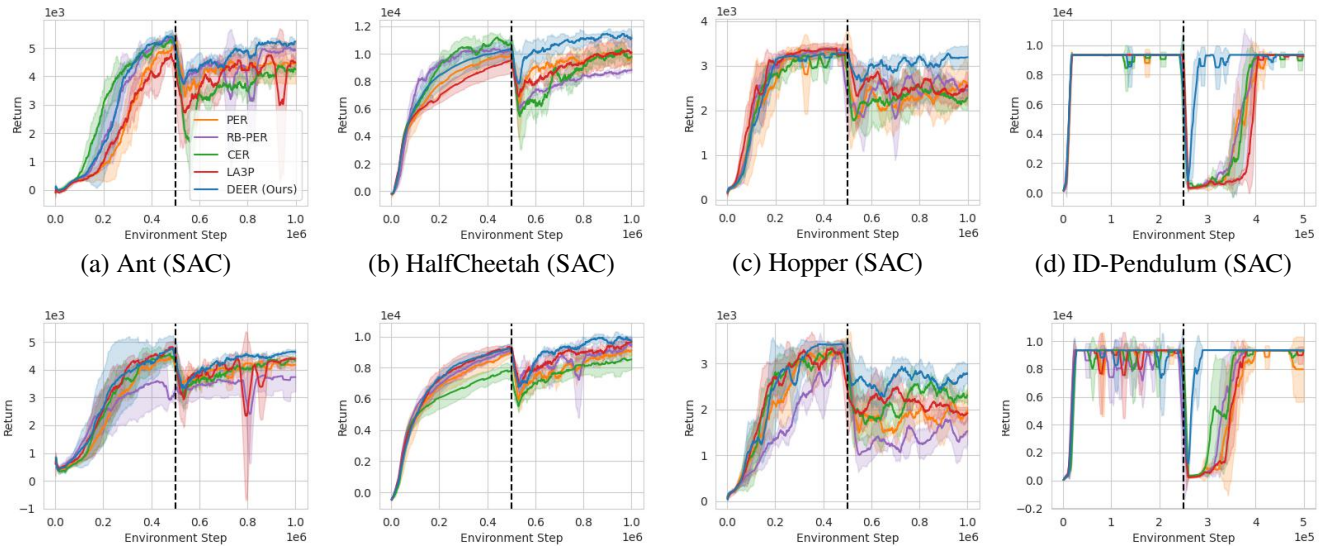
Sample Efficient Sample efficiency refers to the performance level an agent achieves with a limited number of samples. This is crucial in non-stationary environments, where sparse new samples significantly hinder rapid adaptation. We measure the sample efficiency of algorithms by comparing the return (calculated as the sum of rewards within an episode) relative to the environment steps (Yarats et al. 2021a).

Convergence Performance Convergence Performance evaluates the impact of the proposed methods on long-term training. We test the average return of the final model over 100 episodes. The ID-Pendulum task was omitted because all algorithms converged to optimal performance within 5×10^5 steps (See Figure 3(d),(h)).

Comparative Experiments

Figures 3(a)-(d) and (e)-(h) show the episode return curves when various ER methods are integrated with SAC and TD3 algorithms, respectively. Notably, in both scenarios, DEER (indicated by the red line) achieves higher overall returns compared to the baselines. Throughout various environmental changes, DEER exhibits less reduction in rewards and faster recovery rates than other methods, indicating its efficiency in adapting to dynamic environments.

Table 1 presents the convergence performance of DEER and other baselines. DEER exhibits superior convergence performance with the same number of environmental steps.



(a) Ant (SAC) (b) HalfCheetah (SAC) (c) Hopper (SAC) (d) ID-Pendulum (SAC)

(e) Ant (TD3) (f) HalfCheetah (TD3) (g) Hopper (TD3) (h) ID-Pendulum (TD3)

Figure 3: Sample efficiency comparison results on two algorithms (SAC and TD3) and four tasks. Solid lines represent the average of all three experimental runs, shaded areas indicate standard deviations. Dashed line indicates the time step at which environmental dynamics change.

