SCALEAC: SCALE ACTOR-CRITIC BY REPLAY RATIO

Anonymous authors
Paper under double-blind review

ABSTRACT

Employing a high replay ratio, the number of agent updates per environment interaction, has emerged as a promising strategy to improve sample efficiency in reinforcement learning (RL). However, most existing efforts to scale the replay ratio stagnate at small values, and what happens inside the agent's network under a high update frequency remains largely unexamined. In this paper, we bridge this gap by scaling the replay ratio from the perspective of network internals to achieve sample-efficient RL. We identify a critical pathology whereby the dormant neuron phenomenon intensifies with increasing replay ratios, which undermines network capacity and destroys learning. To address this problem, we propose ScaleAC, a synergy of three elements based on Actor-Critic (AC) algorithms, for high-replay-ratio settings with theoretical insights. First, to tackle dormant neurons under a high update frequency, ScaleAC introduces a periodic soft network reset in RL agents. Second, to further stabilize high-replay-ratio training, ScaleAC integrates an in-target random minimization into the Q target computation. Third, to prevent overfitting at high replay ratios, ScaleAC diversifies the replay experience through data augmentations. Extensive experiments in MuJoCo and DMC demonstrate that ScaleAC scales replay ratios in both vector-based RL (up to 256) and pixel-based RL, and extends to a modern RL network architecture (i.e., SimbaV2), yielding substantial learning acceleration and performance improvement.

1 INTRODUCTION

When applying reinforcement learning (RL) to the physical world, a critical limitation of existing RL approaches is their poor sample efficiency, requiring a large number of environment interactions to learn satisfactory policies. As exploring such environments entails high costs and safety risks, it is always desirable to learn with a minimal number of environment interactions in practice. Therefore, sample-efficient RL is essential for real-world scenarios (Yu, 2018).

One natural idea is to scale the replay ratio, the number of updates to an agent's network per environment interaction (Chen et al., 2021). Recent studies found that increasing the replay ratio brings substantial performance improvement in RL algorithms (D'Oro et al., 2023; Smith et al., 2023). For example, Chen et al. (2021) introduce an in-target random minimization technique called REDQ into soft actor-critic (SAC) (Haarnoja et al., 2018) to support a replay ratio of up to 20 for continuous control. Subsequently, DroQ (Hiraoka et al., 2022) regularizes the critic with Dropout (Srivastava et al., 2014) and Layer Normalization (Xu et al., 2019) to achieve similar benefits to REDQ under the same replay ratio with lower computational overhead. Nikishin et al. (2022) further found that deep RL agents incur a risk of overfitting to earlier experiences, and simply resetting part of the agent periodically allows SAC to scale to a high replay ratio of 32. More recently, D'Oro et al. (2023) push the replay ratio of SAC to 128 by proposing Scaled-by-Resetting SAC (SR-SAC), which fully resets the network parameters of SAC at a fixed update interval.

Despite the above works successfully training RL at high replay ratios, they rarely examine what happens inside an agent's network in this setting, thereby limiting the possibility of scaling RL to a higher replay ratio. In this paper, we scale the replay ratio from the perspective of network internals to achieve sample-efficient RL. We identify the dormant neuron phenomenon as a critical bottleneck that intensifies with increasing replay ratios, where a larger portion of neurons become inactive to undermine the network capacity. To address this problem, we propose ScaleAC, which integrates three key components built upon Actor-Critic (AC) algorithms to scale the replay ratio of RL for continuous control. First, we utilize a periodic soft network reset technique (Ash & Adams, 2020)

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

to tackle the severe dormant network neuron problem at high replay ratios. Second, we introduce an in-target random minimization into the Q target computation to further stabilize high-frequency training. Third, we employ two forms of data augmentation techniques in the high-replay-ratio setting to diversify input states and prevent overfitting. Extensive experiments in MuJoCo (Todorov et al., 2012) and DMC (Tunyasuvunakool et al., 2020) environments demonstrate that, compared to various baselines, ScaleAC significantly accelerates learning and improves performance at high replay ratios. The main contributions of this work are summarized below:

- We propose ScaleAC, which integrates periodic soft resets, in-target random minimization, and data augmentation into AC algorithms (e.g., SAC and DDPG) to scale the replay ratio.
- We identify the dormant neuron phenomenon as a critical bottleneck that intensifies when scaling replay ratios, and soft network reset mitigates it with theoretical insights.
- We scale the replay ratio in both vector-based RL (up to 256) and pixel-based RL, and extend ScaleAC to modern RL network architectures such as SimbaV2 (Lee et al., 2025b).

2 Background

2.1 Reinforcement Learning

Consider a Markov Decision Process (MDP). At each time step t, an agent observes a state s_t and selects an action a_t, and then the environment provides the next reward r_t and state s_{t+1}. For convenience, we use the simpler notations of r, s, a, s', and a' to refer to a reward, state, action, next state, and next action, respectively. The objective of an RL agent is to optimize its policy π where $a \sim \pi(s)$ to maximize the expected discounted cumulative reward $J(\pi) = \mathbb{E}_\pi[\sum_{t=0}^{T} \gamma^t r_t]$, where $\gamma \in [0, 1)$ is a discount factor and T is the horizon. The state-action value function $Q^\pi(s, a) = \mathbb{E}_\pi[\sum_{t=0}^{T} \gamma^t r_t | s_0 = s, a_0 = a]$ gives the expected return starting in s, taking an action a, then following policy π. In deep RL, policy and value functions are approximated by neural networks.

2.2 Soft Actor-Critic and Deep Deterministic Policy Gradient

The vector-based ScaleAC builds on SAC, in which an agent's policy maximizes the expected return plus an entropy bonus, a measure of policy randomness that encourages exploration.

$$\pi^* = \arg\max_\pi \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{s_t, a_t}[r_t + \beta \mathcal{H}(\pi(\cdot|s_t))], \tag{1}$$

where $\pi : S \times A \to \mathbb{R}$ is a policy. $\mathcal{H}$ is the entropy where $\mathcal{H}(P) = \mathbb{E}_{x \sim P}[-\log P(x)]$ for a distribution P. The temperature β determines the relative importance of the entropy term against the reward, and thus controls the stochasticity of the policy. The target Q in SAC is computed as

$$y = r + \gamma(Q^\pi(s', \tilde{a}') - \beta \log \pi(\tilde{a}'|s')), \tilde{a}' \sim \pi(\cdot|s'). \tag{2}$$

and the policy is updated with gradient ascent as

$$\nabla_\theta(Q_\phi(s, \tilde{a}_\theta(s)) - \beta \log \pi_\theta(\tilde{a}_\theta(s)|s)), \tilde{a}_\theta(s) \sim \pi_\theta(\cdot|s). \tag{3}$$

The pixel-based ScaleAC builds on Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2016), following DrQ-v2 (Yarats et al., 2022). DDPG concurrently learns a Q-function Q_ϕ and a deterministic policy μ_θ where $a = \mu_\theta(s)$. The Q target is computed as

$$y = r + \gamma(Q_{\bar{\phi}_i}(s', \mu_\theta(s'))), \tag{4}$$

and the policy is updated with gradient ascent as

$$\nabla_\theta Q_\phi(s, \mu_\theta(s)) = \nabla_{\tilde{a}} Q_\phi(s, \tilde{a})|_{\tilde{a}=\mu_\theta(s)} \nabla_\theta \mu_\theta(s). \tag{5}$$

2.3 Dormant Neuron

Sokar et al. (2023) identify the dormant neuron phenomenon in deep RL, where an agent's network suffers from an increasing number of inactive neurons during the training process, thereby affecting network expressivity. This phenomenon is closely related to the broader study of plasticity loss in deep RL (Lyle et al., 2025), where neural networks gradually lose their ability to adapt to new information during training. The definition of the dormant neuron is given below.

Definition 2.1 (τ-Dormant Neuron)**.** Given an input distribution D, let $\rho_j^l(x)$ denote the activation of neuron j in layer l under input $x \in D$ and N_l be the number of neurons in layer l. The normalized activation score of a neuron j in layer l is defined as follows:

$$d_j^l = \frac{\mathbb{E}_{x \in D} |\rho_j^l(x)|}{\frac{1}{N_l} \sum_{k=1}^{N_l} \mathbb{E}_{x \in D} |\rho_k^l(x)|}. \tag{6}$$

Then neuron j in layer l is defined as τ-dormant if its activation score $d_j^l \leq \tau$. In this paper, we set τ at 0.01. The activation function is ReLU, which is a default choice in most RL network designs.

2.4 Scaling Replay Ratio in Deep RL

Moderately increasing the replay ratio for model-free RL algorithms is a competitive data-efficient baseline for both discrete and continuous control when compared to model-based RL methods (Nikishin et al., 2022; D'Oro et al., 2023). Methods like REDQ (Chen et al., 2021), primacy bias mitigation (Nikishin et al., 2022), and SR-SAC (D'Oro et al., 2023) progressively push the replay ratio from 20 to 128. Despite successfully training RL at high replay ratios, these works rarely examine what happens inside the agent's network, thereby limiting the possibility of scaling to a higher replay ratio for sample efficiency. To bridge this gap, we focus on further scaling RL replay ratios from a new perspective that connects training replay ratios to dormant network neurons.

A parallel stream of work attempts to improve sample efficiency of RL agents through more advanced network architectures. For example, BRO (Nauman et al., 2024) proposes the BroNet architecture to scale the critic to about 5 million parameters, using various tricks such as layer normalization and residual connections. Similarly, SimBa (Lee et al., 2025a) scales up network parameters of SAC through network architecture modifications, including an observation normalization layer, a residual feedforward block, and layer normalization. Its successor SimbaV2 (Lee et al., 2025b) replaces that layer normalization by projecting every weight matrix onto the unit hypersphere, which bounds the growth of weight and feature norms during training. These efforts are orthogonal to ours. Rather than modifying and enlarging the network, we scale the replay ratio under the default architecture and model size, from a new perspective of connecting dormant neurons to replay ratios. The two directions also compose, as Section 4.7 shows that ScaleAC applies to SimbaV2.

