DePER: Decoupled Multi-Scheme Prioritized Experience Replay for Sample-Efficient Deep Reinforcement Learning

Diyuan Shi
College of Computer Science and Technology
Zhejiang University
Hangzhou, China
shidiyuan@westlake.edu.cn

Zifeng Zhuang
School of Engineering
Westlake University
Hangzhou, China
zhuangzifeng@westlake.edu.cn

Donglin Wang†
School of Engineering
Westlake University
Hangzhou, China
wangdonglin@westlake.edu.cn

***Abstract*—Prioritized Experience Replay (PER) is a widely adopted technique in Deep Reinforcement Learning (RL) and could significantly improve the sample efficiency and adaptation ability of RL methods. Extensive research effort has been put into designing novel prioritization schemes, such as learning error, recency and visitation count. Unfortunately, given these promising and effective methods, few works have focused on how to 'integrate' multiple prioritization schemes easily and straightforwardly. And the current approach to integrate multiple prioritization schemes is still weighted sum which suffers shortcomings in both practice and theory. As RL methods are becoming more complex, we also need PER method that scales better. Hence in this work, we propose DePER, which performs independent and decoupled PER and avoids the drawbacks in scaling weighted sum PER. We conduct theoretical analysis and detailed experiments (against heavily tuned weight sum PER) to demonstrate DePER requires less tuning effort, offers broader applicability and could obtain better performance in challenging tasks.**

***Index Terms*—deep reinforcement learning, prioritized experience replay**

I. Introduction

Reinforcement Learning (RL) is a machine-learning field aiming to solve sequential decision-making problems and has received significant advancements in recent years. Despite its successful applications in Robotics [1, 2], Gaming AI [3, 4] and Large Language Models [5, 6], the sample efficiency and adaptation ability of RL algorithms still lag behind human intelligence, hindering its further adoption [7, 8, 9].

Prioritized Experience Replay (PER) [10] studies how to select transition samples stored in replay buffer for RL training and has achieved promising results in improving the sample-efficiency and adaptation ability of RL agents [11]. Compared to uniform sampling, PER samples transitions based on their prioritization values: In its general form, PER with i prioritization schemes performs sampling with the following formula[1]:

$$\mathcal{P}_n = \frac{\exp[\tau \sum_i w_i p_i^n]}{\sum_n \exp[\tau \sum_i w_i p_i^n]} \tag{1}$$

where $\mathcal{P}_n$ is the final prioritization value for nth transition in replay buffer, w_i is the weight for ith prioritization scheme p_i, τ is the sampling temperature, p_i^n is the ith prioritization value for nth transition. Equation (1) is the Weighted-Sum PER (**WS-PER**) commonly used in existing literature.

The intuition of PER is that the transitions in the replay buffer are not equally important for algorithm training. Hence various prioritization schemes are designed to evaluate the importance of transitions from different perspectives and sample more important transitions more frequently. Common design choices for p could be temporal difference (td) error [10], model learning errors [12, 11] or recency [13], each of which caters a specific scenario: td error focuses on Q function learning, modeling error benefits learning of dynamics model and reward model, sampling recent transitions could improve sample efficiency while sampling old transitions may mitigate *catastrophic forgetting* of old behaviors.

It is then straightforward to think that by employing multiple prioritization schemes simultaneously, we could retain and combine their positive effects (e.g., sample efficiency, adaptation, unforgetting) to further improve RL methods. This is the *multi-scheme PER* that we study in this work. Unfortunately, WS-PER has several drawbacks preventing it from being a good candidate for multi-scheme PER: 2) High Tuning Requirement; 2) Poor Applicability and 3) Coupled Calculation (explained in details in later section). Hence *how to design a scalable PER method for multi-scheme PER* is still a remaining question.

Inspired by the idea of Stratified Sampling (SS), we propose Decoupled Prioritized Experience Replay (**DePER**), which performs independent prioritized sampling using each prioritization scheme, then concatenates these minibatches to form the final batch for algorithm update. Through this simple yet effective modification, DePER could readily scale to many prioritization schemes compared to WSPER. In summary, the contributions of our work are:

[1]In the literature, a non-exponential form of $\mathcal{P}_n$ is also used. But we find the exponential form with *adaptive temperature* (details in later) would better balance the sampling process and serve as a stronger and more general baseline.

1) We propose to study multi-scheme PER which integrates multiple prioritization schemes together to enjoy their benefits simultaneously. Given the abundance of existing single-scheme PER methods, we believe our work could further advance the performance of RL methods.
2) We analyze the drawbacks of WS-PER, the *de facto* method for multi-scheme PER and propose DePER as a simple yet effective method to replace it.
3) We perform extensive experiments in specially designed, challenging environments and tasks to demonstrate DePER's advantages to heavily tuned WS-PER. And we will publish our source code upon paper acceptance.

II. Related Work

Deep Reinforcement Learning. DRL combines classical Reinforcement Learning (RL) methods with Deep Learning (DL) techniques. As a pioneering work, DQN [3] firstly combined the Neural Network (NN) with Q-Learning in RL and obtained impressive performance on Atari Games [14]. Then DDPG [15], TRPO [16] and PPO [17] tackled continuous control and obtained strong performances. With the integration of entropy-maximization theory, SAC [18] achieved impressive sample efficiency compared to other model-free RL baselines. On another line, Model-Based RL (MbRL) offers better sample efficiency via learning of dynamics model and/or reward model. Many works have been proposed: Dyna [19] and Dreamer [20] leveraged dynamics and reward model to generate new transition samples for data augmentation. TD-MPC [21, 9] utilized the learned models for planning purposes and obtained impressive result on continuous control tasks. Trajectory Transformer [22] explored the combination of dynamics model and Transformer [23] architecture for better prediction consistency.