Table 1: Convergence performance comparison, presented as the average return ± standard deviation.

	Environments	DEER (Ours)	PER	RB-PER	CER	LA3P
SAC	HalfCheetah	**11588.52 ± 220.93**	10255.76 ± 271.38	8926.21 ± 58.65	10689.08 ± 126.14	10257.21 ± 322.46
	Ant	**5401.60 ± 138.97**	4798.73 ± 208.68	5264.82 ± 337.17	4472.21 ± 392.73	4427.89 ± 450.75
	Hopper	**3242.50 ± 267.64**	2378.39 ± 670.14	2346.03 ± 562.33	2437.17 ± 717.00	2622.65 ± 549.05
TD3	HalfCheetah	**10136.38 ± 238.22**	9281.95 ± 179.10	9440.10 ± 376.29	8723.83 ± 511.83	9812.44 ± 225.92
	Ant	**4679.73 ± 110.98**	4263.05 ± 114.51	3785.68 ± 410.80	4301.04 ± 442.67	4425.90 ± 111.02
	Hopper	**2770.21 ± 506.02**	2127.67 ± 665.55	1666.19 ± 461.04	2244.37 ± 711.93	2163.89 ± 690.31

Across all environments and integrated with the two algorithms, the average return of DEER over the last 100 episodes consistently exceeds that of other methods, accompanied by relatively lower standard deviations. This indicates that DEER enables agents to quickly learn the characteristics of the post-change environment, resulting in more optimized and stable performance during long-term training.

Ablation Study

Effectiveness of Each Component We conducted ablation experiments in the HalfCheetah environment using SAC to evaluate the effectiveness of our two key designs: hybrid metric sampling (HM) and dynamic weighting (DW).

- **DEER w/o DW** Density ratio score is solely used for detecting environmental changes, rather than for calculating dynamic sampling weights. The priority p_k of a post-change transition(s_k, a_k, r_k, s_{k+1}) is given by:

$$p_k = 0.5 \cdot (2sig\left(|DoE\left(s_k, a_k\right)|\right) - 1) + 0.5 \cdot (2sig\left(|TD\left(s_k, a_k, r_k, s_{k+1}\right)|\right) - 1)$$

- **DEER w/o HM**
  - **TD-Error**: Sampling priority solely based on TD-error. The priority of p_k a post-change transition (s_k, a_k, r_k, s_{k+1}) is given by:

$$p_k = 2sig\left(|TD\left(s_k, a_k, r_k, s_{k+1}\right)|\right) - 1$$

  - **DoE**: Sampling priority relies solely based on DoE. The priority p_k of a post-change transition (s_k, a_k, r_k, s_{k+1}) is given by:

$$p_k = 2sig\left(|DoE\left(s_k, a_k\right)|\right) - 1$$

Figure 4 demonstrates that the implementation of DW significantly enhances algorithm performance in response to changes in environmental dynamics. In contrast, the performance of HM methods without DW is comparatively lower. This limitation arises from their reliance solely on fixed density ratio scores, which do not effectively utilize information about environmental dynamics. Both single metric methods demonstrate reduced adaptability. This underscores the importance of combining DoE and TD-error to calculate priority, which improves sample efficiency, with further enhancements achieved through the integration of dynamic weighting.

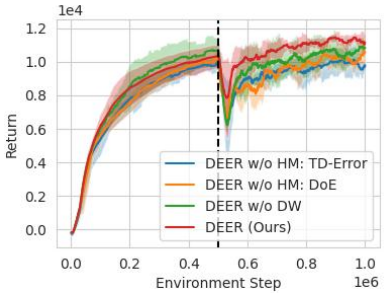


Figure 4: Ablation Result of Hybrid Metric (HM) priority and dynamic weight (DW)