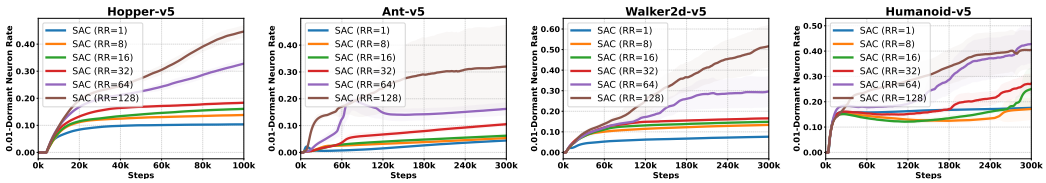
3 Method

In this section, we propose ScaleAC for both vector-based and pixel-based RL at high replay ratios. First, we reveal that high replay ratios cause a severe dormant neuron problem in AC and employ soft network reset to handle it in Section 3.1. Second, in Section 3.2, we introduce an in-target random minimization technique to ScaleAC for training stability at high replay ratios. Third, in Section 3.3, we utilize different data augmentation techniques to diversify state inputs to prevent overfitting.

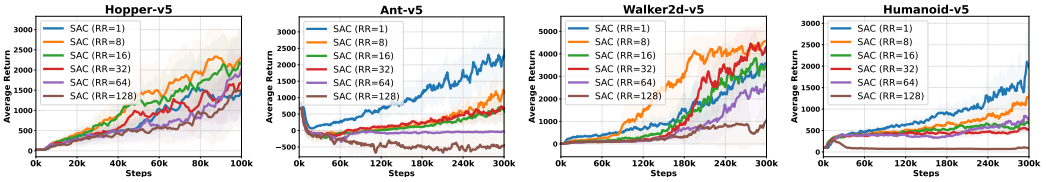
3.1 Shrink & Perturb to Tackle Dormant Neurons

Here, we show how the replay ratio affects dormant neurons of RL agents. Specifically, we measure the dormant neuron rate, defined as the proportion of τ-dormant neurons of a neural network, in the SAC agent. The dormant neuron ratios and test episode returns are shown in Figure 1. Across all tasks tested, we clearly observe that increasing the replay ratio results in high dormant neuron rates. When the replay ratio reaches 64 or 128, the performance of SAC deteriorates significantly. This experimental study reveals that high replay ratios lead to elevated dormant neuron rates in agents, which undermines the network's representation ability as a larger portion of network neurons become inactive, thereby crippling the learning process. Such a correlation between replay ratios and dormant neurons, supported by the theoretical analysis in Theorem 3.1, motivates us to reduce dormant neurons in the agent's network to stabilize RL training at high replay ratios for desired sample efficiency. We formally characterize how the replay mechanism influences neurons as follows.

Theorem 3.1 (Replay Increases Neuron Dormancy Probability)**.** *Under an arbitrary but fixed input x, consider a neuron with ReLU activation and initial pre-activation $z^{(0)} = w^\top x > 0$, where w is the weight vector. Let Y_i be the random variable defined by $Y_i = (\nabla_w \ell_i)^\top x$, where $\nabla_w \ell_i$ is*



(a) Dormant neuron on Hopper-v5. (b) Dormant neuron on Ant-v5. (c) Dormant neuron on Walker2d-v5. (d) Dormant neuron on Humanoid-v5.



(e) Reward on Hopper-v5. (f) Reward on Ant-v5. (g) Reward on Walker2d-v5. (h) Reward on Humanoid-v5.

Figure 1: The dormant neuron rates and episode returns of SAC at different replay ratios (RR).

the gradient of the loss at the i-th stochastic update. Using stochastic gradient updates, let the pre-activation of a neuron evolve after k gradient updates as a discrete stochastic process

$$z^{(k)} = z^{(0)} - \eta \sum_{i=1}^{k} Y_i, \tag{7}$$

where $\eta > 0$ is the learning rate, and $\{Y_i\}_{i\geq 1}$ are i.i.d. random variables with finite non-zero variance. Define the dormancy event within N_{RR} update steps as $\mathcal{D}_{N_{RR}} = \{\exists\, k \in \{1, \ldots, N_{RR}\} : z^{(k)} \leq 0\}$. Then the dormancy probability satisfies

$$\forall N_{RR,1} > N_{RR,2},\ \mathbb{P}(\mathcal{D}_{N_{RR,1}}) \geq \mathbb{P}(\mathcal{D}_{N_{RR,2}}). \tag{8}$$

Therefore, increasing the replay ratio N_{RR} increases the probability of a dormant neuron within a finite optimization horizon. Meanwhile, the fixed-input assumption becomes more plausible when N_{RR} is large, particularly in the early training stage when the replay buffer contains few diverse samples, causing the same or similar inputs to be replayed repeatedly. The proof is in Appendix A.

Next, we tackle the severe dormant neuron problem. Nikishin et al. (2022) choose to fully reset network parameters of a part of an SAC agent, such as its last few layers, to initial values. A similar strategy is followed by D'Oro et al. (2023), who completely reset all the network parameters of an agent every 2.56×10^6 updates. However, such a hard network reset behavior discards the learned weights in the reset network layers and heavily relies on the replay buffer to restore the agent's previously learned experience. In contrast, ScaleAC adopts a soft reset strategy to retain learned parameters partially. Specifically, we introduce the Shrink & Perturb strategy (Ash & Adams, 2020) to periodically and partially reset the agent's network parameters to initial values, thereby injecting network plasticity while keeping learned experience. This Shrink & Perturb strategy was originally proposed to warm-start neural network training to incorporate newly arriving data without sacrificing generalization (Ash & Adams, 2020). More recently, Shrink & Perturb has been successfully applied in the SPR (Schwarzer et al., 2021) family of algorithms built upon Deep Q Networks (Mnih et al., 2015), such as SR-SPR (D'Oro et al., 2023) and BBF (Schwarzer et al., 2023), to prevent overfitting in a high-replay-ratio setting for discrete-action-space control. In this work, we instead integrate Shrink & Perturb into AC algorithms to address dormant neurons for continuous-action-space control. The formulation of Shrink & Perturb is defined as

$$\theta_t \leftarrow \alpha\theta_t + (1-\alpha)\theta_0, \text{ and } \phi_t \leftarrow \alpha\phi_t + (1-\alpha)\phi_0, \tag{9}$$

where θ_0 are the initial policy parameters and ϕ_0 are the initial critic parameters. θ_t and ϕ_t are the current policy and critic parameters, respectively. The interpolation factor α decides how much the current network parameters are kept. As below, Shrink & Perturb can reactivate dormant neurons.

Theorem 3.2 (Shrink & Perturb Reduces Neuron Dormancy)**.** *Under an arbitrary but fixed input x, consider a dormant neuron with ReLU activation and pre-activation $z_t = w_t^\top x \leq 0$. Let the Shrink*

& Perturb update be

$$w_t \leftarrow \alpha w_t + (1-\alpha) w_0, \tag{10}$$

where $\alpha \in (0,1)$ *and* w_0 *is the initial weight vector. Then the updated pre-activation satisfies*

$$z'_t = \alpha z_t + (1-\alpha) z_0, \tag{11}$$

where $z_0 = w_0^\top x$. *Let* $z_0 \sim \mathcal{N}(\mu, \sigma^2)$ *(which holds under standard Gaussian initialization of* w_0*, e.g., Xavier or He initialization). Then*

$$\mathbb{P}(z'_t > 0 \mid z_t \leq 0) > 0, \tag{12}$$

where the probability of activating the dormant neuron is larger than 0. The proof is in Appendix A.

In contrast, under standard stochastic gradient updates without Shrink & Perturb, a dormant ReLU neuron has zero gradient for the same input, giving $\mathbb{P}(z'_t > 0 \mid z_t \leq 0) = 0$ and remaining dormant.

3.2 Random Minimization to Stabilize Training

Standard SAC makes use of double Q-learning (Hasselt, 2010; Fujimoto et al., 2018) to reduce overestimation for training stability, where the minimum of two Q-functions is used for the Q target value estimation. To further enhance training stability, we introduce the in-target random minimization technique (Chen et al., 2021) into ScaleAC to stabilize high-replay-ratio training. In-target random minimization extends the idea of double Q-learning by maintaining an ensemble of N Q-functions and taking an in-target random minimum over a subset $\mathbb{M}$ of M Q-functions from the ensemble. In addition to controlling the average overestimation bias of Q-functions, in-target random minimization also reduces the standard deviation of the Q-function bias to nearly zero for most of the learning process, thus supporting stable SAC training (Chen et al., 2021). The Q target in vector-based ScaleAC, built upon SAC, is computed as

$$y = r + \gamma(\min_{i \in \mathbb{M}} Q_i(s', \tilde{a}') - \beta \log \pi(\tilde{a}'|s')), \tilde{a}' \sim \pi(\cdot|s'), \tag{13}$$

where i is the index of N Q-functions, $\mathbb{M}$ is the random subset, and β is the temperature in SAC. The policy π_θ is updated with gradient ascent as $\nabla_\theta \frac{1}{N} \sum_{i=1}^{N} (Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \beta \log \pi_\theta(\tilde{a}_\theta(s)|s))$, $\tilde{a}_\theta(s) \sim \pi_\theta(\cdot|s)$, where each Q-function is parameterized by ϕ and the policy is parameterized by θ.