Prioritized Experience Replay. PER is also an effective method to improve the sample efficiency of DRL algorithms. PER-DQN [10] firstly replaced the uniform sampling in DQN with prioritized sampling according to temporal difference error and achieved significantly improved sample efficiency on Atari Games [14]. Afterwards, several works have studied the underlying principles of PER [24, 25]. Then tremendous effort has been put into designing novel and effective prioritization schemes for various benefits: By sampling recent transitions more often, ERE [13] achieved significant performance gain on SAC; MaPER [12] integrated learning loss of model-related components with td error and obtained better performance on SAC; CR [11] also utilized the model-based prioritization schemes but targeted on better adaptation ability towards environment change; ReLo [26] proposed to prioritize the samples by their learning potential, which is reflected by whether their loss could be further reduced. In this work, we focus on multi-scheme PER which to our knowledge, is not yet discussed in previous literature and orthogonal towards most of these methods.

III. Background

Reinforcement Learning. Reinforcement Learning (RL) aims to solve sequential decision-making problem which is formulated as a Markovian Decision Process (MDP): $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{R}, d, \gamma, \mathcal{S}_0\}$ where $\mathcal{S}$ is the set of all *states*, $\mathcal{A}$ is the set of all *actions*, $\mathcal{R} : (s_t, a_t) \to \mathbb{R}$ is the reward function that gives a scalar value for given state action pair, $d : s_{t+1} \sim d(s_t, a_t)$ is the transition model that gives the next state given current state action, γ is the discount factor and $\mathcal{S}_0$ is the distribution of initial states. RL is to find a policy function π that maximizes the expected return in $\mathcal{M}$:

$$\pi = \underset{\pi \sim \Pi}{\operatorname{argmax}}\, \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t)\right]$$

Model-based RL. State-of-the-art model-based RL (MbRL) algorithms (e.g., TDMPC2 [9]) have achieved impressive performance via learning of dynamics model and reward models:

$$\begin{aligned}
&\textit{Encoder} && \mathbf{z_t} = h_\theta(\mathbf{s_t}) \\
&\textit{Latent dynamics} && \mathbf{z}_{t+1} = d_\theta(\mathbf{z}_t, \mathbf{a}_t) \\
&\textit{Reward} && \hat{r}_t = R_\theta(\mathbf{z}_t, \mathbf{a}_t) \\
&\textit{Terminal value} && \hat{q}_t = Q_\theta(\mathbf{z}_t, \mathbf{a}_t) \\
&\textit{Policy prior} && \hat{\mathbf{a}}_t = \pi_\theta(\mathbf{z}_t)
\end{aligned}$$

where $\mathbf{s}_t$, $\mathbf{a}_t$, $\mathbf{z}_t$, $\mathbf{z}_{t+1}$, $\hat{\mathbf{a}}_t$, $\hat{r}_t$, $\hat{q}_t$ are state, action, latent representation, next-step latent representation, predicted action vectors, predicted transition reward and predicted action value, respectively. h_θ, d_θ, R_θ, Q_θ, π_θ are neural networks modeling *encoder*, *latent dynamics*, *reward model*, *action value function* and *policy* respectively (θ is omitted in below for brevity). Then the optimization objectives are:

$$\textit{Consistency Loss} \quad \mathcal{L}_c = \mathbb{E}\left[\sum_{t=0}^{H} \lambda^t ||\mathbf{z}_{t+1} - \text{sg}(h(\mathbf{s}_{t+1}))||^2\right] \tag{2}$$

$$\textit{Value Loss} \quad \mathcal{L}_v = \mathbb{E}\left[\sum_{t=0}^{H} \lambda^t \text{CE}(\hat{q}_t, Q^t_{\text{target}})\right] \tag{3}$$

$$\textit{Reward Loss} \quad \mathcal{L}_r = \mathbb{E}\left[\sum_{t=0}^{H} \lambda^t \text{CE}(\hat{r}_t, r_t)\right] \tag{4}$$

$$\textit{Policy Loss} \quad \mathcal{L}_\pi = \mathbb{E}\left[\sum_{t=0}^{H} \lambda^t [-Q(\mathbf{z}_t, \pi(\mathbf{z}_t)) - \beta\mathcal{H}(\pi(\cdot|\mathbf{z}_t))]\right] \tag{5}$$

where λ is decaying factor, H is the horizon, sg is stop gradient operator, CE is the cross entropy loss used in Farebrother et al. [5], β is a hyperparameter, $\mathcal{H}$ represents entropy of stochastic policy and $Q^t_{\text{target}} := r_t + \gamma Q_{\bar{\theta}}\big(s_{t+1}, \pi(s_{t+1})\big)$. In our experiments, we set $H = 3$, $\lambda = 0.99$ and $\beta = 10^{-4}$

In this work, we choose the planning-based MbRL method as our backbone for 2 reasons: 1) MbRL methods usually obtain stronger sample efficiency than model-free counterparts hence it's important to pursue further improvement on it; 2)

MbRL usually contains several interplaying NN components and learning losses, making it a good testbed for studying multi-scheme PER.

Prioritized Experience Replay. In its general form, (WS-PER takes the following formula: $\mathcal{P}_n = \frac{\exp[\tau \sum_i w_i p_i^n]}{\sum_n \exp[\tau \sum_i w_i p_i^n]}$ where $\mathcal{P}_n$ is the final prioritization value for nth transition in replay buffer, w_i is the weight for ith prioritization schemes p_i, τ is the sampling temperature and p_i^n is the ith prioritization value for nth transition. Common choices for p are td error: $p = \delta(s,a,r,s') = (r + \gamma Q_{\bar{\theta}}(s', \pi(s')) - Q_\theta(s,a))^2$, modeling error: $p = (d_\theta(s,a) - s')^2$, recency: $p = \rho(s,a,s')$ or visitation count: $p = N(s,a,s')$.