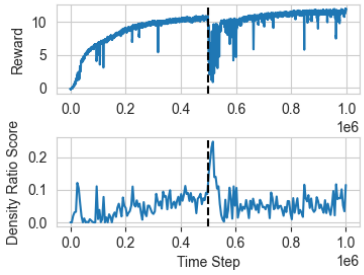


Figure 5: Reward sequence and corresponding real-time density ratio scores

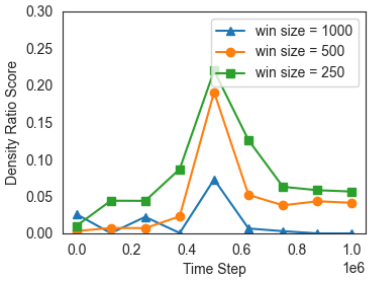


Figure 6: Effects of window sizes on density ratio scores

Effects of Levels of Non-stationarity Table 2 shows the average rewards over 100 episodes after the environmental change for the SAC+HalfCheetah task under different levels of non-stationarity. To evaluate performance under different levels of non-stationarity, we tested five environmental dynamics based on the previously mentioned settings: extreme (200% offset), standard (100% offset), mild (50% offset), and stationary (0% offset).

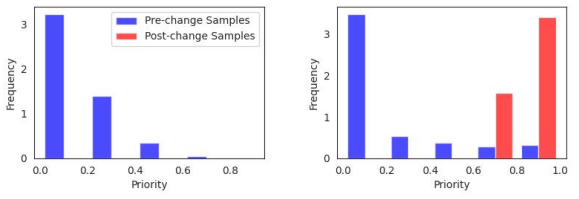
Results show DEER significantly outperforms in 200% and 100% offset scenarios, with approximately 23% higher than the best baseline at 200% offset. This demonstrates DEER's effective adaptation to environmental changes. In mild non-stationarity (50% offset), DEER maintains a slight advantage. In stationary environments (0% offset), all methods perform similarly, indicating DEER introduces no adverse effects in static conditions.

Effects of Time Window Size Figure 5 illustrates an example of reward sequence and real-time density ratio scores during SAC + DEER training in the HalfCheetah environment. When the environment changes (indicated by black dashed lines), the score exhibits a noticeable increase, indicating the detection of significant environmental dynamics.

The size of adjacent time windows is a critical hyperparameter for monitoring environmental dynamics, referring to the length of the reference and test sets in the input binary classifier. As shown in Figure 6, various window sizes 250, 500, and 1000 demonstrate their effectiveness in detecting environmental changes. Smaller window sizes exhibit stronger responsiveness to changes, resulting in more significant increases in scores and lower detection latency. However, smaller window sizes may be sensitive to subtle environmental changes, leading to unstable weights.

Discussion

Sampling Behavior Figure 7 illustrates the priority distribution of samples in the replay buffer at different training stages. Overall, the priority of post-change transitions is higher than that of pre-change ones, with this disparity being particularly pronounced during the early stages of environmental dynamics changes. With density ratio scores decrease as training progresses, The overall sample distribution tends to revert to pre-change state. This reflects DEER's strategy of balancing between pre- and post-change experience: leveraging new experience early on for rapid policy adjustment, followed by a gradual and selective reuse of old experiences to maintain stability and long-term memory.



(a) pre-change 10^4 steps (b) post-change 10^4 steps

(c) post-change 10^5 steps (d) post-change 5×10^5 steps

Figure 7: Priority distribution of pre- and post-change samples at different training stages in SAC+DEER in HalfCheetah. The frequency is normalized as density.

Algorithm Agnosticism DEER is compatible with most off-policy RL frameworks, applicable to both discrete and continuous control tasks. Our evaluations focus on continuous control tasks due to their generally higher complexity. DEER's core design is DoE, which calculates as the expected return difference of pre- and post-change Q-functions. The algorithmic agnosticism of DEER stems from its reliance solely on the action value function, a fundamental component in nearly all popular off-policy algorithms. As shown in Figure 3, it is seen that DEER's enhancement over baselines when integrated with SAC (a-d) and TD3 (e-h).