Similarly, for pixel-based RL (Yarats et al., 2022) with visual image input, we introduce the in-target random minimization technique to ScaleAC built upon DDPG (Lillicrap et al., 2016), where the Q target is computed as $y = r + \gamma(\min_{i \in \mathbb{M}} Q_{\bar{\phi}_i}(s', \mu_\theta(s')))$, where μ_θ is a parameterized actor function which deterministically maps states to a specific action, and the agent policy is updated through the chain rule by maximizing Q. $\mathbb{M}$ is a subset of M Q-functions, which is randomly sampled from the ensemble of size N. The policy π_θ in visual ScaleAC is updated with gradient ascent as $\frac{1}{N} \sum_{i=1}^{N} \nabla_{\tilde{a}} Q_{\phi_i}(s, \tilde{a})|_{\tilde{a}=\mu_\theta(s)} \nabla_\theta \mu_\theta(s)$.

3.3 Data Augmentation to Prevent Overfitting

As ScaleAC updates RL agents with a high replay ratio, where the updating is increased by many times, it is natural to augment each mini-batch of sampled transitions to prevent overfitting (Nikishin et al., 2022). Therefore, in vector-based RL, we introduce the random amplitude scaling (Laskin et al., 2020) into ScaleAC to diversify the replay experiences and prevent overfitting under high replay ratios (Yang et al., 2024). Random amplitude scaling is a classical data augmentation technique, particularly suited for vector-based RL with proprioceptive inputs (e.g., positions and velocities) (Laskin et al., 2020; He et al., 2023). It randomizes the amplitude of input states while preserving intrinsic consistency. Its formulation is defined as

$$s \leftarrow s * z, \text{ and } s' \leftarrow s' * z, \tag{14}$$

where $z \sim U(z_a, z_b)$ is randomly sampled from a uniform distribution over $[z_a, z_b]$. Note that the random amplitude scaling is applied randomly to each batch of experiences but consistently across time, i.e., the same random factor for the current and next input state vectors. Random amplitude scaling preserves the intrinsic consistency, such as the sign of inputs along adjacent time steps.

In pixel-based RL, to diversify the image input of the visual version of ScaleAC, we employ another data augmentation technique called random shift with bilinear interpolation (Yarats et al., 2022).

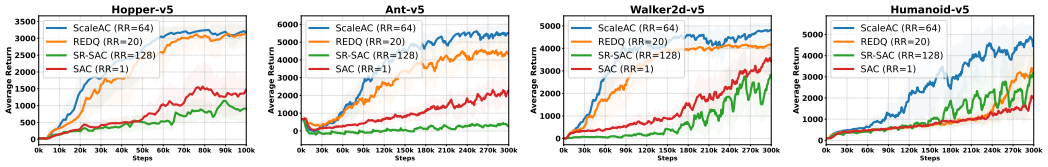
This technique applies random shifts to pixel observations for image augmentation. For the visual image input, the random shift is instantiated by first padding each side of the 84×84 observation by 4 pixels (by repeating boundary pixels), and then selecting a random 84×84 crop, yielding the original image shifted by ± 4 pixels. Subsequently, bilinear interpolation is applied on top of the shifted image by replacing each pixel value with the average of its four nearest pixel values.

The complete algorithm of vector-based ScaleAC is detailed in Algorithm 1 in Appendix B, and the pixel-based visual ScaleAC is described in Appendix C. Next, we conduct extensive experiments to validate ScaleAC in high-replay-ratio settings on a variety of continuous control tasks.

4 Experiments

In this section, we evaluate ScaleAC in both MuJoCo (Todorov et al., 2012) and DeepMind Control Suite (DMC) (Tunyasuvunakool et al., 2020) of continuous action control. First, in Section 4.1, we benchmark ScaleAC in MuJoCo with different SAC baselines of their default replay ratios. Second, in Section 4.2, we compare ScaleAC with SAC baselines, as well as several network reset methods, at the same replay ratio. Third, we further benchmark ScaleAC on a total of 10 tasks in DMC, including 7 challenging DMC-Hard tasks, in Section 4.3. Fourth, in Section 4.4, we show that ScaleAC scales a replay ratio to even 256. Fifth, we give the ablation study of ScaleAC in Section 4.5. Finally, we validate pixel-based ScaleAC in Section 4.6 and extend ScaleAC to SimbaV2 in Section 4.7. Detailed hyperparameters of both vector-based and pixel-based ScaleAC are provided in Appendix E. The wall-clock cost of ScaleAC and the baselines is reported in Appendix J.

4.1 Benchmark Experiments in MuJoCo



(a) Reward on Hopper-v5. (b) Reward on Ant-v5. (c) Reward on Walker2d-v5. (d) Reward on Humanoid-v5.

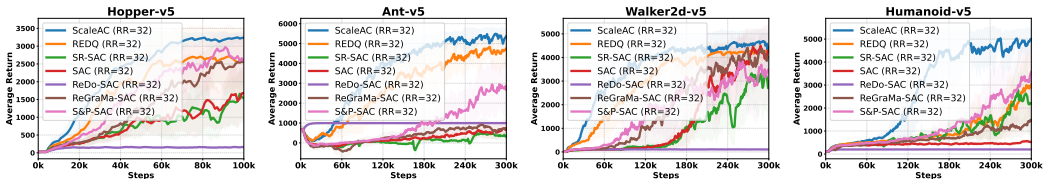
Figure 2: Benchmark RL algorithms with their default replay ratios in the MuJoCo environment.

First, we benchmark ScaleAC on four classical MuJoCo tasks, including Hopper-v5, Ant-v5, Walker2d-v5, and Humanoid-v5. Detailed descriptions of each task are provided in Appendix D. For baselines, SAC uses a replay ratio of 1, REDQ uses 20, and SR-SAC uses 128. For ScaleAC, we set the replay ratio at 64. The benchmark results are shown in Figure 2. The reported metrics are averaged over 6 independent trials, and the 95% confidence intervals are shown as shaded regions.

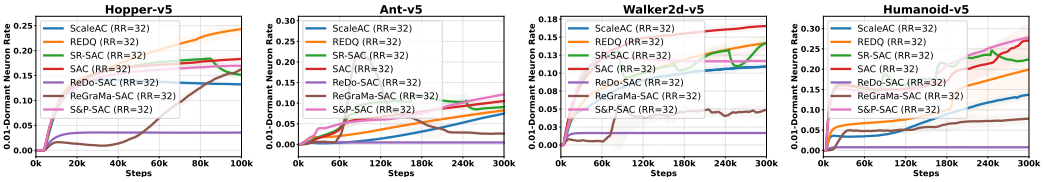
As shown, ScaleAC with a replay ratio of 64 consistently outperforms the baselines given the same interaction budget, demonstrating its superior sample efficiency. Meanwhile, SR-SAC with a replay ratio of 128 underperforms, suggesting that hard resetting of RL agents is a suboptimal choice here.

4.2 Benchmarking with the Same Replay Ratio

Next, we benchmark methods and examine their dormant neurons at the same replay ratio of 32. We also compare other network reset techniques, such as Shrink & Perturb (S&P), ReDo (Sokar et al., 2023), and ReGraMa (Liu et al., 2025). ReDo resets each neuron if it is dormant. ReGraMa resets a neuron if its normalized gradient value is below a threshold. Results are plotted in Figure 3. Under the same replay ratio, ScaleAC achieves the best performance on all four tasks. On the one hand, ScaleAC has a lower dormant neuron rate than SAC baselines (REDQ, SR-SAC, and SAC), showing that reducing severe dormant neurons enables progressive learning in the high-replay-ratio setting. On the other hand, although ReDo-SAC achieves a dormant neuron rate near 0, it fails to learn on all tasks, indicating that forcing a much lower dormant neuron rate may overly discard learned parameters and harm representation ability. In summary, ScaleAC effectively balances both dormant neuron reduction for network capacity and parameter preservation for learned knowledge.



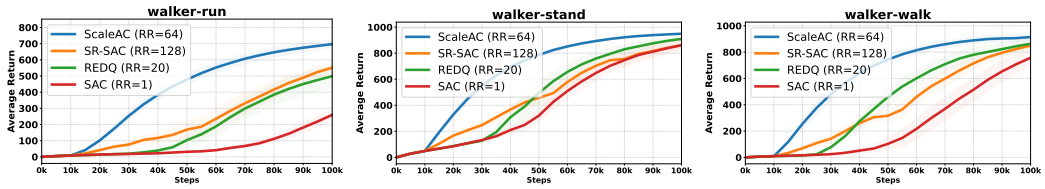
(a) Reward on Hopper-v5. (b) Reward on Ant-v5. (c) Reward on Walker2d-v5. (d) Reward on Humanoid-v5.



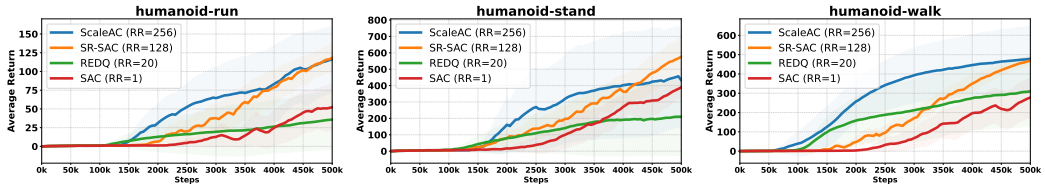
(e) Dormant neuron on Hopper-v5. (f) Dormant neuron on Ant-v5. (g) Dormant neuron on Walker2d-v5. (h) Dormant neuron on Humanoid-v5.

Figure 3: Experimental results of RL methods in MuJoCo with the same replay ratio of 32.