IV. Motivation

In this section, we provide explanations on WS-PER's drawbacks when used for mutli-scheme PER.

1) **High Tuning Requirement**. To demonstrate the difficulty in tuning WS-PER's $\mathbf{w} = [w_1, w_2, ..., w_n]$, we firstly show an empirical example. Specifically, we run the MbRL backbone with uniform sampling in our `Hopper` environment for 1×10^6 frames and collect 4 losses: *consistency loss*, *reward loss*, *value loss* and *policy loss*. The results are shown in Figure 1.

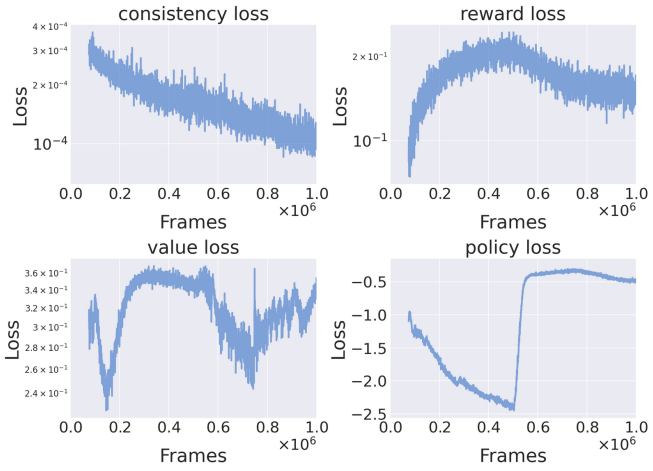

Fig. 1: The varying losses of training in `Hopper`. The y axis is in *log* scale except for policy loss.

From the result, it's clear that these 4 losses have drastically different scale and variance in the course of online training. This causes 2 troubles for tuning $\mathbf{w}$: 1) we need to run the experiment first then manually determine each weight for the corresponding loss, which is tedious, 2) in the course of online training, these 4 losses also vary significantly. With a constant $\mathbf{w}$, it's hard to balance the 4 losses all the time and result in reduced PER [2] in certain timing.

2) **Poor Applicability**. It could be non-trivial to integrate new prioritization schemes via WS-PER. For example, *policy loss* $\left(\mathcal{L}_\pi = -Q\big(s, \pi(a)\big)\right)$ in DRL could have significantly varying value range and hard to be balanced with td error which is usually non-negative.

3) **Coupled Calculation**. Individual prioritization schemes are coupled together by summation operation: Consider a 3-prioritization integration case, a transition with a *larger-than-average* p_2 and *lower-than-average* p_1 and p_3 may still have a lower-than-average $\mathcal{P}$ hence gets ignored by WS-PER. But for the purpose of p_2, it would be beneficial to sample this transition. And this effect becomes more significant with more prioritization schemes.

At the first glance, the issue of WS-PER could be mitigated with rather simple tricks like *Batch Normalization* or *Tracing Normalizer*. However, these methods are often *inadequate* for fully addressing the problems: 1) Batch Normalization normalizes priorities using only statistics from the current mini-batch, causing a locality issue where normalized values are not comparable across training steps; 2) Tracing Normalizer tracks global scales over time to balance schemes, but still relies on extra hyperparameters and tuning overhead.

V. Decoupled Prioritized Experience Replay

We introduce our method, DePER in this section. An illustrative diagram is in Figure 2. Given i prioritization schemes, the WS-PER integrates them in '*prioritization*' level: $\mathcal{P}_n = \frac{\exp[\tau \sum_i w_i p_i^n]}{\sum_n \exp[\tau \sum_i w_i p_i^n]}$. While `DePER` integrates them in '*batch*' level: It first samples sub-minibatches using each prioritization scheme independently, then concatenates these samples in batch level to form the final mini-batch:

$$\mathcal{S}_i \sim \frac{\exp[\tau p_i^n]}{\sum_n \exp[\tau p_i^n]} \tag{6}$$

$$\mathcal{S}_{\text{final}} = [\mathcal{S}_1, \mathcal{S}_2, .., \mathcal{S}_i] \quad \text{s.t.} \sum_i |\mathcal{S}_i| = B \tag{7}$$

where $\mathcal{S}_i$ represents sub-samples for ith prioritization scheme, $\mathcal{S}_{\text{final}}$ is the final mini batch, $|\cdot|$ computes the amount of samples, $[...]$ concatenates vectors in batch axis and B is the scalar batch size. Analogous to stratified sampling, DePER successfully transform the original weight vector $\mathbf{w}$ in WS-PER to a new budget vector $\mathbf{S} = [|\mathcal{S}_1|, |\mathcal{S}_2|, ..., |\mathcal{S}_i|]$ which has the following benefits:

1) **Low Tuning Requirement**. *The tuning effort of* $\mathbf{S}$ *is significantly less than* $\mathbf{w}$. Actually, splitting the batch size B evenly for $\mathbf{S}$ would already be a good starting point. But for $\mathbf{w}$, setting it to be all-one vector would likely result in reduced situation as the scale of each prioritization could be drastically different. Also for budget-based $\mathbf{S}$, all prioritization schemes are guaranteed to contribute to the final minibatch in a *predictable* manner regardless their variations during training process.