Table 2: Average episode rewards over 10^5 steps post-change for SAC+HalfCheetah under different levels of non-stationarity.

Offset	0%	50%	100%	200%
DEER(Ours)	11894.85 ± 182.36	**11789.66 ± 283.81**	**10803.87 ± 267.29**	**9856.27 ± 477.04**
PER	11834.68 ± 244.31	10266.77 ± 327.79	9168.43 ± 1345.23	5204.22 ± 763.55
RB-PER	11754.29 ± 202.61	10506.37 ± 364.37	8397.97 ± 898.58	7332.36 ± 318.96
CER	**11975.13 ± 256.47**	9428.88 ± 199.17	9409.88 ± 274.54	8043.76 ± 467.35
LA3P	11895.44 ± 267.72	11024.15 ± 787.15	10080.97 ± 620.36	6614.79 ± 285.27

Table 3: Runtime of each component of DEER, averaged over 10^6 steps of SAC + DEER in HalfCheetah.

Component		Runtime per step
RL networks update		45.82ms
Priority updates		0.51ms
Score computation	window size = 100	0.91ms
	window size = 1000	0.54ms
	window size = 2000	0.42ms

Time Overhead The additional time overhead introduced by DEER mainly involves (1) Sample priority updates: By utilizing SumTree, a binary tree structure storing priorities, we achieve sampling and weight updates within $\mathcal{O}(\log n)$ time complexity. (2) Density ratio scores Computation: The computation time of density ratio score strongly depends on the size of the adjacent time window. A smaller window size leads to more frequent score updates and longer computation time. In practice, we find a window size of $\sim$1000 is sufficient for most tested tasks.

Table 2 presents the runtime of each component, averaged over each training step. The experiment runs on an NVIDIA GeForce RTX 2080 Ti 11GB GPU. It is evident that none of the aforementioned components exhibit significant additional time overhead. Compared to the parameter update time of the RL networks, the additional overhead is marginal ($\sim$3%).

Related Work

Non-stationary environments are essentially composed of several stationary stages of environmental dynamics. Consequently, some reinforcement learning studies in non-stationary environments focus on the decomposition of MDPs (Ouhamma, Basu, and Maillard 2023). Memory Graph combines Transformer-based architecture with the decomposition of the observation space to improve the sample efficiency of RL (Loynd et al. 2020). Cause-Effect Modeling Agent (CEMA) utilizes a structured world model to decompose the latent state space and modeling of sparse interactions (Zholus, Ivchenkov, and Panov 2022). AdaRL decomposes factor representations under the latent states, rewards, and action domains, and adjusts policies using transitions only within the target domain (Huang et al. 2021). Other studies focus on directly learning latent representations to model non-stationary environments. Lifelong Latent Actor-Critic (LILAC) achieves lifelong learning by jointly learning compact representations of MDPs and maximum entropy policies (Xie, Harrison, and Finn 2020). Zero-shot adaptation to Unknown Systems (ZeUS) uses the block contextual MDP architecture to directly model non-stationary features (Sodhani et al. 2022). Combined Reinforcement via Abstract Representations (CRAR) builds a compact state abstraction to represent environmental features so that agents can adapt to different environments through transfer learning (François-Lavet et al. 2019).