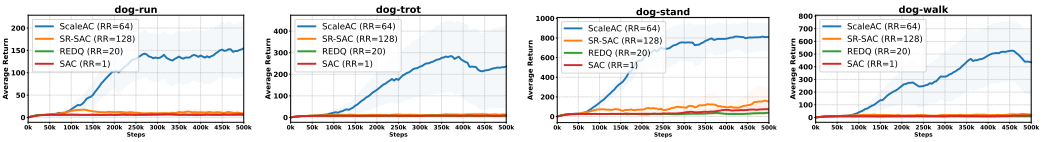
4.3 Benchmark Experiments in DMC



(a) Reward on walker-run. (b) Reward on walker-stand. (c) Reward on walker-walk.



(d) Reward on humanoid-run. (e) Reward on humanoid-stand. (f) Reward on humanoid-walk.



(g) Reward on dog-run. (h) Reward on dog-trot. (i) Reward on dog-stand. (j) Reward on dog-walk.

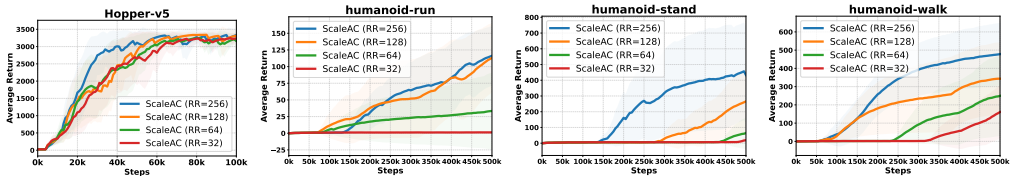
Figure 4: Benchmark RL algorithms with their default replay ratios in the DMC environment.

In this section, we validate ScaleAC in the DMC environment, which features a range of locomotion and manipulation tasks. We benchmark methods in 3 walker-series tasks and 7 challenging DMC-Hard tasks (3 humanoid-series tasks as well as 4 dog-series tasks). More details about these tasks are given in Appendix D. SAC's replay ratio is 1, REDQ's replay ratio is 20, and SR-SAC's replay ratio is 128. For the walker-series and dog-series tasks, we set the replay ratio of ScaleAC at 64. For the humanoid-series tasks, we find that a replay ratio of 256 works best for ScaleAC. Results are shown in Figure 4. The reported test episode returns are averaged over 6 independent trials with different random seeds, and the 95% confidence intervals are shaded.

ScaleAC outperforms baselines in almost all these DMC tasks, with one exception that ScaleAC performs slightly worse than SR-SAC on humanoid-stand in the final environment steps. On the other hand, REDQ struggles to learn on the DMC Hard tasks, i.e., 3 humanoid-series tasks and

4 dog-series tasks, highlighting the necessity of ScaleAC's additional components beyond REDQ, such as Shrink & Perturb and random amplitude scaling. Meanwhile, although SR-SAC achieves good performance on humanoid-series tasks, it fails on dog-series tasks, showing that fully resetting networks may not be a general solution for training RL at high replay ratios. Moreover, on dog-series tasks, ScaleAC successfully learns from scratch while other methods fail, indicating the broad generality of ScaleAC. In conclusion, these experimental results of ScaleAC in both MuJoCo and DMC demonstrate its effectiveness in stabilizing RL training under high replay ratios for sample efficiency while showcasing its generality across different kinds of tasks.

4.4 Scaling up the Replay Ratio to 256



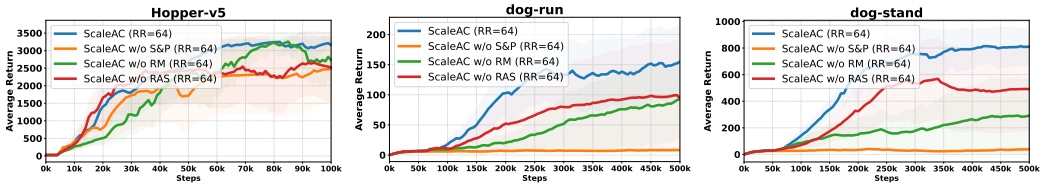
(a) Reward on Hopper-v5. (b) Reward on humanoid-run. (c) Reward on humanoid-stand. (d) Reward on humanoid-walk.

Figure 5: Scaling up the replay ratio of ScaleAC to 256.

In this section, we demonstrate that ScaleAC is able to scale up the replay ratio to even 256. The results of ScaleAC with different replay ratios of 32, 64, 128, and 256 are plotted in Figure 5. As we see, ScaleAC significantly accelerates the learning of RL agents with a replay ratio of up to hundreds, especially on humanoid-stand and humanoid-walk. To the best of our knowledge, this is the highest reported replay ratio to successfully train deep RL agents for continuous control in the current literature (Ma et al., 2025). In summary, ScaleAC unlocks the potential to achieve better sample efficiency through scaling of the replay ratio to hundreds (i.e., 256).

On the other hand, we also notice that a higher replay ratio does not always mean higher performance. ScaleAC may have different optimal replay ratios on different tasks. For example, as shown in the dog-series tasks of Appendix G, the best replay ratio of ScaleAC is 64, while replay ratios such as 128 and 256 decrease the performance. Nevertheless, ScaleAC reveals that further scaling the replay ratio is a possible way to unlock further sample efficiency of RL agents.

4.5 Ablation Study of ScaleAC

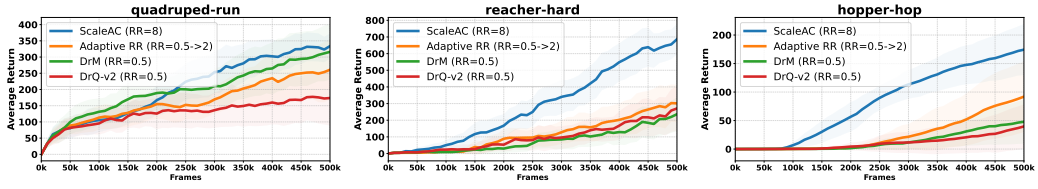


(a) Reward on Hopper-v5. (b) Reward on dog-run. (c) Reward on dog-stand.

Figure 6: The ablation study of ScaleAC. 'w/o S&P' removes Shrink & Perturb. 'w/o RM' removes in-target random minimization. 'w/o RAS' removes random amplitude scaling.

We conduct an ablation study to verify each component of ScaleAC, including Shrink & Perturb (denoted as S&P), in-target random minimization (denoted as RM), and random amplitude scaling (denoted as RAS). The ablation results are given in Figure 6. We see that each component contributes to ScaleAC. Especially, without Shrink & Perturb, ScaleAC fails to learn on the tasks of dog-run and dog-stand, which demonstrates its role in reducing dormant neurons when the replay ratio is high. Additionally, in Appendix H, we replace Shrink & Perturb in ScaleAC with other network reset techniques, such as ReDo, Reset (D'Oro et al., 2023), and ReGraMa, to show that the soft reset mechanism of Shrink & Perturb works best in the high-replay-ratio setting.

4.6 Benchmark Experiments of Visual ScaleAC

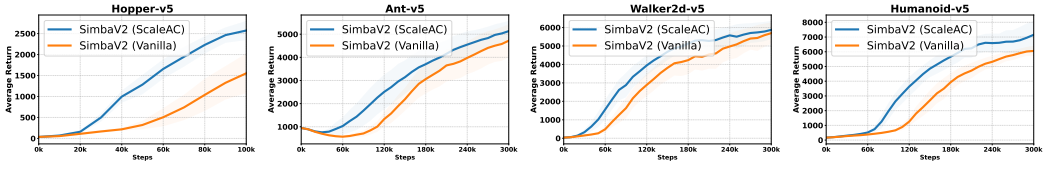


(a) Reward on quadruped-run. (b) Reward on reacher-hard. (c) Reward on hopper-hop.

Figure 7: Benchmarking results on visual DMC tasks.

In this section, we benchmark pixel-based ScaleAC on three tasks in visual DMC, including quadruped-run, reacher-hard, and hopper-hop. The details of each task are provided in Appendix D.3. We compare pixel-based ScaleAC with three advanced visual RL approaches. DrQ-v2 (Yarats et al., 2022) has a default replay ratio of 0.5. DrM (Xu et al., 2024) uses the dormant ratio to guide exploration and network perturbation at the default replay ratio of 0.5. Adaptive RR (Ma et al., 2024) uses an adaptive replay ratio increased from 0.5 to 2 when the fraction of active units in the critic hits a predefined threshold. Results are shown in Figure 7. ScaleAC achieves learning acceleration and performance improvement over baselines with a higher replay ratio of 8, demonstrating the potential of enabling high replay ratios in visual RL.

4.7 Extending ScaleAC to the Modern RL Network Architecture



(a) Reward on Hopper-v5. (b) Reward on Ant-v5. (c) Reward on Walker2d-v5. (d) Reward on Humanoid-v5.

Figure 8: Applying ScaleAC to the SimbaV2 architecture on MuJoCo tasks.

The experiments above build ScaleAC on networks of standard multi-layer perceptrons. Recent work instead improves sample efficiency of RL through the network architecture itself, such as SimbaV2 (Lee et al., 2025b). Whether ScaleAC still contributes to a more advanced RL network architecture is therefore a natural question, and here we examine it.

We apply ScaleAC to SimbaV2 and scale the replay ratio to 4 on four MuJoCo tasks and ten DMC tasks. All runs use 6 seeds. Vanilla SimbaV2 is at its default replay ratio of 1. MuJoCo results are reported in Figure 8 and DMC results are reported in Appendix I. ScaleAC transfers to SimbaV2 and consistently learns faster than the vanilla version on all tasks. This confirms the benefits of extending ScaleAC to modern RL network architectures.

5 Conclusion

In this paper, we propose ScaleAC to scale up the replay ratio for sample-efficient RL. First, we identify the dormant neuron phenomenon as a critical bottleneck that intensifies with increasing replay ratios to undermine network learning capacity. To address it, ScaleAC synergistically combines periodic soft reset, in-target random minimization, and data augmentation to enable high-replay-ratio training. Extensive experiments show that ScaleAC greatly improves sample efficiency of both vector- and pixel-based RL and extends to a modern RL network architecture at high replay ratios.