2) **Broad Applicability**. *New Prioritization Sampling could be seamlessly integrated using DePER*. Since

[2] Certain prioritization scheme fails to contribute to the final minibatch.

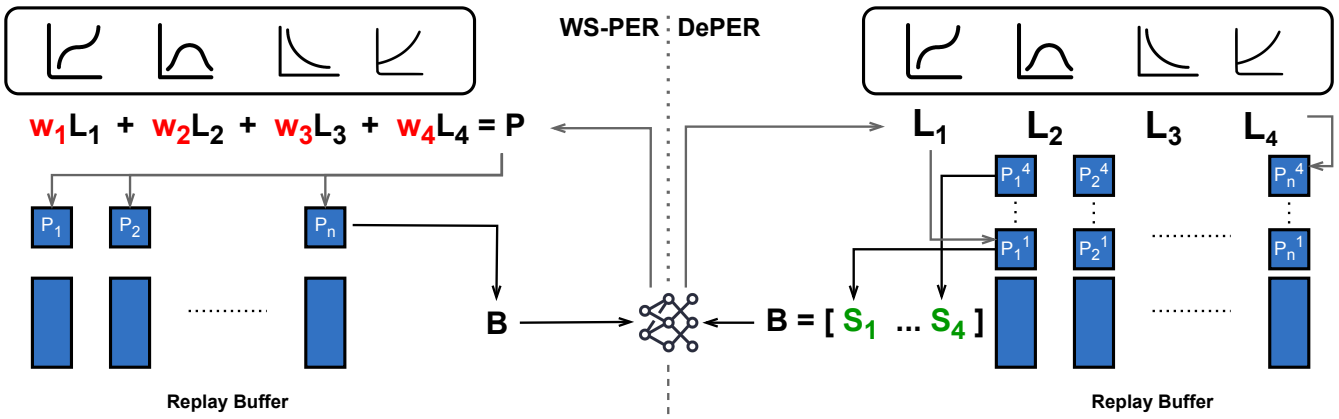

Fig. 2: The framework of DePER. L_i are varying losses used for prioritization calculation, P_i^n are prioritization values of ith scheme for nth transition, B is the final mini batch and $[S_1...S_4]$ represents vector concatenation in batch axis. After each udpate, the new losses of each transition would be updated back to replay buffer.

we're performing independent PER for each prioritization scheme, the scale and variance of various prioritization schemes would not conflict and the requirement for a given prioritization scheme to be applicable in DePER is *total ordering*. For WS-PER, the scale, variance and signature of *every* prioritization scheme would have to be balanced properly otherwise it may influence the whole PER sampling unexpectedly.

3) **Decoupled Calculation**. *Performing budget-based integration via* $\mathbf{S}$ *is more stable than* $\mathbf{w}$. Firstly, since each prioritization scheme is processed independently, DePER would not suffer from the *coupling* issues introduced by summation operation. Then, a deviation of $\mathbf{S}$ to optimal setting $\mathbf{S}^*$ would only result in linear deviation in sampled minibatch B.

A. Theoretical Analysis

We provide a theoretical analysis of DePER and WS-PER on their underlying principles to illustrate why `DePER` could perform *decoupled* sampling while `WSPER` cannot.

Proposition 1 (Sampling distribution of DePER): For the n-th transition in the replay buffer, the probability of it being sampled under DePER is the budget-weighted sum of it being sampled by each individual PER:

$$\begin{aligned} P_{\text{DePER}}(n) &= \sum_i P(n,i) = \sum_i P(n|i)P(i) \\ &= \sum_i P(n|i)\frac{S_i}{B} = \sum_i \frac{S_i}{B}\frac{\exp[\tau p_n^i]}{\sum_n \exp[\tau p_n^i]} \\ &= \sum_i \frac{S_i}{B} P_{\text{PER}}(n,i) \end{aligned}$$

The key lies in the observation that the process of DePER sampling could be reformulated as sampling the latent i (PER scheme) first, then sampling the corresponding conditional distribution.

Proposition 2 (Sampling Distribution of WS-PER): For the n-th transition in the replay buffer, the probability of it being sampled under WS-PER is given by:

$$P_{\text{WSPER}}(n) = \frac{\exp\left[\tau \sum_i w_i p_n^i\right]}{\sum_n \exp\left[\tau \sum_i w_i p_n^i\right]} = \frac{\prod_i \exp\left[\tau w_i p_n^i\right]}{\sum_n \prod_i \exp\left[\tau w_i p_n^i\right]}$$

It cannot be transformed into the form of $\sum_i \alpha_i P_{\text{PER}}(n,i)$ (α_i are non-negative scalars and sum to 1). This illustrates the sampling process of WS-PER is not factorizable and coupled.

VI. Experiments

We firstly introduce our environment design in this section. The basic environments used in this work are 4 classical continuous control environments in DeepMind Control Suite [27]: `Hopper`, `Walker`, `Cheetah` and a 3-dimensional, more challenging `Humanoid`. Notably, to thoroughly test PER methods, we specially modify these environments to make them more challenging: In the first phase, i.e. the first half of online RL training ($0 \sim 0.5$M frames), the task is to train the agent move *backward* solely. Then, in the second phase, i.e. last half of online training ($0.5\text{M} \sim 1\text{M}$ frames), the agent needs to move backward in the first 1/3 episode, then it needs to move *forward* in the remaining 2/3 of episode, representing a sharp change in reward function the agent needs to adapt. In this phase, only the transitions corresponding to *moving forward* are appended into replay buffer to further challenge the given PER method on its ability to tackle *catastrophic forgetting*.