Experience replay stores and resamples past experiences to enhance sample efficiency and learning stability (Lin 1992). While this method could potentially accelerate agent adaptation to non-stationary environments by enhancing policy convergence, this aspect remains to be explored. The most common proportional sampling method involves prioritizing transitions based on TD-error, which has demonstrated performance improvements in many RL tasks (Sun, Zhou, and Li 2020; Li, Qian, and Song 2024). Other studies utilize alternative metrics as sampling priorities. Prioritized Sequence Experience Replay (PSER) assigns high priority to significant transitions and increases the priority of previous transitions that resulted in significant transitions (Brittain et al. 2019). Remember and Forget Experience Replay (ReF-ER) classifies transitions as "near-policy" or "far-policy" based on the ratio of the current policy to the past policy, and only samples near-policy transitions (Novati and Koumoutsakos 2019). Attentive Experience Replay (AER) is a sampling method that prioritizes transitions in the replay buffer based on their similarity to the current state (Sun, Zhou, and Li 2020). Hindsight Experience Replay (HER) (Andrychowicz et al. 2017) and its variants (Luo et al. 2023; Sayar et al. 2024) improve sample efficiency in sparse reward tasks by relabeling transitions to generate sufficient reward feedback. Recency-biased Prioritized Experience Replay(RB-PER) (Li, Jacobsen, and White 2021) assigns high priority to transitions that have been sampled less frequently, ensuring diversity in sampling and thereby adapting to non-stationary environments.

Conclusion

We propose DEER as a solution to the sample efficiency problem in non-stationary environments. DEER adopts a hybrid metric prioritization approach that dynamically adjusts sampling weights based on changes in environmental dynamics. DEER enhances sample efficiency in non-stationary environments by prioritizing valuable samples measured by DoE within the current environment. Experiment results validate the effectiveness of the DEER across different non-stationary tasks.

References

Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Pieter Abbeel, O.; and Zaremba, W. 2017. Hindsight experience replay. *Advances in neural information processing systems*, 30.

Brittain, M.; Bertram, J.; Yang, X.; and Wei, P. 2019. Prioritized sequence experience replay. *arXiv preprint arXiv:1905.12726*.

Cao, X.; Wan, H.; Lin, Y.; and Han, S. 2019. High-value prioritized experience replay for off-policy reinforcement learning. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, 1510–1514. IEEE.

François-Lavet, V.; Bengio, Y.; Precup, D.; and Pineau, J. 2019. Combined reinforcement learning via abstract representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 3582–3589.

Fujimoto, S.; Hoof, H.; and Meger, D. 2018. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, 1587–1596. PMLR.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, 1861–1870. PMLR.

Hillman, J.; and Hocking, T. D. 2021. Optimizing roc curves with a sort-based surrogate loss function for binary classification and changepoint detection. *arXiv preprint arXiv:2107.01285*.

Huang, B.; Feng, F.; Lu, C.; Magliacane, S.; and Zhang, K. 2021. Adarl: What, where, and how to adapt in transfer reinforcement learning. *arXiv preprint arXiv:2107.02729*.

Li, D.; Jacobsen, A.; and White, A. 2021. Revisiting experience replay in non-stationary environments. In *International conference on autonomous agents and multiagent systems*.

Li, H.; Qian, X.; and Song, W. 2024. Prioritized experience replay based on dynamics priority. *Scientific Reports*, 14(1): 6014.

Lin, L.-J. 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8: 293–321.

Loynd, R.; Fernandez, R.; Celikyilmaz, A.; Swaminathan, A.; and Hausknecht, M. 2020. Working memory graphs. In *International conference on machine learning*, 6404–6414. PMLR.

Luo, Y.; Wang, Y.; Dong, K.; Zhang, Q.; Cheng, E.; Sun, Z.; and Song, B. 2023. Relay Hindsight Experience Replay: Self-guided continual reinforcement learning for sequential object manipulation tasks with sparse rewards. *Neurocomputing*, 557: 126620.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Novati, G.; and Koumoutsakos, P. 2019. Remember and forget for experience replay. In *International Conference on Machine Learning*, 4851–4860. PMLR.

Ouhamma, R.; Basu, D.; and Maillard, O. 2023. Bilinear exponential family of MDPs: frequentist regret bound with tractable exploration & planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 9336–9344.

Padakandla, S. 2021. A survey of reinforcement learning algorithms for dynamically varying environments. *ACM Computing Surveys (CSUR)*, 54(6): 1–25.

Rahimi-Kalahroudi, A.; Rajendran, J.; Momennejad, I.; van Seijen, H.; and Chandar, S. 2023. Replay Buffer with Local Forgetting for Adapting to Local Environment Changes in Deep Model-Based Reinforcement Learning. In *Conference on Lifelong Learning Agents*, 21–42. PMLR.