For future work, first, dynamically adjusting the replay ratio is promising to reduce the computation cost. Second, extending ScaleAC to LLM post-training also has great potential. Third, designing plastic network structures for ScaleAC is a natural direction to further scale the replay ratio.

REFERENCES

Jordan T. Ash and Ryan P. Adams. On warm-starting neural network training. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, 2020.

Xinyue Chen, Che Wang, Zijian Zhou, and Keith W. Ross. Randomized Ensembled Double Q-Learning: Learning Fast Without a Model. In *International Conference on Learning Representations*, 2021.

Pierluca D'Oro, Max Schwarzer, Evgenii Nikishin, Pierre-Luc Bacon, Marc G. Bellemare, and Aaron Courville. Sample-Efficient Reinforcement Learning by Breaking the Replay Ratio Barrier. In *The Eleventh International Conference on Learning Representations*, 2023.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pp. 1587–1596, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pp. 1861–1870, 2018.

Hado van Hasselt. Double Q-learning. In *Proceedings of the 24th International Conference on Neural Information Processing Systems - Volume 2*, pp. 2613–2621, 2010.

Haoran He, Chenjia Bai, Kang Xu, Zhuoran Yang, Weinan Zhang, Dong Wang, Bin Zhao, and Xuelong Li. Diffusion Model is an Effective Planner and Data Synthesizer for Multi-Task Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 64896–64917, 2023.

Takuya Hiraoka, Takahisa Imagawa, Taisei Hashimoto, Takashi Onishi, and Yoshimasa Tsuruoka. Dropout Q-Functions for Doubly Efficient Reinforcement Learning. In *International Conference on Learning Representations*, 2022.

Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, 2020.

Hojoon Lee, Dongyoon Hwang, Donghu Kim, Hyunseung Kim, Jun Jet Tai, Kaushik Subramanian, Peter R. Wurman, Jaegul Choo, Peter Stone, and Takuma Seno. SimBa: Simplicity Bias for Scaling Up Parameters in Deep Reinforcement Learning. In *The Thirteenth International Conference on Learning Representations*, 2025a.

Hojoon Lee, Youngdo Lee, Takuma Seno, Donghu Kim, Peter Stone, and Jaegul Choo. Hyperspherical Normalization for Scalable Deep Reinforcement Learning. In *Forty-second International Conference on Machine Learning*, 2025b.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *Proceedings of the 4th International Conference on Learning Representations*, 2016.

Jiashun Liu, Zihao Wu, Johan Obando-Ceron, Pablo Samuel Castro, Aaron Courville, and Ling Pan. Measure gradients, not activations! Enhancing neuronal activity in deep reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Clare Lyle, Zeyu Zheng, Khimya Khetarpal, Hado van Hasselt, Razvan Pascanu, James Martens, and Will Dabney. Disentangling the Causes of Plasticity Loss in Neural Networks. In *Proceedings of The 3rd Conference on Lifelong Learning Agents*, volume 274, pp. 750–783, 2025.

Guozheng Ma, Lu Li, Sen Zhang, Zixuan Liu, Zhen Wang, Yixin Chen, Li Shen, Xueqian Wang, and Dacheng Tao. Revisiting Plasticity in Visual Reinforcement Learning: Data, Modules and Training Stages. In *The Twelfth International Conference on Learning Representations*, 2024.

Yi Ma, Hongyao Tang, Chenjun Xiao, Yaodong Yang, Wei Wei, Jianye Hao, and Jiye Liang. Scaling DRL for Decision Making: A Survey on Data, Network, and Training Budget Strategies, 2025. arXiv:2508.03194 [cs].

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Michal Nauman, Mateusz Ostaszewski, Krzysztof Jankowski, Piotr Miłoś, and Marek Cygan. Bigger, regularized, optimistic: scaling for compute and sample-efficient continuous control. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, 2024.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The Primacy Bias in Deep Reinforcement Learning. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pp. 16828–16847, 2022.

Max Schwarzer, Ankesh Anand, Rishab Goel, R. Devon Hjelm, Aaron Courville, and Philip Bachman. Data-Efficient Reinforcement Learning with Self-Predictive Representations. In *International Conference on Learning Representations*, 2021.

Max Schwarzer, Johan Obando-Ceron, Aaron Courville, Marc G. Bellemare, Rishabh Agarwal, and Pablo Samuel Castro. Bigger, better, faster: human-level atari with human-level efficiency. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Laura Smith, Ilya Kostrikov, and Sergey Levine. Demonstrating a walk in the park: Learning to walk in 20 minutes with model-free reinforcement learning. *Robotics: Science and Systems (RSS) Demo*, 2(3):4, 2023.

Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The Dormant Neuron Phenomenon in Deep Reinforcement Learning. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1): 1929–1958, 2014.

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, 2012.

Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020.

Guowei Xu, Ruijie Zheng, Yongyuan Liang, Xiyao Wang, Zhecheng Yuan, Tianying Ji, Yu Luo, Xiaoyu Liu, Jiaxin Yuan, Pu Hua, Shuzhen Li, Yanjie Ze, Hal Daumé III, Furong Huang, and Huazhe Xu. DrM: Mastering Visual Reinforcement Learning through Dormant Ratio Minimization. In *International Conference on Learning Representations*, volume 2024, pp. 56219–56243, 2024.

Jingjing Xu, Xu Sun, Zhiyuan Zhang, Guangxiang Zhao, and Junyang Lin. Understanding and improving layer normalization. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., 2019.

Yaodong Yang, Guangyong Chen, Jianye Hao, and Pheng Ann Heng. Sample-efficient multiagent reinforcement learning with reset replay. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering Visual Continuous Control: Improved Data-Augmented Reinforcement Learning. In *International Conference on Learning Representations*, 2022.

Yang Yu. Towards sample efficient reinforcement learning. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pp. 5739–5743, 2018.

A Proof of Neuron Dormancy

Theorem A.1 (Replay Increases Neuron Dormancy Probability)**.** *Under an arbitrary but fixed input x, consider a neuron with ReLU activation and initial pre-activation $z^{(0)} = w^\top x > 0$, where w is the weight vector. Let Y_i be the random variable defined by $Y_i = (\nabla_w \ell_i)^\top x$, where $\nabla_w \ell_i$ is the gradient of the loss at the i-th stochastic update. Using stochastic gradient updates, let the pre-activation of a neuron evolve after k gradient updates as a discrete stochastic process*

$$z^{(k)} = z^{(0)} - \eta \sum_{i=1}^{k} Y_i, \tag{15}$$

where $\eta > 0$ is the learning rate, and $\{Y_i\}_{i\geq 1}$ are i.i.d. random variables with finite non-zero variance. Define the dormancy event within N_{RR} update steps as $\mathcal{D}_{N_{RR}} = \big\{\exists\, k \in \{1, \ldots, N_{RR}\} : z^{(k)} \leq 0\big\}$. Then the dormancy probability satisfies

$$\forall N_{RR,1} > N_{RR,2},\ \mathbb{P}(\mathcal{D}_{N_{RR,1}}) \geq \mathbb{P}(\mathcal{D}_{N_{RR,2}}). \tag{16}$$

Therefore, increasing the replay ratio N_{RR} increases the probability of a dormant neuron within a finite optimization horizon.

Proof. Under ReLU activation, the region $z^{(k)} \leq 0$ is absorbing in the sense that once a neuron enters this region, its activation is effectively inactive for subsequent updates. This is because the gradient under the fixed input x is zero, and the weight of this neuron will not be updated. Therefore, dormancy corresponds to the event that the stochastic process enters the negative region once during training.

By definition,

$$\mathcal{D}_{N_{RR}} = \bigcup_{k=1}^{N_{RR}} \{z^{(k)} \leq 0\}. \tag{17}$$

Since a larger N_{RR} allows additional time steps for the same stochastic trajectory to enter the dormant region, for any $N_{RR,1} > N_{RR,2}$, we have

$$\mathcal{D}_{N_{RR,2}} \subseteq \mathcal{D}_{N_{RR,1}}. \tag{18}$$

Then by monotonicity of probability measures, we have

$$\mathbb{P}(\mathcal{D}_{N_{RR,2}}) \leq \mathbb{P}(\mathcal{D}_{N_{RR,1}}). \tag{19}$$

Therefore, increasing the replay ratio N_{RR} enlarges the time horizon over which dormancy can be observed, and hence monotonically increases the probability of neuron dormancy. □

In practical training settings, dormant neurons are commonly observed, suggesting that there exists k such that a neuron's pre-activation $z^{(k)} \leq 0$. Under this condition, increasing N_{RR} reduces the number of environment steps for the neuron to enter dormancy. Moreover, the fixed-input assumption becomes more plausible when N_{RR} is large, particularly in the early training stage when the replay buffer contains few diverse samples, causing the same or similar inputs to be replayed repeatedly.

Theorem A.2 (Shrink & Perturb Reduces Neuron Dormancy)**.** *Under an arbitrary but fixed input x, consider a dormant neuron with ReLU activation and pre-activation $z_t = w_t^\top x \leq 0$. Let the Shrink & Perturb update be*

$$w_t \leftarrow \alpha w_t + (1-\alpha) w_0, \tag{20}$$

where $\alpha \in (0, 1)$ and w_0 is the initial weight vector. Then the updated pre-activation satisfies

$$z_t' = \alpha z_t + (1-\alpha) z_0, \tag{21}$$

where $z_0 = w_0^\top x$. Let $z_0 \sim \mathcal{N}(\mu, \sigma^2)$ (which holds under standard Gaussian initialization of w_0, e.g., Xavier or He initialization). Then

$$\mathbb{P}(z_t' > 0 \mid z_t \leq 0) > 0, \tag{22}$$

where the probability of activating the dormant neuron is larger than 0.