An illustrative diagram about our environment design is provided in Figure 3. The methods compared are:

1) `Uniform`. This is the most basic sampling method.
2) `Near`. This sampling method puts more importance on transitions that are newly appended, as a simple trick to boost sample efficiency as used in [13].
3) `WS-PER`. This is the standard WS-PER method that is widely adopted in existing literature for multi-scheme PER. It utilizes consistency loss, value loss and reward

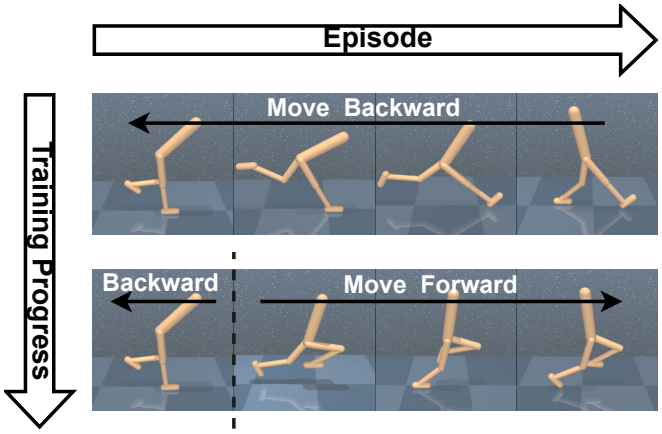

Fig. 3: Illustrative diagram of our environment design.

loss during training. We tune its $\mathbf{w}$ with 13 variants for a fair comparison.

4) `DePER`. This is the method developed above. To showcase its broad applicability, we integrate totally *9* prioritization schemes while keeping the batch size budget $\mathbf{S}$ as intuitive as possible.

In this section, we aim to answer the following 5 questions:

1) How is `DePER`'s performance compared to baselines?
2) How is `DePER`'s performance compared to *heavily tuned* `WS-PER`?
3) How does `DePER` perform with perturbed S?
4) What is the influence of certain prioritization scheme in `DePER`?
5) How is the computation efficiency of `DePER`?

All experiments are run in *10 seeds* and we plot all episodic returns in the course of full online training (roughly 667 points) to better depict the learning curves.

A. *Q1: How is `DePER`'s performance compared to baselines?*

In this section, we perform experiments evaluating various PER methods in environments defined above. The results is shown in Figure 4 (`WSPER-median` is the one obtaining *median* performance across tuned `WSPER` variants while `WSPER-best` is the variant obtaining *best* result, see Q2 for all tuned `WSPER`s' results) and the following observations could be drawn:

1) In all environments except `Humanoid`, applying `near` alone could obtain decent sample efficiency in the first phase but would cause performance degradation in the second phase, indicating applying simple trick to emphasize recent transitions is not sufficient for our challenging tasks. Also, compared to other methods, the performance of `near` is less stable and more fluctuating, especially in `hopper`.
2) In the first phase, most other algorithms perform similarly well. This is because when the replay buffer is relatively small, various sampling strategies don't show significant differences.
3) In the second phase, `DePER` could bring notable sample efficiency gains compared to other baselines: it surpasses `WSPER-median` on all environments and `WSPER-best` on 3 out of 4 environments (on second phase of `Hopper`, `DePER` shows a fast adaptation ability but gets surpassed by `WSPER-best` on converged performance). Also, on `Humanoid`, the converged performance of it is slightly higher than other baselines.
4) On relatively simpler environments like `Walker` and `Cheetah`, `WSPER-median` and `WSPER-best` perform similarly and mostly on par with `Uniform`, indicating tuning the $\mathbf{w}$ on these environments may be insignificant. While on `Hopper`, `WSPER-best` performs notably better than `WSPER-median` and `Uniform`,

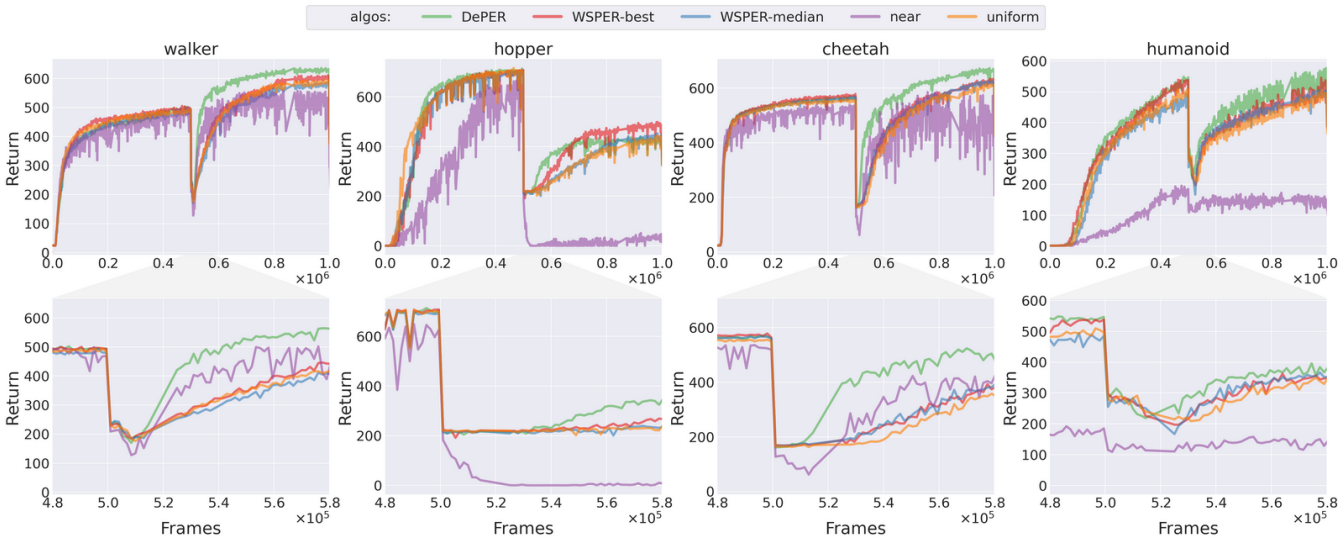

Fig. 4: The main results of various PER methods. The first row plots episodic returns during the whole online learning process, the second row is the *zoomed version* of first row on $48\% \sim 58\%$.

mainly on the second phase. For Humanoid, the WSPER-best obtains sample efficiency gains comparable to DePER in the first phase but fails to maintain it in the second phase.