Saglam, B.; Mutlu, F. B.; Cicek, D. C.; and Kozat, S. S. 2023. Actor prioritized experience replay. *Journal of Artificial Intelligence Research*, 78: 639–672.

Sayar, E.; Vintaykin, V.; Iacca, G.; and Knoll, A. 2024. Hindsight Experience Replay with Evolutionary Decision Trees for Curriculum Goal Generation. In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*, 3–18. Springer.

Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. 2015. Prioritized Experience Replay. *arXiv preprint arXiv:1511.05952*.

Sodhani, S.; Meier, F.; Pineau, J.; and Zhang, A. 2022. Block contextual mdps for continual learning. In *Learning for Dynamics and Control Conference*, 608–623. PMLR.

Sun, P.; Zhou, W.; and Li, H. 2020. Attentive experience replay. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 5900–5907.

Todorov, E.; Erez, T.; and Tassa, Y. 2012. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5026–5033. IEEE.

Tu, D.; Li, Y.; and Cai, Y. 2023. A new perspective on detecting performance decline: A change-point analysis based on Jensen-Shannon divergence. *Behavior Research Methods*, 55(3): 963–980.

Xie, A.; Harrison, J.; and Finn, C. 2020. Deep reinforcement learning amidst lifelong non-stationarity. *arXiv preprint arXiv:2006.10701*.

Yarats, D.; Fergus, R.; Lazaric, A.; and Pinto, L. 2021a. Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv preprint arXiv:2107.09645*.

Yarats, D.; Zhang, A.; Kostrikov, I.; Amos, B.; Pineau, J.; and Fergus, R. 2021b. Improving sample efficiency in model-free reinforcement learning from images. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 10674–10681.

Zhang, S.; and Sutton, R. S. 2017. A deeper look at experience replay. *arXiv preprint arXiv:1712.01275*.

Zholus, A.; Ivchenkov, Y.; and Panov, A. 2022. Factorized world models for learning causal relationships. In *ICLR2022 Workshop on the Elements of Reasoning: Objects, Structure and Causality*.

Appendix

Appendix A: Density Ratio Score Function

In this section, we describe how to compute the density ratio score function using Jensen-Shannon divergence (JS divergence). JS divergence is symmetric, meaning that regardless of which one is taken as the test or reference set between two adjacent time windows, the result obtained is the same. Note that $P_{te}(\mathbf{r}) + P_{rf}(\mathbf{r}) = 1$ in the binary classification, so the density ratio score function can be derived as the following discrete case:

$$
\begin{aligned}
S(\widetilde{\mathbf{r}}_{te}, \widetilde{\mathbf{r}}_{rf}) &= JS\left(P_{te}(\widetilde{\mathbf{r}}_{te}) \parallel P_{rf}(\widetilde{\mathbf{r}}_{rf})\right) \\
&= \frac{1}{2}\int P_{te}(\mathbf{r}) \log \frac{2P_{te}(\mathbf{r})}{P_{te}(\mathbf{r}) + P_{rf}(\mathbf{r})} d\mathbf{r} \\
&+ \frac{1}{2}\int P_{rf}(\mathbf{r}) \log \frac{2P_{rf}(\mathbf{r})}{P_{te}(\mathbf{r}) + P_{rf}(\mathbf{r})} d\mathbf{r} \\
&= \frac{1}{2n_{te}} \sum_{\mathbf{r}\in\widetilde{\mathbf{r}}_{te}} \log \frac{2P_{te}(\mathbf{r})}{P_{te}(\mathbf{r}) + P_{rf}(\mathbf{r})} \\
&+ \frac{1}{2n_{rf}} \sum_{\mathbf{r}\in\widetilde{\mathbf{r}}_{rf}} \log \frac{2P_{rf}(\mathbf{r})}{P_{te}(\mathbf{r}) + P_{rf}(\mathbf{r})} \\
&= \frac{1}{2n_{te}} \sum_{\mathbf{r}\in\widetilde{\mathbf{r}}_{te}} \log 2P_{te}(\mathbf{r}) + \frac{1}{2n_{rf}} \sum_{\mathbf{r}\in\widetilde{\mathbf{r}}_{rf}} \log 2P_{rf}(\mathbf{r}) \\
&= \log 2 + \frac{1}{2n_{te}} \sum_{\mathbf{r}\in\widetilde{\mathbf{r}}_{te}} \log P_{te}(\mathbf{r}) \\
&+ \frac{1}{2n_{rf}} \sum_{\mathbf{r}\in\widetilde{\mathbf{r}}_{rf}} \log P_{rf}(\mathbf{r}) \\
&= \log 2 + \frac{1}{2n_{te}} \sum_{\mathbf{r}\in\widetilde{\mathbf{r}}_{te}} \log P(\mathbf{r} \mid f(\mathbf{r}) = 1) \\
&+ \frac{1}{2n_{rf}} \sum_{\mathbf{r}\in\widetilde{\mathbf{r}}_{rf}} \log P(\mathbf{r} \mid f(\mathbf{r}) = 0)
\end{aligned}
$$