Proof. From the update rule,

$$z'_t = (w'_t)^\top x = (\alpha w_t + (1-\alpha)w_0)^\top x = \alpha z_t + (1-\alpha)z_0. \tag{23}$$

Assume the neuron is dormant at step t, i.e., $z_t \leq 0$. Then the condition $z'_t > 0$ is equivalent to

$$z_0 > -\frac{\alpha}{1-\alpha} z_t. \tag{24}$$

Thus,

$$\mathbb{P}(z'_t > 0 \mid z_t \leq 0) = \mathbb{P}\Big(z_0 > -\frac{\alpha}{1-\alpha} z_t\Big). \tag{25}$$

Since z_0 is a random variable with Gaussian initialization, we have

$$\mathbb{P}(z'_t > 0 \mid z_t \leq 0) = \mathbb{P}\Big(z_0 > -\frac{\alpha}{1-\alpha} z_t\Big) = 1 - \Phi\left(\frac{-\frac{\alpha}{1-\alpha} z_t - \mu}{\sigma}\right) = \Phi\left(\frac{\alpha z_t + (1-\alpha)\mu}{(1-\alpha)\sigma}\right), \tag{26}$$

where $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution and $\Phi(-a) = 1 - \Phi(a)$ by symmetry of the standard normal distribution. Since $\Phi(z) > 0$ for all $z \in \mathbb{R}$, we have

$$\mathbb{P}(z'_t > 0 \mid z_t \leq 0) > 0. \tag{27}$$

Therefore, a dormant neuron has a strictly positive probability of being reactivated by Shrink & Perturb. □

If we update the weights with standard SGD and do not apply Shrink & Perturb. Under standard SGD, if $z_t \leq 0$, then for the same input x, the ReLU neuron produces zero gradient:

$$\frac{\partial L}{\partial w_t} = 0, \tag{28}$$

and thus

$$z_{t+1} = z_t. \tag{29}$$

Hence,

$$\mathbb{P}(z_{t+1} > 0 \mid z_t \leq 0) = 0. \tag{30}$$

The Shrink & Perturb update introduces a non-zero transition probability from the dormant region ($z \leq 0$) to the active region ($z > 0$), while standard SGD has zero probability. Therefore, the dormant state is no longer absorbing.

B VECTOR-BASED SCALEAC

In Algorithm 1, in Line 7, the random amplitude scaling is performed on the sampled mini-batch transitions. Lines 8-9 utilize the in-target random minimization to calculate the target Q-value based on the augmented transitions. In Lines 18-19, the Shrink & Perturb operation is conducted every T_R environment steps on agent networks, which suffer from the severe dormant neuron problem when updated at high replay ratios.

C PIXEL-BASED SCALEAC

In this section, we give the detailed algorithm of ScaleAC based on DDPG for pixel-based visual RL, which is shown in Algorithm 2. In Line 7, the random shift with bilinear interpolation is performed on the sampled mini-batch transitions. Lines 8-9 use the in-target random minimization to calculate the target Q-value. In Lines 18-19, Shrink & Perturb is conducted every T_R environment steps on agent networks, which suffer from the severe dormant neuron problem when training at a high replay ratio. The critic is updated N_{RR} times every environment interaction. As indicated in Line 14, the actor is updated every T_θ critic updates or after the N_{RR} critic updates. The visual version of ScaleAC has an encoder to encode an image into a vector for both the actor and critic networks. We also reset the encoder in the same way as resetting the critic.

Algorithm 1 ScaleAC for Scaling Vector-Based Actor-Critic by Replay Ratio

Initialize policy network parameters θ, the critic network parameters $\phi_i, i = 1, 2, \cdots, N$, and an empty replay buffer D. Set target critic network parameters $\bar{\phi}_i \leftarrow \phi_i, i = 1, 2, \cdots, N$. Set agent network reset interval T_R. Set policy network update interval T_θ.
for each time step t **do**
  Agent takes action $a_t \sim \pi_\theta(\cdot|s_t)$. Observe next state s_{t+1}. Receive reward r_t.
  Add transition data to the replay buffer: $D \leftarrow D \cup \{(s_t, a_t, r_t, s_{t+1})\}$.
  for each update time n_{RR} from 1 to N_{RR} **do**
    Sample a mini-batch $B = \{(s, a, r, s')\}$ from D.
    Apply random amplitude scaling as in Equation (14) on sampled transition batch B.
    Sample a set $\mathbb{M}$ of M distinct indices from $\{1, 2, \cdots, N\}$.
    Compute the target Q-value y (same for all critics) as $y = r + \gamma(\min_{i\in\mathbb{M}} Q_{\bar{\phi}_i}(s', \tilde{a}') - \beta \log \pi_\theta(\tilde{a}'|s')), \tilde{a}' \sim \pi_\theta(\cdot|s')$.
    for $i = 1, 2, \cdots, N$ **do**
      Update ϕ_i with gradient descent as $\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s')\in B} (Q_{\phi_i}(s, a) - y)^2$.
      Update target networks as $\bar{\phi}_i \leftarrow \rho\bar{\phi}_i + (1 - \rho)\phi_i$.
    end for
    if $(n_{RR} \bmod T_\theta = 0)$ or $(n_{RR} = N_{RR})$ **then**
      Update policy θ with gradient ascent as $\nabla_\theta \frac{1}{|B|} \sum_{s\in B} (\frac{1}{N} \sum_{i=1}^{N} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \beta \log \pi_\theta(\tilde{a}_\theta(s)|s)), \tilde{a}_\theta(s) \sim \pi_\theta(\cdot|s)$.
    end if
  end for
  if $t \bmod T_R = 0$ **then**
    Perform Shrink & Perturb as in Equation (9) on agent networks.
  end if
end for

Algorithm 2 ScaleAC for Scaling Pixel-Based Actor-Critic by Replay Ratio

Initialize policy network parameters θ, the critic network parameters $\phi_i, i = 1, 2, \cdots, N$, and an empty replay buffer D. Set target critic network parameters $\bar{\phi}_i \leftarrow \phi_i, i = 1, 2, \cdots, N$. Set agent network reset interval T_R. Set policy network update interval T_θ.
for each time step t **do**
  Agent takes action $a_t \sim \pi_\theta(\cdot|s_t)$. Observe next state s_{t+1}. Receive reward r_t.
  Add transition data to the replay buffer: $D \leftarrow D \cup \{(s_t, a_t, r_t, s_{t+1})\}$.
  for each update time n_{RR} from 1 to N_{RR} **do**
    Sample a mini-batch $B = \{(s, a, r, s')\}$ from D.
    Apply random shift with bilinear interpolation on sampled transition batch B.
    Sample a set $\mathbb{M}$ of M distinct indices from $\{1, 2, \cdots, N\}$.
    Compute the target Q-value y (same for all critics) as $y = r + \gamma(\min_{i\in\mathbb{M}} Q_{\bar{\phi}_i}(s', \mu_\theta(s')))$.
    for $i = 1, 2, \cdots, N$ **do**
      Update ϕ_i with gradient descent using $\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s')\in B} (Q_{\phi_i}(s, a) - y)^2$.
      Update target networks with $\bar{\phi}_i \leftarrow \rho\bar{\phi}_i + (1 - \rho)\phi_i$.
    end for
    if $(n_{RR} \bmod T_\theta = 0)$ or $(n_{RR} = N_{RR})$ **then**
      Update policy network parameters θ with gradient ascent using $\frac{1}{|B|} \sum_{s\in B} (\frac{1}{N} \sum_{i=1}^{N} \nabla_{\tilde{a}} Q_{\phi_i}(s, \tilde{a})|_{\tilde{a}=\mu_\theta(s)} \nabla_\theta \mu_\theta(s))$.
    end if
  end for
  if $t \bmod T_R = 0$ **then**
    Perform Shrink & Perturb as in Equation (9) on agent networks.
  end if
end for

D Environment Details

D.1 MuJoCo

We consider a total of 4 continuous control tasks in the MuJoCo environment (Todorov et al., 2012). These tasks include Hopper-v5, Ant-v5, Walker2d-v5, and Humanoid-v5. The short description, observation dimension, and action space dimension of each task are listed in Table 1. For ScaleAC and baselines in the MuJoCo benchmark experiments, we utilize the official codebase of REDQ (Chen et al., 2021) with PyTorch to implement algorithms.

Table 1: Description of Each MuJoCo Task.

Task	Robot	Description	State dim	Action dim
Hopper-v5	2D Runners	2D monoped for hopping	11	3
Ant-v5	Quadruped	3D quadruped for running	105	8
Walker2d-v5	2D Runners	2D biped for walking	17	6
Humanoid-v5	Humanoid Bipeds	3D humanoid for running	348	17

D.2 DMC

We consider a total of 10 continuous control tasks in the DMC environment (Tunyasuvunakool et al., 2020). These tasks include three walker-series tasks, including walker-run, walker-stand, and walker-walk, and seven DMC-Hard tasks, including humanoid-run, humanoid-stand, humanoid-walk, dog-run, dog-trot, dog-stand, and dog-walk. The short description, observation dimension and action space dimension of each task are listed in Table 2. For ScaleAC and baselines in DMC, we utilize the official codebase of SR-SAC (D'Oro et al., 2023) with JAX to implement these algorithms.

Table 2: Description of Each DMC Task.