B. Q2: How is DePER's performance compared to tuned WS-PER?

To enable a fair comparison between WS-PER and DePER, we performed a **13-budget** tuning over WS-PER's $\mathbf{w}$ on each environment (the $\mathbf{S}$ in our DePER is *not tuned* and the same as in Q1). In this section, we analyze the results of these tuned WSPERs and how DePER's performance is compared to them. We also employ **Rliable** (an advanced statistical tool to test the significance of RL method's improvement as in [28]) to better compare the DePER's performance against WSPER.

In Figure 5 and Figure 6, we compare the performance of WS-PER-best, WS-PER-Q3, WS-PER-median, WS-PER-Q1 and WS-PER-worst which are the best, 75% quantile, median, 25% quantile and worst results among the 13 tuned WS-PERs, respectively. In Figure 6 the 2 metrics used are: 1) *Performance* which is episodic return averaging across the whole learning process, and 2) *Adaptation ability* which is the number of episodes needed for performance recovery after the environmental change on 0.5M.

From the results, we could find: 1) On simple environments like cheetah, tuning $\mathbf{w}$ shows no significant performance difference as almost all WSPER shows similar performance (tuning $\mathbf{S}$ is also insignificant, as shown in Q3); 2) On other environments, tuning $\mathbf{w}$ could bring notable performance difference in both first and second phase. But the magnitude is not significant and even WSPER-best's performance and adaptation ability lag clearly behind DePER. 3) From *Rliable*'s result, it's clearly our DePER outperforms WSPER variants by a large margin in both performance and adaptation. WSPER-max could achieve a relatively comparable performance against DePER but with a larger variation. For adaptation ability, all WSPER variants are significantly worse than DePER, which shows DePER could benefit from integrating multiple priori-

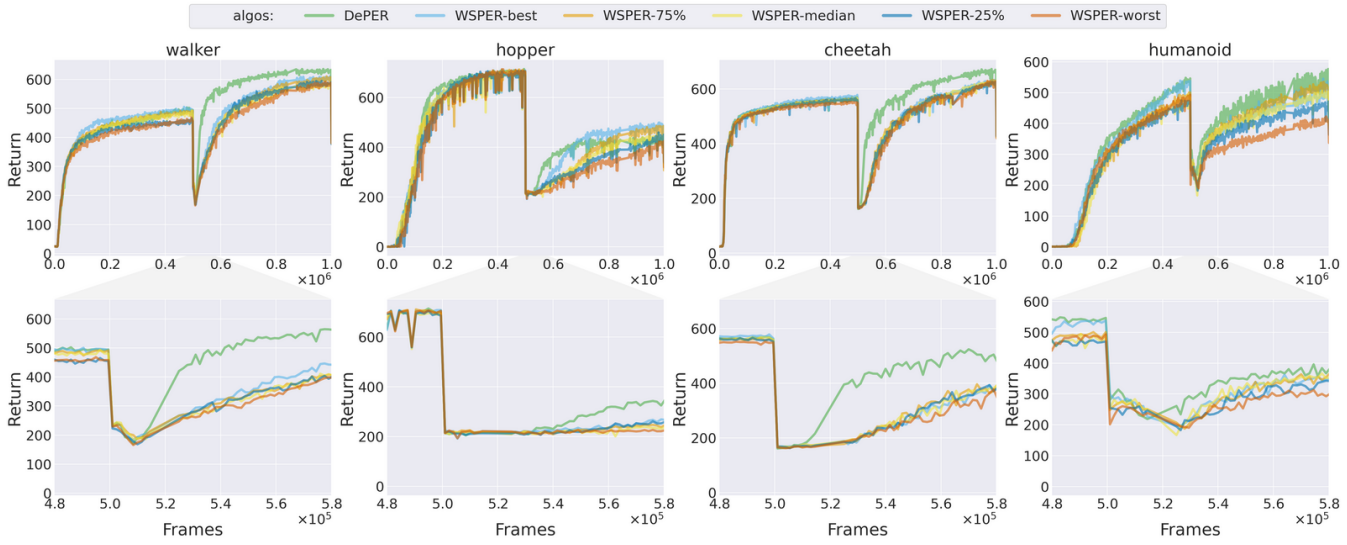

Fig. 5: The summary plot of training curves of DePER and representative WSPER variants. The second rows plot the *zoomed* version of first row on 48% $\sim$ 58%.