where $JS(\cdot \parallel \cdot)$ denotes the JS divergence.

Appendix B: Algorithm Pseudocode

In this section, we describe the pseudocode for the subroutines `ScoreComputation` in Algorithm 2 and `PriorityUpdate` in Algorithm 3, respectively.

Algorithm 2: `ScoreComputation`

Input: Binary classifier f, reference set $\widetilde{\mathbf{r}}_{rf}$, test set $\widetilde{\mathbf{r}}_{te}$, samples per window n, detection threshold μ, and maximum iteration M.
Output: Density ratio score d.

$d \leftarrow 0,\ k \leftarrow 0$
while $k < M$ and **not** Convergence **do**
  $\widetilde{\mathbf{r}}_{rf}^{train},\ \widetilde{\mathbf{r}}_{rf}^{inf} \leftarrow \text{RandomSplit}\left(\widetilde{\mathbf{r}}_{rf},\ \text{samples} = n\right)$
  $\widetilde{\mathbf{r}}_{te}^{train},\ \widetilde{\mathbf{r}}_{te}^{inf} \leftarrow \text{RandomSplit}\left(\widetilde{\mathbf{r}}_{te},\ \text{samples} = n\right)$
  $f.\text{train}\left(\widetilde{\mathbf{r}}_{rf}^{train},\ \widetilde{\mathbf{r}}_{te}^{train}\right)$
  $P_{rf},\ P_{te} \leftarrow f.\text{inference}\left(\widetilde{\mathbf{r}}_{rf}^{inf},\ \widetilde{\mathbf{r}}_{te}^{inf}\right)$
  $d \leftarrow d + \log 2 + \frac{1}{2}P_{rf} + \frac{1}{2}P_{te}$
  $k \leftarrow k + 1$
end while
$d \leftarrow \frac{d}{k}$
return d

Algorithm 3: `PriorityUpdate`

Input: Q-function Q with network parameters θ, the snapshot of the Q function Q', replay buffer $\mathcal{D}$, density ratio score d, environmental dynamics monitor flag *IsChange*, exponents α and β, and batch size N.
Output: Q-function update weights Δ.