Task	Difficulty	Description	State dim	Action dim
Walker-run	Medium	An improved planar walker to run	24	6
Walker-stand	Easy	An improved planar walker to stand	24	6
Walker-walk	Easy	An improved planar walker to walk	24	6
Dog-run	Hard	A Pharaoh Dog model to run	223	38
Dog-trot	Hard	A Pharaoh Dog model to trot	223	38
Dog-stand	Hard	A Pharaoh Dog model to stand	223	38
Dog-walk	Hard	A Pharaoh Dog model to walk	223	38
Humanoid-run	Hard	A 21-joint humanoid to run at 10 m/s	67	21
Humanoid-stand	Hard	A 21-joint humanoid to stand at 0 m/s	67	21
Humanoid-walk	Hard	A 21-joint humanoid to walk at 1 m/s	67	21

D.3 Visual DMC

We consider three tasks in the visual DMC environment, including quadruped-run, reacher-hard, and hopper-hop, to validate the visual ScaleAC. Tasks are summarized in Table 3. In this setting, environment observations are stacks of 3 consecutive RGB images of size 84×84, stacked along the channel dimension to enable inference of dynamic information like velocity and acceleration. For ScaleAC and baselines in visual DMC, we utilize the official codebase of DrQ-v2 (Yarats et al., 2022) and Adaptive RR (Ma et al., 2024) with PyTorch to implement these algorithms.

Table 3: Description of Each Visual DMC Task.

Task	Traits	Difficulty	Action dim
Quadruped-run	run	medium	12
Reacher-hard	reach	medium	2
Hopper-hop	move	medium	4

E HYPERPARAMETERS OF SCALEAC

In this section, we provide the hyperparameters of ScaleAC. Notably, all component-specific hyperparameters are directly adopted from their original papers without additional tuning: α for Shrink & Perturb follows (Ash & Adams, 2020), $N = 10$ and $M = 2$ for random minimization follow (Chen et al., 2021), and $[z_a, z_b]$ for random amplitude scaling follows (Laskin et al., 2020). For baselines, we use the default hyperparameters from their official codebases (Chen et al., 2021; D'Oro et al., 2023; Yarats et al., 2022; Ma et al., 2024). For the Shrink & Perturb strategy, we set $T_R = 2000$ to reset the agent networks every 2000 environment steps and set $\alpha = 0.8$ to keep 80% of the values of current network parameters. For random minimization, we set $N = 10$ for the Q-function ensemble size and $M = 2$ for the random subset size, following the default setting as (Chen et al., 2021). For random amplitude scaling, we set $z_a = 0.8$ and $z_b = 1.2$ for the amplitude range. Specifically, in MuJoCo, we set the policy network update interval T_θ at 20, which is updated less frequently than the critic network. Therefore, we only reset the critic network in MuJoCo. Meanwhile, in DMC, we set the policy network update interval T_θ at 1 and reset both the policy network and critic network. In visual DMC, we set the policy network update interval T_θ at 1 and reset the encoder network, policy network, and critic network at the same time. The setting of the random shift with bilinear interpolation for visual ScaleAC is the same as in DrQ-v2. Other hyperparameters are kept the same as the original configurations provided in the official codebases (Chen et al., 2021; D'Oro et al., 2023; Yarats et al., 2022; Ma et al., 2024). The study of hyperparameter sensitivity of ScaleAC is provided in Appendix F for reference.

For the experiments on SimbaV2 in Section 4.7, we keep every component-specific hyperparameter at the value used above, namely $T_R = 2000$, $N = 10$, $M = 2$, $z_a = 0.8$ and $z_b = 1.2$, and reset both the policy and critic networks. One adaptation is required by the architecture. SimbaV2 constrains each weight matrix to the unit hypersphere and stores the representation scale in separate learnable parameters, so the two are reset independently: the weight matrices are left untouched, since interpolating a unit-norm direction toward its initial value discards the learned feature selector rather than restoring plasticity, and $\alpha = 0.8$ is applied to the scale parameters alone. All remaining hyperparameters, including the network widths, the number of residual blocks, the optimizer and the learning-rate schedule, follow the official SimbaV2 configuration (Lee et al., 2025b).

F HYPERPARAMETER SENSITIVITY

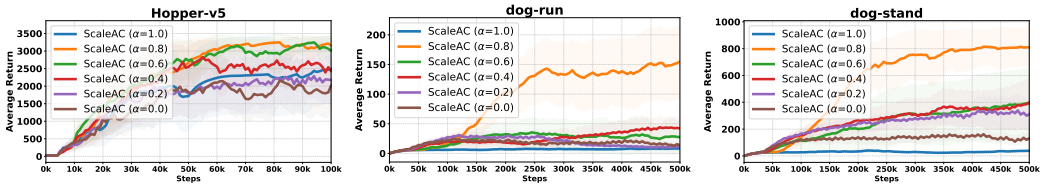
In this section, we study the hyperparameter sensitivity specifically in ScaleAC. First, we show how the interpolation factor α, which determines how much of the current network parameters is mixed with the initial network parameters, affects the performance of ScaleAC in Appendix F.1. Second, in Appendix F.2, we investigate the sensitivity of the reset interval T_R. Finally, we also show how the amplitude range $[z_a, z_b]$ in random amplitude scaling affects ScaleAC in Appendix F.3. The default values of studied hyperparameters of ScaleAC here are $\alpha = 0.8$, $T_R = 2000$, and $[z_a = 0.8, z_b = 1.2]$.

F.1 THE INTERPOLATION FACTOR IN SHRINK & PERTURB

We study the interpolation factor α with different values, and the corresponding results are given in Figure 9. When $\alpha = 0.0$, the network parameters are fully reset to initial values. When $\alpha = 1.0$, the network parameters are totally kept. As shown in Figure 9, $\alpha = 0.8$ achieves the best performance in the three scenarios tested, which is also the default value across tasks and environments. In the dog-run and dog-stand tasks, other values of α do not perform well, indicating that the interpolation factor α is an important hyperparameter to tune in ScaleAC carefully.

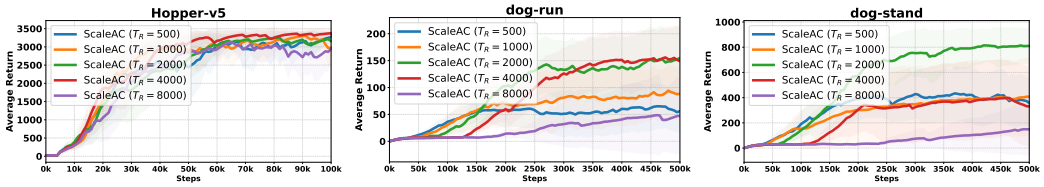
F.2 THE RESET INTERVAL IN SHRINK & PERTURB

The default reset interval T_R is 2000, which means ScaleAC applies Shrink & Perturb every 2000 environment steps. Here we also experiment with $T_R = 500$, 1000, 4000, and 8000. The resulting plots are given in Figure 10. When $T_R = 4000$, ScaleAC achieves the best performance in Hopper-v5 and dog-run, but performs worse than $T_R = 2000$ in the task of dog-stand. Generally, the default reset interval $T_R = 2000$ performs well in these tested cases.



(a) Reward on Hopper-v5. (b) Reward on dog-run. (c) Reward on dog-stand.

Figure 9: The hyperparameter sensitivity study on the interpolation factor α of ScaleAC.

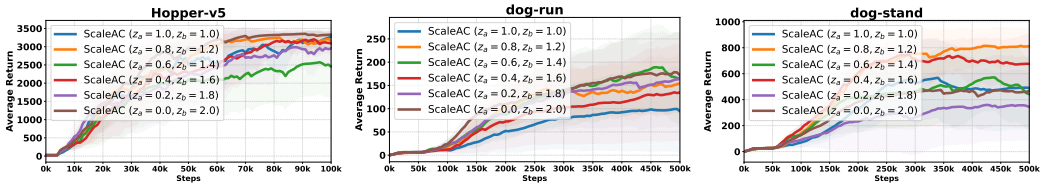


(a) Reward on Hopper-v5. (b) Reward on dog-run. (c) Reward on dog-stand.

Figure 10: The hyperparameter sensitivity study on the reset interval T_R of ScaleAC.

F.3 The Scaling Range in Random Amplitude Scaling

We study the amplitude scaling range $[z_a, z_b]$ with different values, and the corresponding results are given in Figure 11. The best scaling range changes in different tasks. For example, in Hopper-v5, $[0.0, 2.0]$ achieves the highest average return while performing sub-optimally in dog-stand. At the same time, the default $[0.8, 1.2]$ consistently performs well in three tasks.



(a) Reward on Hopper-v5. (b) Reward on dog-run. (c) Reward on dog-stand.

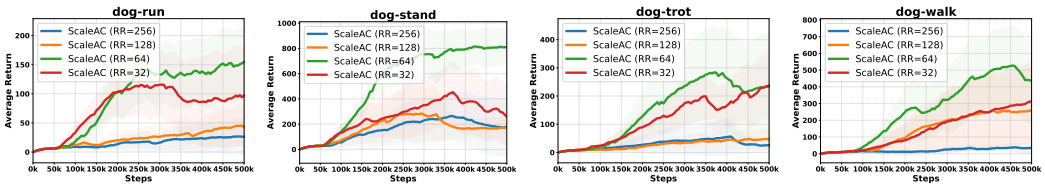
Figure 11: The hyperparameter sensitivity study on the amplitude range $[z_a, z_b]$ of ScaleAC.

G Replay Ratio Scaling of ScaleAC in Dog-Series Tasks

In this section, we show that a higher replay ratio does not always correspond to higher performance. The replay ratio scaling of ScaleAC on dog-series tasks is plotted in Figure 12. It is clear that 64 is the optimal replay ratio on these tasks. Higher values, such as 128 and 256, hurt the performance. Therefore, we infer that there exists a saturation point of the replay ratio that exhausts a model's plasticity when fitting the given replay buffer. Furthermore, experience augmentation in ScaleAC reduces the plasticity cost of each trained sample by forcing networks to find a meaningful pattern with prior constraints instead of overfitting the same sample repeatedly, which coincides somewhat with the findings in the domain of visual reinforcement learning (Ma et al., 2024).