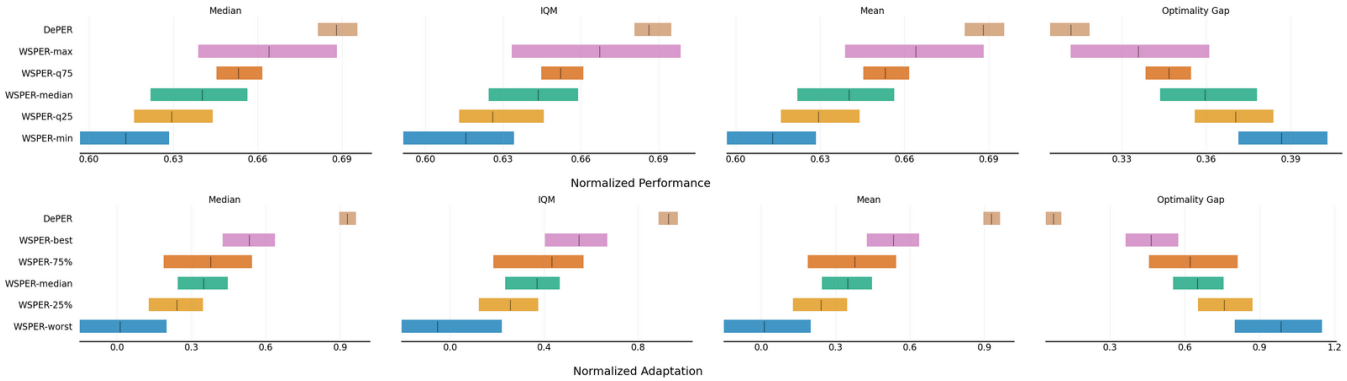

Fig. 6: Plots comparing DePER to WS-PER variants on *Performance* (first row, average episodic return) and *Adaptation* (second row, episodes needed for performance recovery after environmental change) using *Rliable*.

tization schemes.

C. Q3: How does DePER perform with perturbed $\mathcal{S}$?

In this section we aim to test how certain level of perturbation in $\mathcal{S}$ influences the performance of DePER. In details, for the original DePER where $\mathcal{S}$ = [32, 32, 32, 32, 32, 32, 16, 16, 32] (stands for csc_loss, rwd_loss, val_loss, pol_loss, value, uniform, near, old and reward, respectively and $\sum S = 256$), we add roughly 25% level of perturbation to it and obtained 4 perturbed versions: DePER-pt1, pt2, pt3 and pt4 [3]. The result is shown in Figure 7 where WSPER-75% and WSPER-25% are also included for comparison.

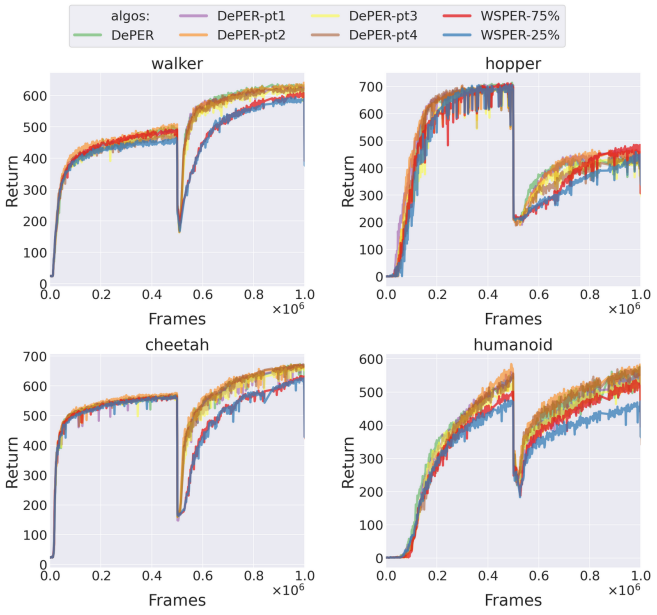

Fig. 7: The performance of DePER with perturbed $\mathcal{S}$.

From the result, we could find: 1) On walker and cheetah, the perturbation in $\mathcal{S}$ doesn't influence the DePER's performance significantly; while 2) On hopper and humanoid, the perturbation would cause certain level of performance variation but the variation magnitude of DePER is still smaller than WSPER's; 3) Also, the performance of all DePER's variants are still better than WSPER's, indicating $\mathbf{S}$ has better robustness than $\mathbf{w}$.

D. Q4: What is the influence of certain prioritization scheme in DePER?

In this section, we aim to investigate the influence of certain prioritization scheme in DePER. We conduct *minus-one* experiment where 1 prioritization scheme is removed (its budget is allocated to *uniform* scheme, i.e., total budget remains fixed) and test how this would influence the performance of DePER. The results are in Figure 8 where -near, -old, -value and -val_loss represents the removal of *near*, *old*, *value* and *values loss*. There are 2 interesting observations: 1) On

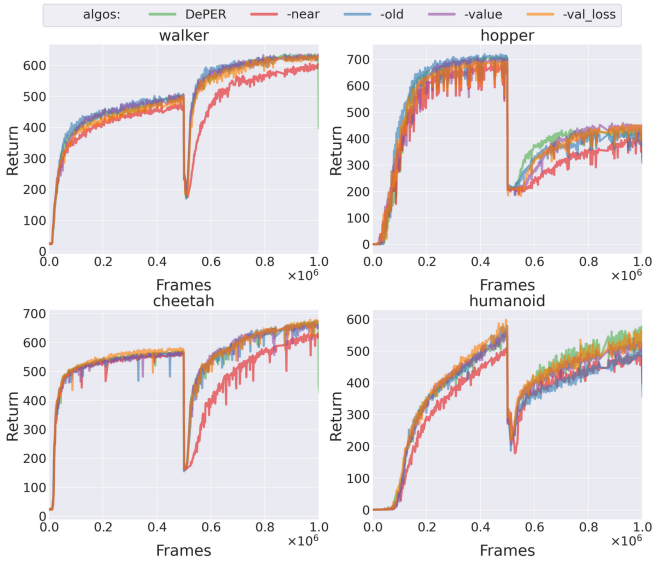

Fig. 8: The performances with certain scheme removed.

relatively simple environments like walker and cheetah, removing near would cause significant performance degradation than removal of other schemes; 2) On hopper, each prioritization scheme has certain level of contributions and removal of any one would all cause performance decrease; 3) On the second phase of humanoid environment, removal of old would have similar adverse effects as to the removal of near because only newer samples are appended into replay buffer hence the old is important for catering forgetting.