$\Delta \leftarrow 0$
for $j = 1$ to N **do**
  Sampling a transition (s_j, a_j, r_j, s_{j+1}) from the replay buffer $\mathcal{D}$: $j \sim \frac{p_j^\alpha}{\sum_i p_i^\alpha}$.
  $w_j \leftarrow \frac{1}{(N \cdot P(j))^\beta \cdot \max_{i\in\mathcal{D}} w_i}$
  $\delta^{TD} \leftarrow r_j + \gamma \max_{a'} Q(s_{j+1}, a') - Q(s_j, a_j)$
  if *IsChange == TRUE* **then**
    $\delta^{DoE} \leftarrow Q(s_j, a_j) - Q'(s_j, a_j)$
    if (s_j, a_j, r_j, s_{j+1}) is a pre-change transition **then**
      $p_j \leftarrow 2sig\left(-\left|\delta^{DoE}\right|\right)$
    else
      $p_j \leftarrow d\left(2sig\left(\left|\delta^{DoE}\right|\right) - 1\right) + (1-d)\left(2sig\left(\left|\delta^{TD}\right|\right) - 1\right)$
    end if
  else
    $p_j \leftarrow \left|\delta^{TD}\right|$
  end if
  $\Delta \leftarrow \Delta + w_j \cdot \delta^{TD} \cdot \bigtriangledown_\theta Q(s_j, a_j)$
end for
return Δ

Appendix C: Environment Dynamics Modifications

In this section, we outline the specific adjustments made to the environment dynamics of the continuous control tasks in order to introduce non-stationarity. The friction coefficients are expressed as components on the x, y, and z axes respectively.

1. **Ant-v4:**
   - **Description:** Quadrupedal robot navigation task with rewards of forward progress and balance.
   - **Pre-change environment dynamics:**
   – Global geometry friction coefficients: 1 0.5 0.5
   – Joint damping coefficient: 1
   - **Post-change environment dynamics:**
   – Global geometry friction coefficients: 1.3 0.8 0.8
   – Joint damping coefficient: 1.2
2. **HalfCheetah-v4:**
   - **Description:** Two-legged robot locomotion task with rewards for achieving fast forward motion while maintaining stability.
   - **Pre-change environment dynamics:**
   – Global geometry friction coefficients: 0.4 0.1 0.1
   - **Post-change environment dynamics:**
   – Global geometry friction coefficients: 0.8 0.5 0.5
3. **Hopper-v4:**
   - **Description:** Single-legged robot hopping task with rewards based on forward distance.
   - **Pre-change environment dynamics:**
   – Torso, thigh, leg, and foot joint geometry friction coefficients: 0.9 0.9 0.9 2.0
   - **Post-change environment dynamics:**
   – Torso, thigh, leg, and foot joint geometry friction coefficients: 0.5 0.5 0.5 1.5
4. **InvertedDoublePendulum-v4:**
   - **Description:** Double pendulum stabilization task with rewards for maintaining the inverted position.
   - **Pre-change environment dynamics:**
   – Joint damping coefficient: 0.05
   - **Post-change environment dynamics:**
   – Joint damping coefficient: 1

Appendix D: Table of Hyperparameters

In this section, we present a complete list of hyperparameters used in experiments.

Table 4: The hyperparameters used in in experiments

Hyperparameter	Value
Training Step	5×10^5 for Pendulum task 1×10^6 for other tasks
Buffer Size	1.0×10^6
Hidden Layer Sizes	[256, 256]
Actor Learning Rate	1.0×10^{-3}
Critic Learning Rate	1.0×10^{-3}
Discount Factor	0.99
Soft Update Coefficient	0.005
Start Timesteps	10000
Epochs	200
Steps per Epoch	5000
Steps per Data Collection	1
Batch Size	256
SAC Hyperparameter	
Alpha	0.2
Alpha Learning Rate	3.0×10^{-4}
TD3 Hyperparameter	
Exploration Noise	0.1
Policy Noise	0.2
Noise Clip	(-0.5, 0.5)
Actor Update Frequency	2
Training Instances	1
ER-related Hyperparameter	
PER Alpha	0.6
PER Beta	0.4
RB-PER Scaling Factor	0.05
LA3P Uniform Fraction	0.5
DEER Alpha	0.6
DEER Beta	0.4
DEER Classifier Hidden Layer Sizes	[100, 100]
DEER Classifier Max Iteration	50
DEER Time Window Size	500
DEER Samples per Window	10
DEER Data Points per Sample	50
DEER Detection Threshold	{0.2,0.4,0.6}