H Comparing Other Network Reset Techniques in ScaleAC

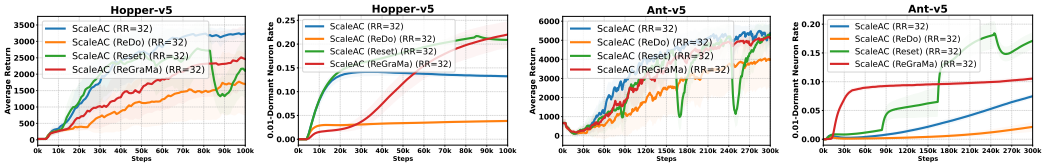
In this section, we replace Shrink & Perturb in ScaleAC with other network reset techniques, such as ReDo (Sokar et al., 2023), Reset (D'Oro et al., 2023), and ReGraMa (Liu et al., 2025), to show that the soft reset mechanism of Shrink & Perturb works best in the high-replay-ratio training setting.



(a) Reward on dog-run. (b) Reward on dog-stand. (c) Reward on dog-trot. (d) Reward on dog-walk.

Figure 12: Replay ratio scaling of ScaleAC on dog-run, dog-stand, dog-trot, and dog-walk in DMC.

ReDo resets a neuron if it is dormant by re-initializing the input weights of dormant neurons and zeroing the output weights of dormant neurons. Reset fully resets networks within a fixed interval, which is the same as SR-SAC (D'Oro et al., 2023). ReGraMa resets an inactive neuron if its normalized gradient value is below a threshold. Specifically, re-initialization in ReGraMa involves resetting incoming weights to the original weight distribution, while outgoing weights are set to zero. At the same time, the inactive neuron's gradient is also set to zero. The comparison results of ScaleAC with each network reset technique are given in Figure 13.



(a) Reward on Hopper-v5. (b) Dormant neuron rate on Hopper-v5. (c) Reward on Ant-v5. (d) Dormant neuron rate on Ant-v5.

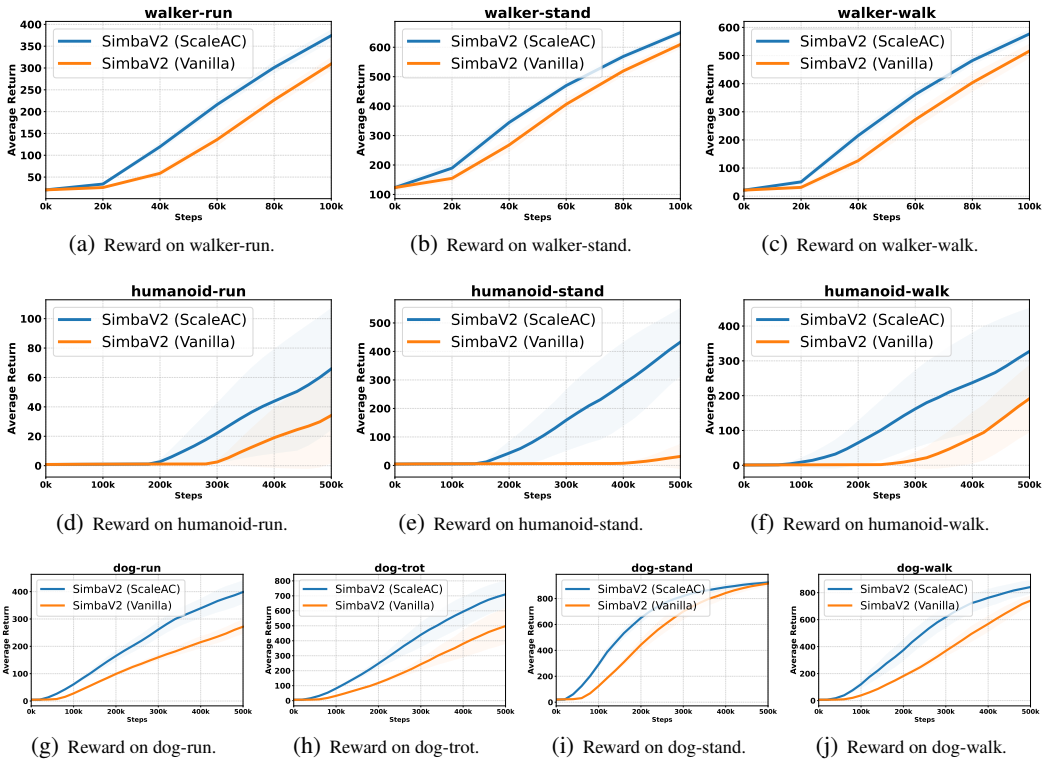
Figure 13: Comparing other network reset techniques in ScaleAC at high replay ratios.

As we see, ScaleAC with the default Shrink & Perturb outperforms ScaleAC variants with other network reset techniques. At the same time, although ScaleAC with ReDo achieves the lowest dormant neuron rates in both Hopper-v5 and Ant-v5, it has the lowest average reward. This is because ReDo re-initializes every dormant neuron, which may have a high gradient magnitude, to hurt the learning capacity of networks (Liu et al., 2025). Moreover, both ScaleAC with Reset and ScaleAC with ReGraMa perform well in Ant-v5, indicating that ScaleAC is robust to the network reset techniques at high replay ratios.

I DMC Results of Extending ScaleAC to SimbaV2

In this section, we report the DMC results of extending ScaleAC to SimbaV2, complementing the MuJoCo results of Section 4.7. We evaluate on the same 10 DMC tasks as in Section 4.3. SimbaV2 (ScaleAC) uses a replay ratio of 4 and SimbaV2 (Vanilla) uses its default replay ratio of 1, with the hyperparameters given in Appendix E. Results are shown in Figure 14, averaged over 6 random seeds with 95% confidence intervals shaded.

ScaleAC improves the sample efficiency of SimbaV2 on all 10 DMC tasks. Following Section 4.3, the walker-series tasks are run for 100k environment steps and the DMC-Hard tasks for 500k. The largest gain is on humanoid-stand, where vanilla SimbaV2 stays close to a zero return while ScaleAC reaches a return above 400. On dog-stand the two variants reach the same final return, and the gain of ScaleAC is one of speed rather than of final performance. These results are consistent with the MuJoCo results of Section 4.7, indicating that ScaleAC also transfers to SimbaV2 in DMC.



(a) Reward on walker-run. (b) Reward on walker-stand. (c) Reward on walker-walk.

(d) Reward on humanoid-run. (e) Reward on humanoid-stand. (f) Reward on humanoid-walk.

(g) Reward on dog-run. (h) Reward on dog-trot. (i) Reward on dog-stand. (j) Reward on dog-walk.

Figure 14: Results of extending ScaleAC to SimbaV2 in the DMC environment.

J Wall-Clock Cost

Table 4 reports the training time of ScaleAC and the baselines on Hopper-v5. Each entry is the mean over 6 seeds, where each seed takes 100k environment steps. All runs were executed on the same hardware.

Table 4: Wall-clock cost on Hopper-v5, averaged over 6 seeds. Each run takes 100k environment steps. The relative column is normalized to SAC at a replay ratio of 1.

Method	Replay ratio	Hours	Seconds / 1k steps	Relative
SAC	1	1.4	51	1.0×
SAC	32	15.2	548	10.8×
REDQ	32	17.5	631	12.5×
SR-SAC	32	21.5	775	15.3×
ScaleAC	32	21.6	778	15.4×
SAC	64	32.7	1178	23.3×
ScaleAC	64	48.4	1741	34.4×
SAC	128	49.9	1798	35.5×
ScaleAC	128	77.4	2786	55.0×
ScaleAC	256	125.3	4511	89.1×

At a matched replay ratio the cost of ScaleAC is close to that of the baselines: it is within 1% of SR-SAC and 23% of REDQ at a replay ratio of 32, and between 1.4 and 1.6 times that of SAC across the three replay ratios shown. The training time is therefore governed by the replay ratio itself rather than by the components of ScaleAC, and the additional cost buys the sample efficiency reported in Section 4.1.

Table 5 reports the same measurement for the SimbaV2 experiments of Section 4.7. SimbaV2 is implemented in JAX. ScaleAC raises the replay ratio from 1 to 4 and the critic ensemble from 2 to

10, which multiplies the gradient computation by a factor of twenty. The measured cost grows by only 5.9 times, because part of the training time is spent on environment stepping and evaluation, which are the same for both configurations and do not grow with the replay ratio.

Table 5: Wall-clock cost of SimbaV2 (ScaleAC) on Hopper-v5 with 100k environment steps.

Method	Replay ratio	Hours	Seconds / 1k steps	Relative
SimbaV2 (Vanilla)	1	0.38	13.6	1.0×
SimbaV2 (ScaleAC)	4	2.23	80.4	5.9×

K LIMITATIONS

There are several limitations of our work. First, ScaleAC is currently validated only on off-policy actor-critic algorithms (SAC and DDPG), and its applicability to on-policy or discrete-action-space methods remains unexplored. Second, the replay ratio is fixed during training, and a dynamically adjusted replay ratio could further reduce computation overhead. Third, the computational cost of high replay ratios with large ensembles is non-trivial: as measured in Appendix J, ScaleAC at a replay ratio of 64 takes roughly $34\times$ the wall-clock time of SAC at a replay ratio of 1, though this overhead is often acceptable in scenarios where environment interactions are the primary bottleneck.

L BROADER IMPACTS

Deep reinforcement learning is a powerful paradigm that can model real-world systems, such as robotic manipulation and autonomous driving. ScaleAC could improve the sample efficiency of off-policy RL algorithms, thus reducing the number of costly environment interactions needed in realistic applications. However, when deployed to real-world tasks, the learning process of RL agents with ScaleAC still requires exploration in the environment, which may lead to unsafe situations and should be handled with care.