E. Q5: How is the computation efficiency of DePER?

In this section, we show how the computation time (wall clock time) of DePER changes with the number of schemes in Figure 9. In this experiment, we remove prioritization scheme one by one from the original DePER and record the runtime of each variant. The results are shown in Figure 9.

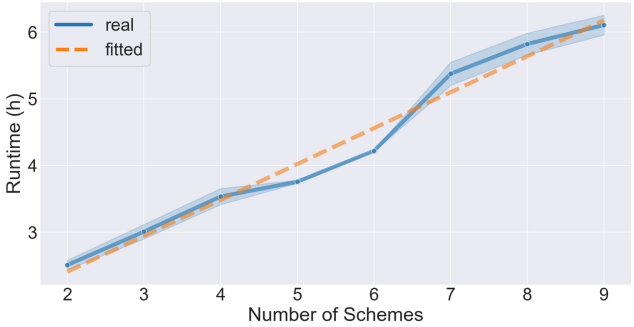

Fig. 9: Computation efficiency of DePER.

From the result, DePER's computation time increases approximately *linearly* with the number of schemes. The fitted line is $y = 0.54x + 1.33$ (hour) based on linear regression.

[3] pt1 to pt4: [24, 40, 24, 40, 24, 20, 24, 20, 40], [24, 40, 40, 40, 24, 12, 40, 12, 24], [40, 24, 24, 24, 40, 20, 24, 20, 40] and [40, 24, 40, 24, 40, 12, 40, 12, 24]. The meaning of each element is [csc_loss, rwd_loss, val_loss, pol_loss, value, uniform, near, old, reward].

VII. Conclusion

In this paper we study multi-scheme PER: Considering WS-PER's practical and theoretical difficulty which hinders its wide adoption in multi-scheme PER, we propose `DePER` and validate its advantages with extensive experiments. In the future work, we would explore employing multi-thread techniques to further reduce `DePER`'s runtime overhead.

VIII. Acknowledgements

This work was supported by the Brain Science and Brain-like Intelligence Technology — National Science and Technology Major Project (Grant No. 2022ZD0208800)

References

[1] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "Rma: Rapid motor adaptation for legged robots," in *Robotics: Science and Systems*, 2021.

[2] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science robotics*, vol. 5, no. 47, p. eabc5986, 2020.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[4] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, "First return, then explore," *Nature*, vol. 590, no. 7847, pp. 580–586, Feb. 2021.

[5] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *Advances in neural information processing systems*, vol. 35, pp. 27 730–27 744, 2022.

[6] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn, "Direct Preference Optimization: Your Language Model is Secretly a Reward Model," Jul. 2024.

[7] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," *arXiv preprint arXiv:1810.12894*, 2018.

[8] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," https://arxiv.org/abs/1511.05952v4, Nov. 2015.

[9] N. Hansen, H. Su, and X. Wang, "TD-MPC2: Scalable, Robust World Models for Continuous Control," Mar. 2024.

[10] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," Feb. 2016.

[11] I. Kauvar, C. Doyle, L. Zhou, and N. Haber, "Curious Replay for Model-based Adaptation," Jun. 2023.

[12] Y. Oh, J. Shin, E. Yang, and S. J. Hwang, "Model-augmented Prioritized Experience Replay," in *International Conference on Learning Representations*, Oct. 2021.

[13] C. Wang, Y. Wu, Q. Vuong, and K. Ross, "Striving for Simplicity and Performance in Off-Policy DRL: Output Normalization and Non-Uniform Sampling," Dec. 2020.

[14] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The Arcade Learning Environment: An Evaluation Platform for General Agents," Jun. 2013.

[15] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv:1509.02971 [cs, stat]*, Jul. 2019.

[16] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust Region Policy Optimization," Apr. 2017.

[17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv:1707.06347 [cs]*, Aug. 2017.

[18] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft Actor-Critic Algorithms and Applications," Jan. 2019.

[19] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *SIGART Bull.*, vol. 2, no. 4, pp. 160–163, Jul. 1991.

[20] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to Control: Learning Behaviors by Latent Imagination," Mar. 2020.

[21] N. Hansen, X. Wang, and H. Su, "Temporal Difference Learning for Model Predictive Control," Jul. 2022.

[22] M. Janner, Q. Li, and S. Levine, "Offline Reinforcement Learning as One Big Sequence Modeling Problem," Nov. 2021.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is All you Need," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017.

[24] S. Fujimoto, D. Meger, and D. Precup, "An Equivalence between Loss Functions and Non-Uniform Sampling in Experience Replay," Oct. 2020.

[25] T. Lahire, M. Geist, and E. Rachelson, "Large Batch Experience Replay," Jun. 2022.

[26] S. Sujit, S. Nath, P. H. M. Braga, and S. E. Kahou, "Prioritizing Samples in Reinforcement Learning with Reducible Loss," Nov. 2023.

[27] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, T. Lillicrap, and M. Riedmiller, "DeepMind Control Suite," Jan. 2018.

[28] R. Agarwal, M. Schwarzer, P. S. Castro, A. Courville, and M. G. Bellemare, "Deep Reinforcement Learning at the Edge of the Statistical Precipice," Jan. 2022.